School of Mathematics and Systems Engineering

Reports from MSI - Rapporter från MSI

On the Semantic Web language

Benny Gustavsson

Juni
2001

MSI
Växjö University
SE-351 95 VÄXJÖ

Report 01065
ISSN 1650-2647
ISRN VXU/MSI/DA/E/--01065/--

Abstract

The amount of information that is available on the Web has increased enormously in a very short period of time. This increase of information is desirable, but it has made the problems of the Web as an information space more evident. One solution to these problems is to enhance the way information is encoded so as to enable machines to help human users of the Web to locate and deduce information from this vast information space. This "future" Web has been named the *Semantic Web*, and has as a core concept to make information machine processable by using metadata or expressing information directly in a machine processable form, with the final goal of making things easier for the human users of the Web. But to create this Semantic Web there is a need to develop a language that can encode information in a way that enables a machine to enhance the processing of it.

This thesis delivers in the first part a formalization of the Web that is needed in order to develop the language for the Semantic Web since it affects the semantics of the language. The second part of the thesis describes the semantics of the language that is to be used on the Semantic Web. It is considered to consist of three main layers, where each of these layers contributes with primitives that are used by higher layers to create a more powerful language. These three layers together provide the functionality and the expressive power that is needed to be able to represent almost any information that is present on the Web today in a machine processable way.

Table of contents

Errata

Due to the size of this report, minor errors are unfortunately inevitable. Corrections and extensions will be accessible thru `http://purl.org/net/semanticweb/`

Acknowledgment

I would like to thank my supervisor Joakim Nivre for very useful comments, especially concerning formal languages and language semantics.

1 Introduction

The Web has influenced the way people communicate and collaborate. We can publish information on the Web, making it accessible to anyone with access to the Web, or we can use the Web as a source of information to derive new knowledge. The amount of information that is accessible on the Web has increased enormously in a short period of time. This increase of information is a desirable evolution, but it has also made the problems with the Web more evident. Everyone that has used the Web to search for information knows that it is not as easy or as fast as one would like it to be. Using today's search engines to find information is no longer a process of entering keywords and receiving a list of answers: It has also become a process of searching long lists of mostly useless answers to find the good answers.

Since there is so much information present on the Web, a simple search could return a huge list of answers. Of course, if you make a wide search in a big pool of information the list of sources that contain the requested information will always be extensive, and this is generally not a problem – it is an obvious result. The problem on the Web is that narrowing a search on today's search engines leaves out good answers. To narrow the search you normally decrease the set of possible answers by adding keywords that only could be present in the documents that one seeks. But on the Web, as the searches are basically only done with keywords in a global information space, this does not work. Generally, when keyword searches are used there is a problem with synonyms (car vs. vehicle), different languages (bil vs. car), and polysemy (search-*engine* vs. car-*engine*). There is also a problem in searching for e.g., poems, because one seldom uses the keyword "poem" in a poem. This has the effect that searches with keywords are, evidently, not effective to use when searching among information sources that are highly different in terms of language, cultural context, or domain etc. as on the Web. Then, the question of why almost every search engines uses keyword indexes often appears. The reason for using keyword indexes is that it is today the only way to automatically create a search-service by letting robots traverse the Web, collecting information (keywords) and throwing it into a database. This is cheaper than manually categorizing pages, because no humans are involved, and faster because a robot could index a page faster that a human could categorize it. But why *keyword* index? To answer that question one has to consider the formats that are used to represent information on the Web. Some of the most common text formats are HTML, PDF, PostScript, or pure ASCII text (putting multimedia aside for now). These formats all represent the information encoded as a serialization of natural language and are intended for human consumption via a presentation tool. So the only way to make that information searchable is to collect the individual words and use them in a keyword index because a computer sees the string of characters not as a word but as a string of bytes. Indexing a Web page is merely a process of throwing the sting of bytes delimited by the space-character byte encoding into a database. As this should indicate: we cannot today (with limited resources) use computers to create other search-services than keyword services, and this is becoming a big problem. The easiest solution to this problem is not to teach computers to behave as humans and "understand" natural languages, but to change the way information is expressed [Berners-Lee, RM].

There is another problem with the information on the Web that is a bit harder to see: it lacks precision. If I use the name Gore in a text, e.g. "the topic of this page is Gore", how does a viewer know that it is the author's dog that the page is about? The pervious statement lacks precision because it is impossible to know who Gore is in that statement. Before the Web, precision of statements in e.g. books was not that problematic. A book covers a certain topic in a domain and is written in a special

context. Thus, it uses words that have a perhaps implicit but (hopefully) always exact meaning that could be deduced by the books topic, domain and context. The Web could be seen as filled with information from every book that exists. This means that the precision is lost due to the mixing of topic, domains, and contexts and this has a number of negative effects. One is that it becomes harder to find statements (or information generally) about the thing that you search for (e.g. a Web page that contains a specific topic) and it makes it harder to collaborate because the communication could be ambiguous. If I would like to find information about the politician Gore, and uses a basic search engine, the statement in the previous example ("the topic of this page is Gore" – about the dog) would make the page about the dog Gore appear in the list of answers. (If I add the keyword "politician" I will miss pages that contain the "statesman" keyword.)

This problem of precision also relates to another drawback of the information on the Web: it is very time-consuming to use information from more than one Web page to derive knowledge. This is easiest to explain with an example. If I enter a keyword searching for information about the town "Lichavon" I will receive information that has that keyword. But if there is a page (that the search-service trusts) somewhere on the Web that states that "Lichavon" is also called "Objissjf", that information should have indicated to the search-service that I also search for information that has the keyword "Objissjf" (of course offered as an option). This type of deduction has to be manually made by the human that uses the search engines.

The problem of precision is a problem that comes from the fact that information is not adapted to the global information space that the Web is before it being published. We need a way of disambiguating information for it to become truly useful on the Web. It is easy to move information to the Web, but it is almost impossible to preserve its full value if the information is not prepared for being published on the global medium that the Web is. Because the context, domain, and meaning are implicit, they get lost in the universal information space. We need to explicitly express information in a way that explicitly describes the domain, context, and hopefully meaning.

Computers out-performance humans in one sort of activity: their indefatigable ability to search and perform well-defined manipulations on well-defined data. How is it then possible that the service we get from computers is so low when computers indefatigable ability to search and make small inferences is what we need on the Web? The simple answer is that the information on the Web is not well-defined data that computers could process. They can read it, but they don't get it. To a computer this text is simply a string of bytes, with no semantics. If it is a word processor, it might be able to deduce that this is a sentence, but it has now clue what the sentence is *about*. It is a bit frustrating that computers can't help us with such seemingly easy tasks as finding information or making small deductions from the Web. And generally, it would be extremely useful if computers could use information on the Web as if it was hand-coded into their applications. There are of course numerous advantages with having computers that use the Web as their information source; only imagination sets the limits.

Instead of considering the Web as an information space for humans, it could also become a database of well-defined data that machines could use to help us. It could be simple applications such as shopping agents, but also heuristic applications that spans the Web deducing and guessing on information about the stock market and actually buys and sells stocks on our behalf – acting as automatic and personal around-the-clock-brokers. Perhaps this will be the end of the stock market – both terrifying and appealing – but the point is that almost anything becomes possible. With the information pool that exists on the Web today computers can only process it blindly, not making any use of it.

Again the easiest solution to this problem is not to teach computers to behave as humans, but to change the way information is expressed.

All the previously mentioned problems stem from fact that the information is not adapted to the global information space that the Web is, and that it is targeted at human consumption. The information is intended for direct human use, after using some rendering software, and the information is simply text where the semantics is stated through the use of natural languages. If we manage to make part of the information easier for machines to process we could use it better to search for and deduce information from the Web. This does not involve natural language processing, it is done by attaching some well-defined data to the information that computers could use as a hint of how they could use the information. This would help us humans to better deal with the huge amount of information that is present on the Web. If we continue to produce more and more information on the Web, without changing its format, it will become harder and harder to use. We need to change the abstract information space into an abstract space where computers can help manage and use the information.

Ironically, the Web that makes it easier to transmit information has made it more difficult to share information [Sowa, 2000].

1.1 Purpose

The main purpose of this thesis is to describe the computer-processable Web of "semantic data" originated by Tim Berners-Lee called *The Semantic Web* [Berners-Lee, RM] with a focus on the underlying theories and the language for its creation. In order to be able to fulfill this purpose on a formal basis, the theories and definitions that govern the Web need to be examined. Since the Web currently lacks a rigid definition, a necessarily sub-purpose is to offer a formal definition and explanation of the Web and related concepts.

The Semantic Web could be seen as a huge system. In this thesis I will focus on the language architecture of that future system because, in my opinion, this future system is currently being built from the bottom up without a great deal of architectural decisions. To not lose sight of the "goal" I think it is important at early stages in the development process to describe the theoretical properties. Therefore, this thesis is oriented at the architecture of the Semantic Web and should be seen as contributing to the architectural description and theoretical aspects that govern the eventual implementation.

This is not a thesis that will explain the cultural, political, or social effects that the Semantic Web will have. Nor will it discuss why it should be built, by whom it should be built, or the cost of realizing this vision. Further, this thesis will not be an evaluation of standards and techniques that will enable this future Web; it explains the vision in a standardization and technology independent way, concentrating on the core concepts and theories. Choosing not to discuss specific standards or syntaxes, that will eventually implement the vision, and concentrating on the underlying theories and concepts, is in my opinion a better way to describe a future system because there are so many things that are uncertain about the implementation at this point. This way, it can be precise and valuable without making choices on "prospect standards" that the rest of the solution depends on.

1.2 Problem definition

Since the thesis has two general purposes where the first purpose is independent of the second part, but not vice versa there exist two problem definitions. The first problem is oriented around the Web, and the lack of its definitions, that is:

Define the Web and related concepts, such as Uniform Resource Identifiers, resources, and the Web-space itself, in a formal way so that the interpretation of what the Web is, what is "on" the Web, how entities are represented on the Web is unambiguous.

The problem definition above was not originally identified as something that was to be solved in this thesis. Its inclusion grew out of the necessity of formally grounding the solution to the second (originally the only) problem to be examined. The second problem is oriented toward the Semantic Web:

Explain and formally define the semantics and the features of the language to be used on the Semantic Web and the Web philosophies that needs to be adhered to if the language is to be universal.

1.3 Outline

In section 2.1 it is described what has been done on the definition of the Web previously, and since the overall purpose of this thesis is basically to develop a language that could make the Web more meaningful for computers, section 2.2 briefly mentions what has been done in this area.

As the second purpose of this thesis is to explain the semantics of an artificial language, section 2.3 very briefly explains how a language is defined and how the semantics of a language can be interpreted.

The terms *Web* and *Internet* are often used as if they denoted the same thing, and its use and selection of term is often arbitrary. In contrast to the Web, the Internet is well defined, and the definitions used through out is presented in the section 2.4.

The chosen research method that is used is presented in the chapter 3.

Chapter 4 is devoted to the first purpose in this thesis and solves the problems as defined in the first problem definition by formally define the Web and related concepts.

The second problem definition could be loosely dived into "philosophies", which then is to be accounted for in the formal definition of the Semantic Web language. Chapter 5 numerates certain aspect, or "philosophies", of the Semantic Web that needs to be understood. It includes the use of URIs, how precision on the Web is achieved, etc. All these philosophies needs to be observed in order to comprehend the solution, and in fact to be able to achieve the solution.

Chapter 6 explains and defines formally the language that is to be used on the Semantic Web. This section uses definitions and philosophies from especially chapter 4 and 5. Therefore, to understand chapter 6 the previous chapters (4-5) are mandatory. Since chapter 4 is independent, there are no prerequisites to read that chapter (except chapter 2.2).

The discussion in chapter 7 is a summary of the results that chapters 4-6 have produced.

2 Background

This chapter first mentions what has been done in the two areas under consideration in this thesis, i.e. the Web itself and the "prospect" language of the Semantic Web. Then the needed definition of the Internet is explained shortly and finally, as the language to be used on the Web is a formal language, a short explanation of formal languages is given.

2.1 Previous work related to the definition of the Web

There are no formal definitions of the Web and in particular not on how the Web space relates to entities of interest. The closest to definitions of the Web and related concepts are the Uniform Resource Identifier standard [RFC2396] and the informal explanation of concepts [RFC1630]. But these specifications are (obvious) quite ambiguous and imprecise regarding the semantics of the space itself and the "resources" that they should identify. This lack of precision and formalism has led to numerous debates that could have been avoided if agreed formalizations had been used. A source of the debate has also been the introduction of the Resource Description Framework (RDF) [Lassila & Swick, 1999], which uses terms in a conflicting way as compared to the URI standards. Numerous authors have identified the lack of formal definitions, e.g. in [Champin et al., 2001], but no formal definition has yet been provided.

2.2 Previous work related to the language to be used on the Semantic Web

All the problems that the information on the Web gives rise to as caught the attention of numerous and diversified communities. The general goal and desire to better organize and structure knowledge has been the topic of numerous research efforts, e.g. in Knowledge Representation (KR) for a long time. To this end, languages have been developed in the Artificial Intelligence community, such as the Knowledge Interchange Format (KIF) [Genesereth & Fikes, 1992] with the intent to capture knowledge in a computer processable form. KIF is a logic language that has been proposed as a standard to describe things within a computer system, e.g. expert systems, databases, intelligent agents, etc. and to exchange knowledge between systems. Although these languages developed in the AI community were not designed to be used on the Web they are still interesting, but they have properties that make them inappropriate for direct use on the Web. Chapter 5.7 briefly describes why, generally, languages developed in the AI community are not directly usable on the Web. Generally, these languages are developed with a "closed world" assumption and an assumption that they have to be the "center of all knowledge".

To introduce computers to the Web space to help us humans was first originated by the inventor of the Web, Hyper Text Transfer Protocol (HTTP), Hyper Text Markup Language (HTML), and URI, Tim Berners-Lee. He has always considered the Web to be a space where both computers and humans should collaborate [Berners-Lee, 2000]. Expressing information so that a computer can process it is something that he stated early in the development of the Web. He has also coined the name of the "future" Web: *The Semantic Web* where computers help the humans to cope with the vast amount of information present on the Web. As the head of the Word Wide Web Consortium (W3C) he has initiated a few projects to develop languages to be used on the Web targeted at computers. These languages were to take the best ideas from the KR community and "webify" them. One of the major developments by W3C is the Resource Description Framework [Lassila & Swick, 1999] (RDF). RDF is a complete framework aimed at describing Web resources with metadata in a way that enables automated processing of these resources. Extensions to this language have also been

developed at W3C, e.g. the Resource Description Framework Schema Specification (RDFS) [Brickely & Guha, 2000]. There has also been some experimental development, e.g. Metalog [Marchiori & Saarela, 1999] that is a system, also developed at W3C that allows users to write metadata, inference rules, and queries in English-like syntax. The system, developed as a side project at W3C, is a layered system where the first layer enriches the RDF model by providing logical relationships and building up complex inference rules that encode logical reasoning [Marchiori & Saarela, 1999]. Even more experimental is a language called Notation 3 (N3) [Berners-Lee, N3] that is a language built with a simplified syntax of RDF that is targeted at developers for experimenting with features available on the Semantic Web.

There are languages developed outside the W3C too. SHOE [Heflin et al., 2000] is a language aimed at the Semantic Web developed at the University of Maryland that is not built on top of RDF; instead it is built by extending HTML. Even so, it is still very useful to examine the logical capabilities of SHOE and see what design decisions and compromises that the language design faced. SHOE combines features from knowledge representation, Datalog, and ontologies, but also from predicate logic and frame systems [Heflin et al., 2000]. Ontology Inference Layer (OIL) [Horrocks et al., 2000] developed at MIT, Stanford, Bell Labs, and other institutes, is a Web-based representation and inference layer for ontologies. The language is defined as an extension to RDF. DARPA Agent Markup Language (DAML) [Hendler, 2000] is a language that is also an extension to RDF. DAML was initiated by the Defense Advanced Research Project Agency of DoD in United States. DAML and OIL has produced a language specification (DAML+OIL) that combines the efforts. A quite different (visual) language is the development of Concept Graphs (CG) [Sowa, 2001]. It is mainly a visual language but it has serialization syntax in XML.

All these languages are fairly similar regarding the semantics, but differ in terms of the syntax. These languages generally explain the semantics by "describing the semantics of the syntax". In this thesis the focus is on the semantics, ignoring the syntax, and I therefore try to use the, most often equal, semantics of the above-mentioned languages in a uniform way. The language described in the latter part of the thesis is therefore highly influenced by all these languages, but does not consider the syntax of any language. In this way the core semantics of the language could be revealed, and the mapping to syntax becomes fairly straightforward.

2.3 The Internet and internet

It is quite common to mix up the terms *Internet*, *internet*, and the *Web*. This section will reproduce the definitions of the terms *Internet* and *internet* (the difference is in the capitalisation) that will be used throughout. Chapter 4 will then define the Web.

An internet(-work) is a network of networks [Halsall, 1995]. This means that if you connect two or more networks the combined network is an internet(-work).

As a definition of the term "Internet" I will use the one proposed by Federal Networking Council (FNC). In their FNC Resolution: Definition of "Internet" [FNC, 1995] they state that:

"'Internet' refers to the global information system that –

(i) is logically linked together by a globally unique address space based on the Internet Protocol (IP) or its subsequent extensions/follow-ons;

(ii) is able to support communications using the Transmission Control Protocol/Internet Protocol (TCP/IP) suite or its subsequent extensions/follow-ons, and/or other IP-compatible protocols; and

(iii) provides, uses or makes accessible, either publicly or privately, high level services layered on the communications and related infrastructure described herein."

When I use the term Internet (with capital I), I refer to the US government funded internet(-work) based on the TCP/IP suite as described above. The term internet (with lower-case "i"), which is an abbreviated name for internetwork, is a name for the type of topology that refers to the collection of interconnected networks. Thus, the Internet is an internet, but an internet does not have to be the Internet.

At this point I will not explain further how the different protocols and standards that enable the Internet work (see e.g. [Halsall, 1995] for further explanation). Instead I will assume that the Internet provides its users with the following capability: The Internet enables computers to communicate irrespective of their individual geographical location, as long as they are properly connected to the Internet.

2.4 Language, Syntax, and Semantics

In the context of computers, the use of *language* is generally regarded as: "Any of numerous systems of precisely defined symbols and rules for using them that have been devised for writing programs or representing instructions and data" [Oxford English Dictionary]. By being general and only describing the common use of the word, it does obviously not provide the formality that is needed in latter parts of this thesis. To that end, there is a need to use a more formal explanation on what a computer-oriented language is. Seeing that a language is governed by its definition it is easiest to start by examining how a language is defined. Note that this section is oriented at logical languages since the chapter about the Semantic Web language (6) will use first-order predicate calculus (FOPC) to explain the semantics and the features of the language (i.e. the Semantic Web language is the *object language* and FOPC is used as a *metalanguage*). Thus, this section will explain the syntax and semantics of FOPC for two reasons: (1) to get some insight of how the syntax and semantics of a logic language is created, and (2) to provide a repetition of FOPC that will be need to understand chapter 6.

An *alphabet* Σ is a nonempty set of *symbols* [Ebbinghaus et al., 1996]. Finite sequences of symbols from the alphabet Σ are called *strings* over Σ. Σ^* denotes the set of all possible strings over Σ. For a given alphabet Σ, any subset of Σ^* is called a *language* over Σ.

The alphabet of a FOPC language contains the following symbols [Ebbinghaus et al., 1996]:

1. $v_0, v_1, v_2, \ldots$ (*variables*)
2. $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$
3. $\forall, \exists$
4. $\equiv$
5.), (
6. (a) for every $n \geq 1$ a set of n-ary *relation symbols*
   (b) for every $n \geq 1$ a set of n-ary *function symbols*
   (c) a set of *constants*

In the following, A will stand for the symbols listed in 1 through 5, and S for the set of symbols from 6. S determines a FOPC language [Ebbinghaus et al., 1996], and $A_S := A \cup S$ is the alphabet of this language and S its *symbol* set.

Given a symbol set S, certain strings over A_S are called *formulas* over the FOPC language determined by S. To define the set formulas of a FOPC language one first needs to define the set of strings that are *terms*. *S-terms* (the terms of the FOPC language determined by S) are those strings in A_S^* which can be obtained by applying the following rules [Ebbinghaus et al., 1996]:

1. Every variable is an S-term
2. Every constant in S is an S-term
3. If the strings $t_1,\ldots, t_n$ are S-terms and f is an n ary function symbol in S, then $f(t_1,\ldots, t_n)$ is also an S-term.

T^S denote the set of S-terms. Now that the terms are defined, it is possible to define *formulas*.

S-formulas are those strings of A_S^* which are obtained by applying the following rules [Ebbinghaus et al., 1996]:

1. If t_1 and t_2 are S-terms, then $t_1 \equiv t_2$ is an S-formula
2. If $t_1,\ldots, t_n$ are S-terms and R is an n-ary relation symbol in S, then $R(t_1,\ldots, t_n)$ is an S-formula
3. If φ is an S-formula, then $\neg\varphi$ is an S-formula
4. If φ and ψ are S-formulas, then $(\varphi\wedge\psi)$, $(\varphi\vee\psi)$, $(\varphi\rightarrow\psi)$, and $(\varphi\leftrightarrow\psi)$ are also S-formulas
5. If φ is an S-formula and x is a variable, then $\forall x\varphi$ and $\exists x\varphi$ are S-formulas

The formulas derived using 1 and 2 above are called *atomic formulas*.

At this point the syntax of the FOPC language is defined, but how about the "meaning", semantics, of a formula? Meaning is a word that it rather hard to use, and the philosophical discussions are left out. The FOPC language has a certain "meaning" attached to certain symbols in the language that has been defined by a truth table, e.g. the meaning of the connective $\wedge$ is specified by a truth table and this meaning is then attached to the symbol. But how is the meaning of e.g. a binary relation R specified? Imagine the formula $\forall xR(x,x)$ (i.e. xRx) from some language. What does this formula mean? Nothing, at least considered in isolation. But if we associate a domain with the language, e.g. the natural numbers **N**, and specify that x is an element of that domain, $\forall x$ means "for all elements in **N**", and that we further specify that xRy means that x is divisible by y, the formula $\forall xR(x,x)$ has a certain meaning. Further, since all natural numbers are divisible by itself, the formula $\forall xR(x,x)$ becomes a true statement, but only using this specific domain and interpretation of R. The following explanations and definitions are to formalize the use of *domain*, *interpretation*, and *model*.

A *S-model* is a pair $M = (D, \alpha)$ with the following properties [Ebbinghaus et al., 1996]:

1. D is a nonempty set, the *domain* or *universe* of M
2. α is a map defined on S satisfying:
   (a) for every n-ary relation symbol R in S, $\alpha(R)$ is an n-ary relation on D
   (b) for every n-ary function symbol F in S, $\alpha(F)$ is an n-ary function on D
   (c) for every constant c in S, $\alpha(c)$ is an element of D

An *assignment* in an *S-model M* is a map $\beta : \{\forall v_n \mid n \in \mathbf{N}\} \rightarrow D$ of the set of variables into the domain *D* [Ebbinghaus et al., 1996]. An assignment basically maps all the variables to objects in the domain. Using assignment it is possible to define what an interpretation is: An *S-interpretation I* is a pair (M,β) that consists of a model *M* and an assignment β in *M* [Ebbinghaus et al., 1996]. What is needed now is a way of stating that a formula is true under an interpretation. As a simplified example, the formula "$\overline{2} + \overline{2} = \overline{4}$" is true in the model of natural numbers with addition if 2+2=4 (e.g. $\overline{2}$ is mapped onto the *number* 2 and the binary function symbol "+" is mapped onto the binary function "addition" on natural numbers). Interpretation is the art of relating syntactic objects and states of affaires "in reality" [Dalen, 1980].

Now it is time to formally define this "relation of truth" between formulas in the language and the objects in the domain. Prior to defining the relation there is a need to associate with every interpretation *I* and every term *t* an element $I(t)$ from the domain. Therefore, in the following, if $I = (M,\beta)$ [Ebbinghaus et al., 1996]:

1. for a variable *x* let $I(x) := \beta(x)$
2. for a constant $c \in S$ let $I(c) := \alpha(c)$
3. for an *n*-ary function symbol $f \in S$ and terms $t_1,\ldots, t_n$ let $I(t_1,\ldots, t_n) := \alpha(\mathrm{f})(I(t_1),\ldots, I(t_1))$

Now it is possible to define the *satisfactory relation* (|=). For all interpretations $I = (M,\beta)$ we let:

$I \models t_1 \equiv t_2$ iff $I(t_1) = I(t_1)$
$I \models (\phi \wedge \gamma)$ iff $I \models \gamma$ and $I \models \phi$

The rest is left out since it follows in similar fashion and on account of this sections scope.

This relation is used to state that a particular interpretation *I* is a model of a formula φ. If *I* is a model of φ one says that *I* satisfies φ, or that φ hold in *I*. This means that $I \models \varphi$ if and only if φ becomes a true statement under the interpretation *I*. As this should indicate, a formula in a language could be true, or hold, in an interpretation I_1 but be false in another interpretation I_2. This simplified explanation should give some insight of how the semantics of a FOPC language is specified.

The semantics of *PC* is very well defined, but the semantics presented here (particular regarding satisfiability) is simplified on account of this sections scope. For further details see e.g. [Ebbinghaus et al., 1996], [Dalen, 1980], or [Hansen, 1994].

3 Method

The first problem of formally defining the Web requires an extensive literature study to collect as much knowledge as possible about the previous achievements, but also to identify the problems with the current knowledge and standards in this area. The main material used to solve the first problem is the standards of today [RFC1630] [RFC2396], but also design decisions document from the World Wide Web consortium [W3C DesignIssues]. As the input source on the problems with the standards and the knowledge of the Web today have been mostly mailing lists, e.g. [RDF-Interest Group] [RDF-Logic] [URI], occasional papers describing the problems e.g. [Champin P et al., 2001], and debates on dedicated IRC channels. From the existing informal definitions and all the information on design issues and problems, it is then a process of formalizing the definitions by using formal mathematics. Since the standards are quite informal and ambiguous there was also a creative element in the process. The formalizations where tested for inconsistency and valued by appropriateness and the explanatory power. Further, the formalizations were also made public for discussion in the community.

The second problem also calls for a rigorous literature study to collect information about what has been achieved so far. Since the Semantic Web is new as a topic of study, most of the interesting information regarding the Semantic Web is found on the Web. But detailed and rigorous information on certain general aspects of the Semantic Web, e.g. concerning the logic, can be found in the Artificial Intelligence literature. Since the artificial language that is to be explained in this thesis is developed for the Semantic Web there are certain aspects that needs to be fully understood, which distinguishes this language from other formal languages. Therefore the literature study was first oriented at collecting knowledge about the philosophies of the Web and the Semantic Web that have a direct effect on the languages. Only after collecting this knowledge is it possible to look at existing prospect languages to see the core semantics of them and if they are capable of being the language of the Semantic Web. From the collective knowledge about the existing languages and the knowledge about important design issues about the Web, the semantics of the language to be used on the Semantic Web was then derived. The process was both a creative and investigating process since the semantics of the current languages was missing or poorly defined. As with the first problem, the formalizations where tested for inconsistency and valued by appropriateness and the explanatory power.

4 Uniform Resource Identifiers and the World Wide Web

The *World Wide Web* (Web) is an abstraction that "hangs" over the Internet to provide a uniform addressing of network retrievable objects. The Web has made it possible to access information published on the Internet in the same way irrespective of the service that publishes the information. It is possible to access information on an FTP-server in the same way as on an HTTP-server thanks to the Web and the concept of *Uniform Resource Identifiers* (URIs) [RFC2396] in particular. From the user's perspective, there is no difference in clicking on a link in a hypertext document that point to an information entity on a FTP-server, compared to a link to an information entity on a HTTP-server. In other words, the Web lies as a connective tissue between the users and the Internet. But the use of the Web is starting to change. It is now becoming more than a tissue between the users and the Internet, and the reason for this is one of it properties: it is a universal *namespace*. It is the property of being a universal namespace that makes it possible to create links between documents, links that have the same function irrespective of its context. A link to `http://server.com/doc` works in the same way irrespective of the "clickers" geographical location, use of language, or culture. This is something that did not exist before the Web and the Internet, and this is what people are starting to notice. In a way, the use of the Web is changing from a connective tissue between people and the Internet, to a connective tissue between people.

The Web is usually "defined" as an abstract information space, e.g. in [Berners-Lee, WA]. Within this space we can navigate and seemingly enter a world of information. At first, this might seem to be more of an imagination and an indefinable creation than something built by letting a set of highly definable and formal computers communicate over a network. But the Web is very well definable. This section takes away the fuzzy atmosphere that sometimes surrounds the word *Web* by providing formal definitions of the Web and related concepts. Also, as the use of the Web has changed, the definitions need to be examined from this perspective too.

The definitions presented here are to a certain extent new, while others are extensions to present definitions of related Web concepts. There are two main reasons for the necessary evolution in the definitions: (1) the use of URIs has been extended, as described above, from the initial use as "the syntax used by the World-Wide Web initiative to encode the names and addresses of objects on the Internet" [RFC1630] to also describe any type of entities – not only Internet objects –, and (2) there is some confusion about how to interpret the concept of URI, as e.g. noticed by [Champin et al., 2001]. Given that the URI concept is going to be used more extensively and will have an extremely important role in the future, its theories has to evolve and be more well defined as the use of them has changed and will increase. Another reason for this formalization is that the rest of this thesis will highly depend on these definitions as it describes the "future" Web. The definitions presented here do not conflict with the common use of URIs, only how they should be interpreted theoretically.

4.1 The Web and URIs

The Web is formally an abstract *namespace*. This namespace is effectively created by the *Uniform Resource Identifier* [RFC2936, RFC1630] (URI) concept. The URI-specification defines an infinite set of strings that are the possible set of valid URIs. This creates an abstract namespace wherein each point in the namespace is a unique URI and using these URIs, we can navigate in this abstract namespace.

Definition 1 – Uniform Resource Identifier
A URI is a point in the URI namespace as defined by the URI specifications RFC2936

Definition 2 – The Web
The Web *is* the abstract URI namespace

These definitions could at first seem imprecise, but they are not. To fully understand the Web one needs to examine the theoretical aspects of the URI-space and concepts used inside this space.

A URI is a point in the namespace that the URI standard creates, as said previously. Equally: A URI is an *element* in the *set* of possible URIs. The set *U* defined below (4.1) is the set of possible URIs at a specific time. This set is created by a set former that basically defines the set to be all valid URIs. The set *S* contains all the possible stings that can be created and where each symbol is a *UNICODE* symbol, and the predicate *URI*(*x*) denotes that *x* is a valid URI according to present standard at a specific time.

$$U = \{x \mid x \in S \wedge URI(x)\} \tag{4.1}$$

Another way of expressing this is that *U* is a language over the UNICODE symbol set.

The set *U* of all possible and valid URIs at a specific time will be called the *URI-space*, *URI-set*, or the *Web* depending on the context. A specific URI is a *point* in the abstract URI-space and equally a *member* of the URI-set. Further, the set *U* is infinite since there is no length restriction on URIs.

There exist two types of URIs: *Uniform Resource Locators* [RFC1736, RFC2936, RFC1630] (URL) and *Uniform Resource Names* [RFC2141, RFC2936, RFC1630, RFC1737] (URN). A URN differs from a URL in that its primary purpose is said to be persistent labeling of an entity with an identity [RFC2396] while a URL identifies the entity by its primary access method [RFC2396]. And a URI could be a URN, URL, or both [RFC2396]. Thus, the URI-set is a superset consisting of the *URL-set L* (or the *URL-space*) and the *URN-set N* (or the *URN-space*). A URI is a member of the URL-set if there is a standard that defines the URI to be a URL. Similar, a URI is a member of the URN-set if there is a standard that defines the URI to be a URN. Generally, a standard that defines URLs or URNs governs a set of URIs. Thus, at a specific time *t* there is a set of standards that defines a set of "classes" of URIs that constitute the URL (*L*) and URN (*N*) set. The set *L* is the set of *possible* URLs and *N* is the set of *possible* URNs, both at a specific time, i.e. governed by a specific set of standards. The relationship with these spaces and *U* is as follows:

$$U \supset (L \cup N) \tag{4.2}$$

4.2 state that the union of the URL-set and the URN-set is a proper subset of the URI-set at a specific time. This means that there are possible URIs that are not a member of the set of possible URNs or URLs. The set *U* is an infinite set of strings governed only by the URI standard, but *L* and *N* are created by a set of standards. Thus, further subsets could be included in the URI-set to handle future sets without disturbance. This is important for the Web's evolution [Berners-Lee, AU]. Figure 4.1 illustrates the different spaces in a Venn diagram.

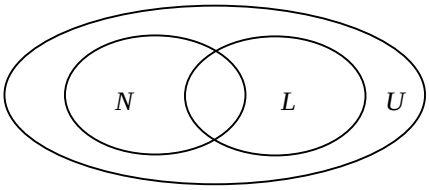


Figure 4.1

There are some concerns in the community about how URNs and URLs relate. The standard [RFC2396] says that a URI could be a URN a URL or *both*. That is something that has stirred up things in the community, but this is not investigated in this thesis. What is also important is that a URI could be outside these sets (*N* and *L*) but its meaning is at present time undefined, e.g. `sdfh://jim@web` is a valid URI but is neither a URL or a URN at present time.

There exists a big set of *protocols* (basically an access mechanism) that can be used to send, access, or retrieve a representation of some information through the use of the Internet, e.g. Hyper Text Transfer Protocol (HTTP) and File Transfer Protocol (FTP). Adding to that, there exist different versions of a specific protocol type, e.g. HTTP1.0 and HTTP1.1. Throughout this section, the set *P* will be considered to be a set of algorithms that implements all standardized protocols at a specific time. This set will be called the *protocol-algorithm set* or simply *P*. All the protocols that this set "implements" should be considered to be standard (e.g. standardized by IETF) at a specific time. Every algorithm in this set is possible to implement in a computer processable form.

As mentioned in the introduction, the use of the Web has gone past being only an artifact for making the access to the objects on the Internet easier. It is now also a space wherein abstract and physical entities are being represented. Assigning an identifier to an entity is a relation (possibly a function, e.g., social security number) from a set of identifiers to a set of entities. On the Web URIs are used as identifiers. If it is not possible to represent the entity "digitally" and to retrieve a representation of the entity, the algorithm that maps the relation becomes abstract. The use of the Web will in the future be as a global namespace in which we all will have an identifier. Therefore, the concept and definitions of the Web has to be able to handle these abstract algorithms. An abstract algorithm is an algorithm that is not possible to implement on a computer. Naming a dog with a URN is not possible to implement on a computer a present time. The set *A* is considered to include all the abstract algorithms that exists in order to map a URI to a entity, which should be identified on the Web by that URI. (An abstract algorithm could exist inside someone's head as a fragment that states the "my dog has the identifier `urn:dogy:buck` on the Web") The set *A* will be called the *abstract-algorithm set*. This set is not usable to a machine, and generally, only the thing that controls the URI or who makes statements knows if, how, and to what entity the relation maps the URI. The two sets (abstract-algorithm set and the protocol-algorithm set) are disjoint, i.e. $P \cap A = \varnothing$.

To connect a URI to the protocol algorithm or the abstract algorithm that are needed to retrieve a representation of some entity or to name something with a URI, respectively, there exists a relation R_1 between the set *U* of URIs and the two disjoint sets *P* and *A*. (It is a relation since e.g. a URL that uses HTTP is related to both an algorithm that implements version 1.0 and on that implements version 1.1.) In rule 4.3 this relations is defined.

$$R_1 \subseteq U \times (P \cup A)\} \tag{4.3}$$

This relation R_1 is illustrated in figure 4.2.

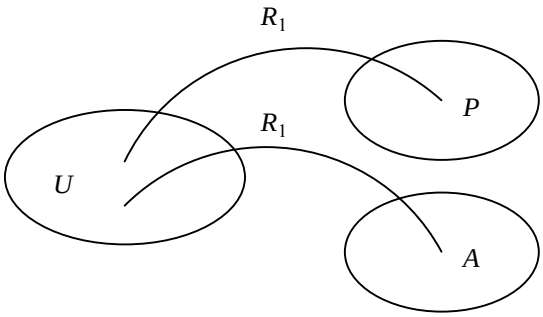


Figure 4.2

This relation does need some explanation, and the easiest way is to do that with an example. Imagine an ordinary Web browser that receives an URI from the user, e.g. `http://server.com/doc.html`. The first thing that the Web browser does is that it tries to figure out what type of URI the user entered. The browser looks at the first part, `http` and concludes that it is a URL and that the protocol that the browser must follow to access/receive a representation of the information entity is the HTTP protocol. The browser knows an algorithm (which is an element in *P*) for the HTTP1.0 protocol, and therefore tries to use that. Thus, the browser has to know the relation between a URI and an algorithm it knows by looking at the URI.

To make this easier for the application, the URI space is divided into subsets dependent on the type of access methods [RFC1630]. This division is created by the URI *scheme* [RFC2396], which is used to declare further namespaces inside the URI-space. Generally, the scheme part of a URI is the part before the first ":" character.

First, the URI space is divided into a URL and a URN space, as said previously. Then, the URL-space is further divided into a number of spaces. There exist as many spaces at a specific time as there exists access methods to entities (e.g. HTTP, FTP, etc.) at that time. Since the URN space currently only has the `urn:` scheme that is the only subspace at the moment. This subset division of the URI space is depicted in figure 4.3 below.

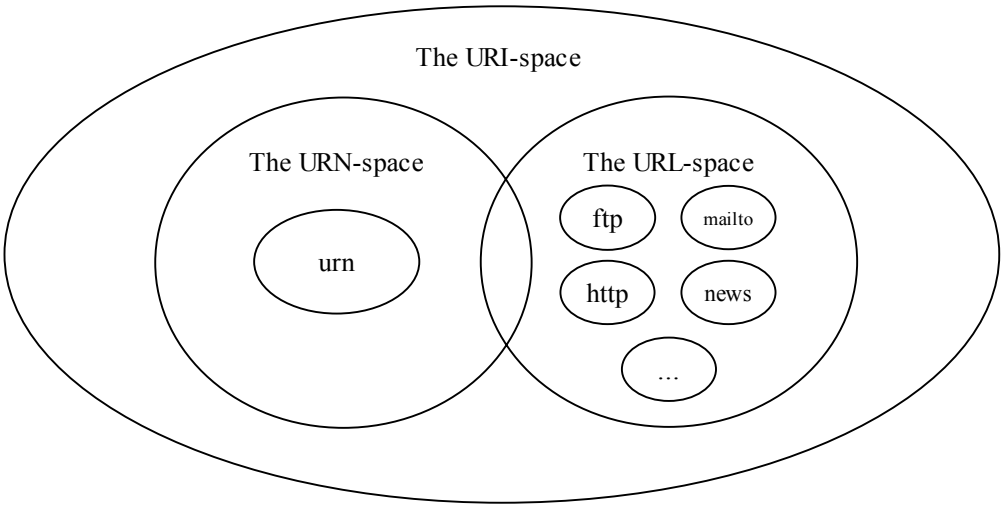


Figure 4.3

As mentioned previously, the URL and URN spaces are created by standards that state that a particular type of URI is governed by a particular standard. Therefore, to be exact, the spaces *L* and *N* actually don't have the interspaces that figure 4.3 illustrates.

[RFC1630] states that a URL is a form of URI that expresses an address that maps onto an access algorithm using network protocols. But there exists URLs, e.g. `file:` that does not use a network protocol.

So far, the concept of URIs as points in a subdivided space and the relation between a URI and a protocol or abstract algorithm has been discussed. But the use of the Web is about representing entities on the Web. The entities are the things that we want to make public or to name in the URI space. Therefore the relation between the Web and the entities needs to be examined.

There exists a domain *D* of interest. *D* is considered to be a set of entities that we are interested in. The element of the set *D* could be files, books, or people, etc. To make a relation between the entities that we are interested in and the relation between a URI and the mapping, we need another relation. This relation R_2 makes the mapping onto the entities of interest.

$$R_2 \subseteq R_1 \times D \tag{4.4}$$

This relation is illustrated in figure 4.4.

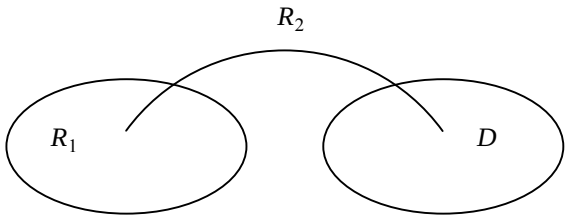


Figure 4.4

A URI is related (relation R_1) to a protocol or abstract algorithm (set *P* or *A*) and this relation is determined by the application that receives the URI. The URI and the algorithm *together* (R_1), is part of a relation (relation R_2) to an entity in the domain *D*. This relation is "explored" by the application by using the URI and the protocol algorithm.

What does all this mean? The main point of this section is to enhance the understanding of what a URI *is*, what it *represents*, and how it should be *interpreted*. At this stage a URI *is* a point the URI-space (the Web) but what it *represents* and how it should be *interpreted* needs some further explanation.

A URI stands in a certain relation to an algorithm and that relation stand in another relation to the set of entities of interest. Thus, a URI as a point in a space does not represent anything, but a relation between a URI and an access algorithm could be used to represent an information entity on the Web. Figure 4.5 below gives an overview how a URI indirectly could be used to represent an entity on the Web.

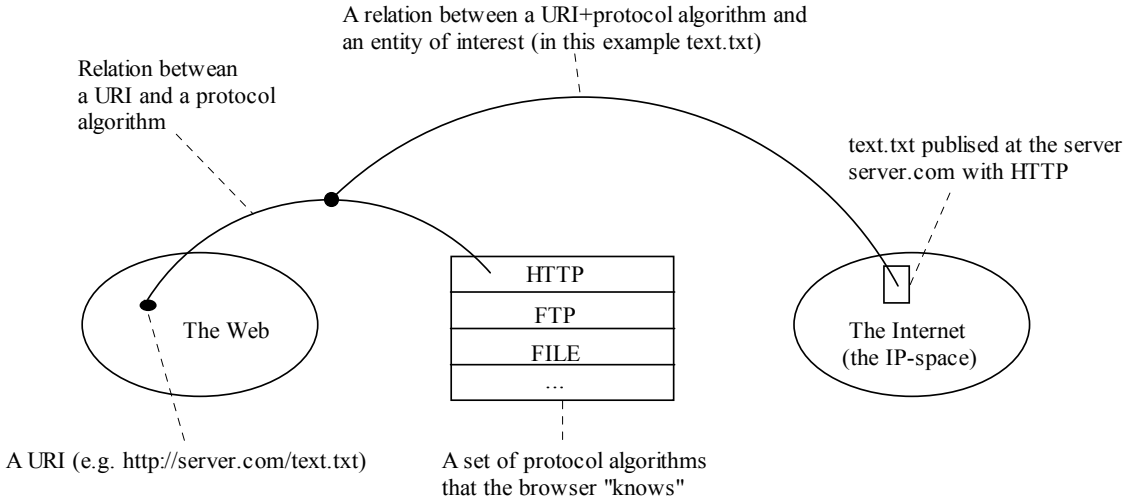


Figure 4.5

The image above illustrates the important fact that a URI does not represent (i.e. has a relation to something) anything *directly*. A URI without a related algorithm does not represent anything, apart from being a point in the URI space. Its creator generally only knows what is represented by a URI. In the latter parts of this thesis, there is a need to fully understand what the URI represents, or identifies. Often, there is some confusion in the community that a URI identifies an entity (e.g. a file) but strictly speaking this is not true. E.g. the URI `http://purl.org/net/semanticweb` is not an identifier for a web page: it is simply a point in the URI space. The only way to strictly interpret a URI is to regard it as a point in the URI-space.

To use the definitions directly is too cumbersome when writing or talking about the Web, so there is a need for an abstraction. This abstraction that is used in this thesis is a *web-resource*. The choice of word is not arbitrary. The word *resource* is used in e.g. URI, URL, and URN standards. Unfortunately, the use of the word resource is the source of the most common confusion and unnecessary debate in the Web community. The problem started, or was identified, with the introduction of the Resource Description Framework [Lassila & Swick, 1999] and its explanation of what a resource is and how it is identified. Central to this conflict is the use of *fragment ID*s. A fragment ID makes it possible to create links to information units inside the document or to create different views of a document. The anchor ID in HTML offers such a feature, where it is possible to link to different sections of a HTML document. As this implicitly indicates, the fragment ID is dependent of the (MIME) type of document that the link is to. Further, a fragment identifier is appended at the end of a URL, e.g. http://server.com/doc1#part1. In this example #part1 is the fragment identifier. Strictly speaking, a fragment identifier is not part of the URI [RFC2396].

When RDF was introduced, it considered a URI with an optional fragment identifier to identify a resource [Lassila & Swick, 1999]. Thus, RDF regards http://server.com/doc1#part1 and http://server.com/doc1#part2 as two different resources. This is one conflict with the use of URIs and fragment IDs between the URI specifications and the RDF specification.

The source of most of the debates of URIs and resources is simply that the URI concept is not fully understood. In [RFC2396] resource is explained as: “anything that has an identity”, and that a URI identifies such a resource. Further, [RFC2396] states: “The resource is the conceptual mapping to an entity or a set of entities …“. In my opinion, there is not a clear understanding of the quotations above if the resource is the

representation of the entity or if it is the URI and the mapping. In [Champin et al., 2001] they do not consider the resource to be the URI *and* the mapping.

The solution to avoid this conflict is an abstraction. Even if a fragment ID is strictly not part of the URI it still serves the same purpose: a URI is a point in the URI space, and a fragment ID is a point in, or view of, an information entity. Therefore, creating an abstraction that treats URI and fragment IDs uniformly is useful and simple.

The definition of *web-resource* that is given here is formal and should mainly be used in the context of the Semantic Web but could be useful in other areas too.

A *web-resource* is an artifact that is used to describe something that is *represented* on the Web. It is not possible to move physical things into an abstract space, so the web-resource, which is abstract, is used to represent the entity in this URI space. A web-resource is a pair where the first element is a URI with an optional fragment ID and the second element is a function (abstract or formal) that maps the URI to an entity. Important to notice is that the web-resource is a URI *and* a function, not the actual entity that the function together with the URI maps onto. This is important because e.g. the exact same web-resource could represent two different entities at two different moments of time. The use of web-resource is less exact, but much easier to use plus that it abstract the use of the fragment identifier.

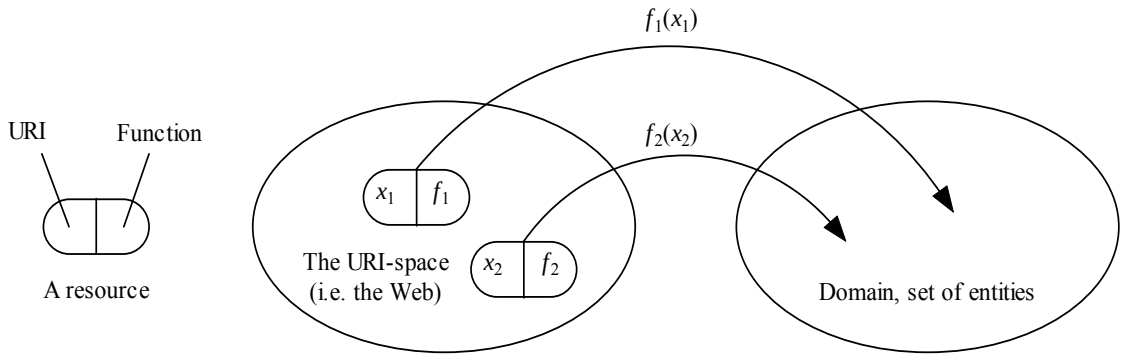


Figure 4.6

Figure 4.6 above illustrates, a bit simplified, how two web-resources on the Web map onto the entities that they represent on the Web. A bit more philosophically, a web-resource is used as a "port" from the Web-space to other possible disjoint spaces (e.g. the real world, the Internet – the IP-space –, etc.). It is the web-resources that we use to represent *entities* in this abstract space – the Web.

Definition – *Web-Resource*
A web-resource is an abstract artifact that is used to talk about something being represented on the Web. A web-resource is a pair (x, f) where x is an element from the set U of possible URIs plus an optional fragment ID, and f is a specific function drawn from the set F of possible functions

F is different from P and A that was previously defined in that the function that it contains is an abstraction that maps a URI to the entity directly. Thus, F abstracts both the relations R_1 and R_2 into a function and incorporates the fragment issue, i.e. the function could be used to map a URI into a document directly. This makes the use easier.

A web-resource is *identified* by its URI element. The interpretation of an identifier used here is the same as [RFC2396] in which it is considered to be an object that can act

as a reference to something (an entity) that has an identity. The interpretation of *entity* that is used thought is: "That which constitutes the being of a thing" [Oxford English Dictionary]. This means that abstract or physical things are entities, and that a web-resource could be used to represent anything.

I like to stress that the web-resource is used to represent an entity on the Web; therefore it is important to consider the mapping from the web-resource (which is identified by its URI) to the entity. In [Berners-Lee, GR], Berners-Lee writes two statements that conceptually correspond to my theory but uses a different vocabulary: "A URI represent a resource" and "A 'resource' is a conceptual entity" – this is, with my definitions, equal to: a URI identifies a web-resource that represent an entity on the Web. Since the Web is an abstract space we can't move entities into that space. Instead, web-resources are used to represent entities on the Web. This is something that becomes very important in latter parts of this thesis, and that is the main reason for making this distinction.

There is a common misunderstanding, e.g. in [Champin et al., 2001], when it comes to URNs in that some think that labeling an entity with a name does not include a mapping, and therefore it would make them more "persistent". "Labeling" a dog with the name "Buck" is a mapping from the set of names to the set of entities – in this case the dog. The mapping is of course an abstract function inside somebody's head.

4.2 Syntax and examples

To connect all these theories to see the actual use of them on the Web it is useful to look at some URI examples, but first a short look at the URI syntax. In [RFC 2396] the URI is considered to consist of two parts: a *scheme* part and a *scheme specific* part.

```
URI = <scheme>:<scheme-specific-part>
```

The `<scheme>` part decides in some cases which function *f* that maps the URI to the entity. If the `<scheme>` is e.g. `http` then the URI is mapped to the algorithm that is decided by the http specification. The `scheme-specific-part` is completely governed by the specification for the `scheme`, i.e. the function.

Four examples:
(1) `http://www.msi.vxu.se/personal`
(2) `mailto:bgudi97@student.vxu.se`
(3) `urn:ietf:rfc:2396`
(4) `file://docs/swintro.txt`

The first example (1) is a URI that is also a URL. Generally, the scheme specification determines if a URI is a URL or a URN. Since the schema specific part is determined by the scheme, the `//www.msi.vxu.se/personal` part is restricted under the `http` specification. The second (2) example is also a URL (think of it as a way to access a mailbox's in-folder) and the part `bgudi97@student.vxu.se` is restricted under the `mailto` specification. The third (3) example is a URN. `urn` is a reserved space for URNs. In this example the URN is an identifier for the RFC2396 document. The last (4) example is a URL that accesses a file on the local disk, relative to where it was "entered". This is just to show that a URL does not need always to be a network action. In the case of the `file:` protocol it can be viewed as an abstraction for an object that is totally determined by its location, i.e. its identifier is not "content dependent".

To get some of the philosophy behind the design of URI the following is an important view: "in order to abstract the idea of a generic object, the Web needs the

concepts of the universal set of objects, and of the universal set of names or addresses of objects." [RFC1630] This is a bit differently expressed in [Berners-Lee, WM] "The Web is a very general concept – one universal space of information. The concept it requires such as identifiers and resources are as general and abstract as possible."

4.3 Short summary of conclusions related to URIs and the Web

The Web is the infinite URI-namespace that the URI-specification [RFC2396] creates. Strictly speaking, a URI does not represent anything by itself; it is simply a point in the URI-space to a general viewer. The relation between a URI and an algorithm, formal or abstract, has to be inferred by the interpreter of the URI in order to relate the URI to an entity.

The use of the Web has started to change. The value of a universal namespace has been recognized, and therefore the future use of it will be extensive. In figure 4.7 below, it is shown that a web-resource can be used to represent entities such as humans on the Web (a web-resource is the filled ovals in URI-space together with the line illustrating the function to an entity).

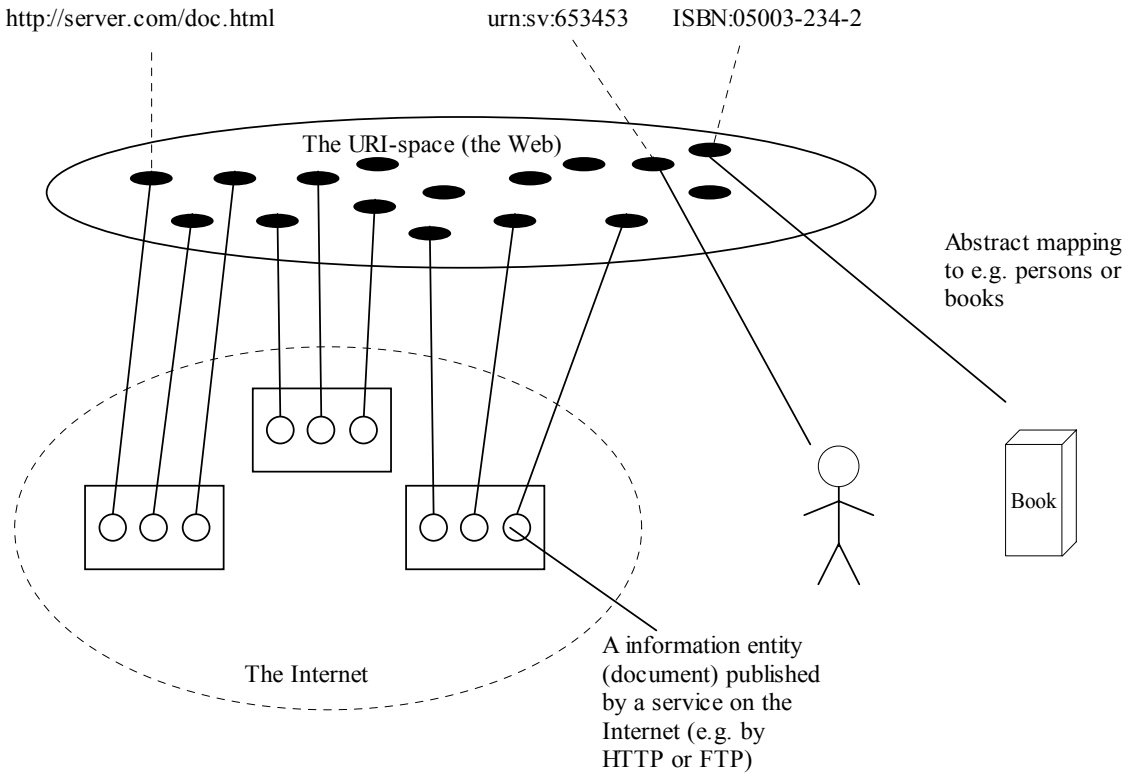


Figure 4.7

The most important thing about the Web is that it is a universal namespace, something that has not been available before, not at this level of precision.

5 The philosophy and theories behind the Semantic Web

The Semantic Web is a "web of meaning", as opposed to the "web of links" that the Web is today. This "web of meaning" will enable computers with specialized programs to help us not only to find information but also to derive information that did not exist before. What we need is to make information "meaningfully processable" by computers so they can use the information present on the Semantic Web as if they created it. The Semantic Web is thereby not a vision that tries to teach computers human languages; it is about either attaching computer processable information to human oriented information or to explicitly express information directly in a computer processable format. This implies that we do not need to rewrite everything on the Web, information owners that want to make their information more valuable just need to annotate it with machine processable information. The power of the Semantic Web will be most visible if data from databases intended for public consumption are expressed in the Semantic Web language. An improvement in digital trust is also included in the vision, both because it is desirable and because it is necessary as latter parts will show.

The Semantic Web is not directly aimed at human consumption, but instead directed at computers who will use this information to produce valuable human oriented information. Human users will most probably use Web browsers, similar to today's, accessing information created for humans, or they will interact with specialized and personalized software agents.

All this may sound a bit too "futuristic", but as one examines the foundation that is needed, it becomes a matter of time. To explain the Semantic Web language from a theoretical point of view, there are many things and philosophies that need to be understood or observed. The following sections will gently and informally introduce the Semantic Web concept and the problem it must overcome and important philosophies of Web oriented design that needs to be followed.

5.1 What is the Semantic Web?

There is, in my opinion, a slight misunderstanding when it comes to what the Semantic Web *is*. This is because people tend to explain the phrase by describing what can be done on it, or they get stuck in fruitless discussions on the syntax that is needed low down at the code-level instead of explaining what it actually is. Although it is important to discuss the use, and how it will be implemented, I don't think that it is the only way to explain it – in particular not this early in the development process. This confusion was, and is as a matter of fact, apparent with the "ordinary" Web. The definition of the Web I use is focused on the theoretical aspects, so "what the Web *is*" is that it *is* the URI-space as defined in chapter 4. The Web can then be defined as "an information space wherein people can communicate and interact", but that is in my opinion too abstract to be used for explaining what the Web actually is from a theoretical point of view. And the theoretical view is important if computer applications are going to be built on that as a foundation. Of course, when explaining to a layperson what the Web is using the theoretical definition would probably not help, but in the context of those who need to understand the foundations that is what is needed. Today the Semantic Web community is focusing extremely hard on the implementation of parts of the Semantic Web. Too little effort has been devoted to actually describing the underlying theories and the architecture of what they are building. This can lead to a number of problems due to the fact that the development loses its overview. I am not stating that the work done by implementers are wrong – it's very important too – but I think it is easy to lose the grip if one does not have an architectural description of some aspects of what one tries to build. I will in this thesis concentrate on what the Semantic Web actually *is* and

describe the *architecture* of the *language* from a theoretical point of view. I will explain the Semantic Web from what it *is* rather than what can be done with it. This might also be a bit dangerous: the reason for developing the Semantic Web is to make the Web as a concept more usable for humans. It is of no value to build the Semantic Web if it is of no value to its users, but there is no need to describe what one can do on it if it is not possible to construct.

As a sidetrack, Douglas Adams has stated that part of the Internet's extraordinary power was the fact that it "evolved as an organic entity, a bottom-up design rather that being hierarchically controlled from above". The language for the Semantic Web *needs* to be designed so that it can be used "out of control".

To visualize the Semantic Web inside the reader's head I will present a simplified definition and an image illustrating the Semantic Web. The simplest way to describe what the Semantic Web is to say: the Semantic Web consists of two implicit parts (1) a computer processable *metaweb* that describes the web-resources on the Web and contains directly computer oriented information, and (2) the Web itself. The reason for using "implicit" is that the Semantic Web is the Web, but can conceptually be considered to be at another abstract level and aimed at computers consumption (this is further explained in section 5.2). This is illustrated in figure 5.1 below.

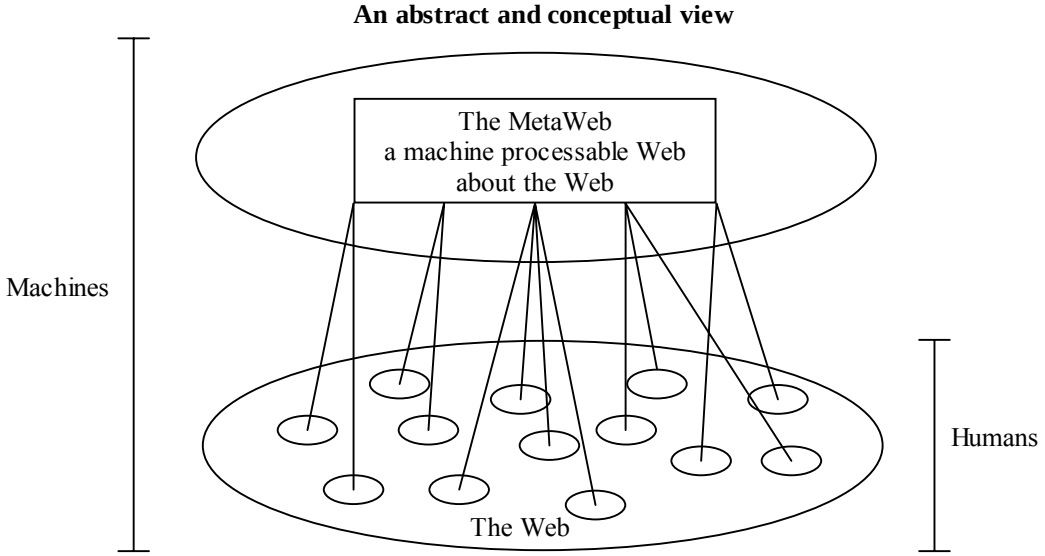


Figure 5.1

As shown is in figure 5.1, the Semantic Web (more precisely the *metaweb* in the picture) has one form of users: computers [S. Melnik & S. Decker, 2000]. The human users will not actually use the Semantic Web, they will profit from it because the computers use the Semantic Web to make it easier to use the information on the Web and to derive new information. The Semantic Web will basically do one thing: it will facilitate an infrastructure for expressing information that makes it possible for machines to process information (or even share knowledge).

5.2 The difference between the Web and the Semantic Web

The Semantic Web and the Web are basically two names for the same thing: a global information space wherein something navigates with URIs. The Semantic Web exists in the Web, and is a part of the Web at the same time. This makes them inseparable at the URI-level, and consequently, that level is not useful for explaining the Semantic Web or relating it to the Web. Analogously, you cannot compare two stories in a book by

comparing the individual letters – one needs to increase the level of abstraction of concepts to make that kind of comparison. If one increases the abstraction one sees that the Semantic Web and the Web are quite different, but also shares a few concepts.

Basically, the name Semantic Web comes from that the fact that it "represents" a set of semantically and formally interlinked data units – thereby creating a *semantic* web inside the Web [Berners-Lee, RM]. But this should also indicate that there exist important conceptual differences between them. Roughly, there are two conceptual differences between the Semantic Web and the Web: (1) The Semantic Web is an information space in which the information is expressed in a special machine-targeted language, whereas the Web is an information space that contains information targeted at human consumption expressed in a wide range of natural languages. (2) The Semantic Web is a web of formally and semantically interlinked data, whereas the Web is a set of informally interlinked information.

By examine these conceptual differences, one finds that there are similarities; and these similarities are the use of links and their importance. It is easy to separate links and human oriented textual information; a text document basically means the same thing to the author with or without links. But the use of links can highly increase the understanding and precision of the information – to a viewer. Instead of only writing: "At the conference I met Benny", the semantics and precision would be increased to a viewer if the word *Benny* were a link to that persons homepage, and *conference* was a link to the conference. The use of links is not only to make it possible to navigate in the space, but also to *share concepts*.

The separation of data expressed in the machine-targeted language and semantical links on the Semantic Web is harder to separate, and there is a reason for this: The machine-targeted language is defined (as will be explained in chapter 6) by making semantical links between different concepts. The semantics of the data expressed in the machine-targeted language thereby highly depends on how its parts, or the descriptions of its parts, are semantically linked, i.e. how one concept relates to another concept. This is very similar to how humans communicate (a bit simplified). Languages used by humans express meaning by referring to a set of shared concepts that grounds the understanding of their communication. This sharing and building of concepts has taken thousands of years to establish. Two applications developed separately of each other have no shared language but possibly shared concepts. By creating universal concepts identified by URIs, we can create the set of shared concepts that machines need if the machines are going to comprehend the machine-targeted language. If a language should become universal the concepts that are used has to be universal, and by using URI they are. Hence, the use of semantical links on the Semantic Web is to *share concepts*.

The biggest difference is that the Web is aimed at human consumption through the use of rendering software and the Semantic Web will ONLY be used by computers. The debate on whether the Semantic Web is the "new" Web is on the wrong level of abstraction. At the URI level they are the same thing, but at a higher level they differ substantially.

5.3 The difference in use

To make it easier to see how applications are going to be able to communicate on the Semantic Web it is often useful to see how humans communicate on the Web. Figure 5.2 shows a very simplified communication process between two individuals.

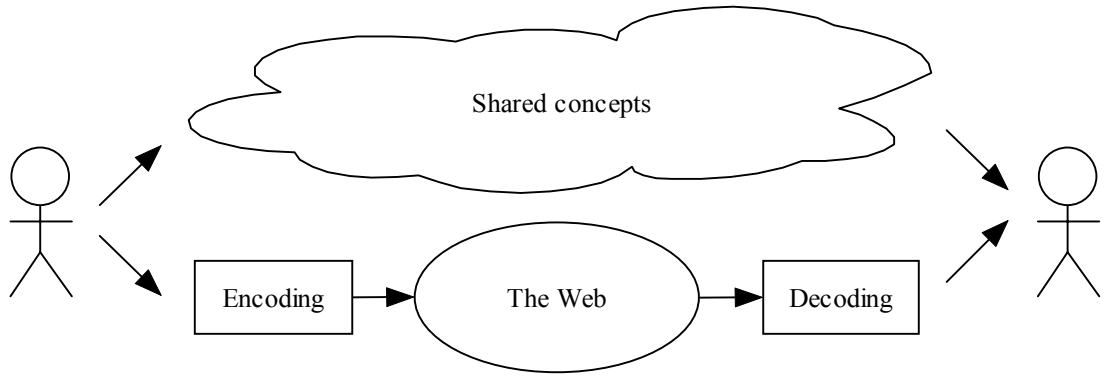


Figure 5.2

What the figure emphasizes is that, first of all, people need a set of shared concepts to be able to communicate. Different languages could then be used to encode the concepts that are communicated. The receiver of the message has to be able to decode the message and "rebuild" the meaning of the message by using the shared concepts. To understand the meaning, in some sense, of information one needs to handle both the language and the concepts it encodes. Another thing that is important is that the encoding informally declares which concepts that are used. A person that cannot handle a language cannot deduce which concept that the message that the language encoded uses. Reading a document is basically a heuristic process – guessing concepts.

Humans can, if they understand the language in which the information was encoded in and the used concepts, use the Web as a source of knowledge. The Semantic Web has the same role; it is a giant *knowledge base* expressed in a way that makes it possible for machines to use. Figure 5.3 shows how two applications running on computers could communicate on the Semantic Web.

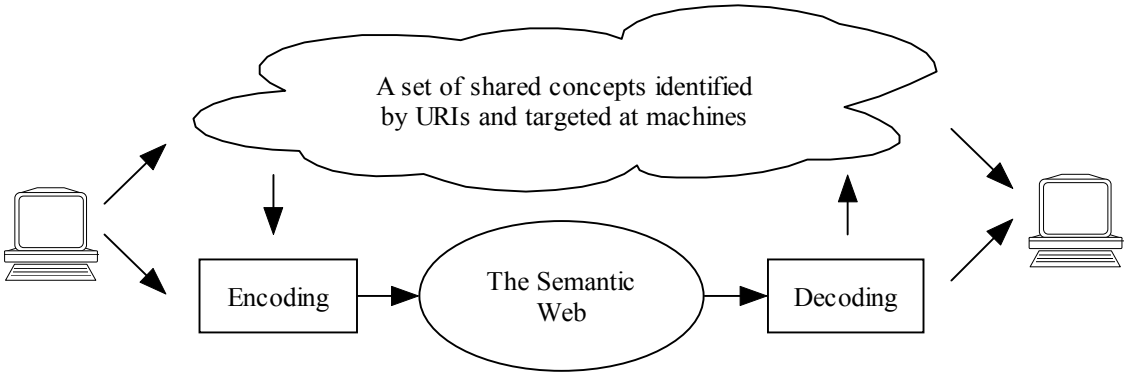


Figure 5.3

The similarities with human communication are obvious. The key thing to consider is the differences in the figures: Applications that communicate on the Semantic Web explicitly declare concepts that are used in the information encoding! The arrows from, e.g. the space of shared concepts to the encoding of the information illustrate this. Also, the space of shared concepts is the Web. This means that an application that receives an information encoding can extract, without guessing, which concepts are being used, and since the concepts are identified with URIs, they are universal. This does not mean that the receiving application automatically understands the concepts that are being used, but it knows *what* concepts are being used. Then, if the application is "smart" enough, it can try to relate the used concept to its hand-coded concepts so it can process the information. The Semantic Web is all about converting things [Berners-Lee, 2000].

As stated previously, the purpose of creating the Semantic Web is to help humans; we should not build the Semantic Web if it does not provide any use to humans. Humans will never use the Semantic Web – directly, Humans will use applications that will use the Semantic Web, or humans will use the Web to access information that computers accessing the Semantic Web have produced. Figure 5.4 shows how humans will use the Semantic Web, indirectly.

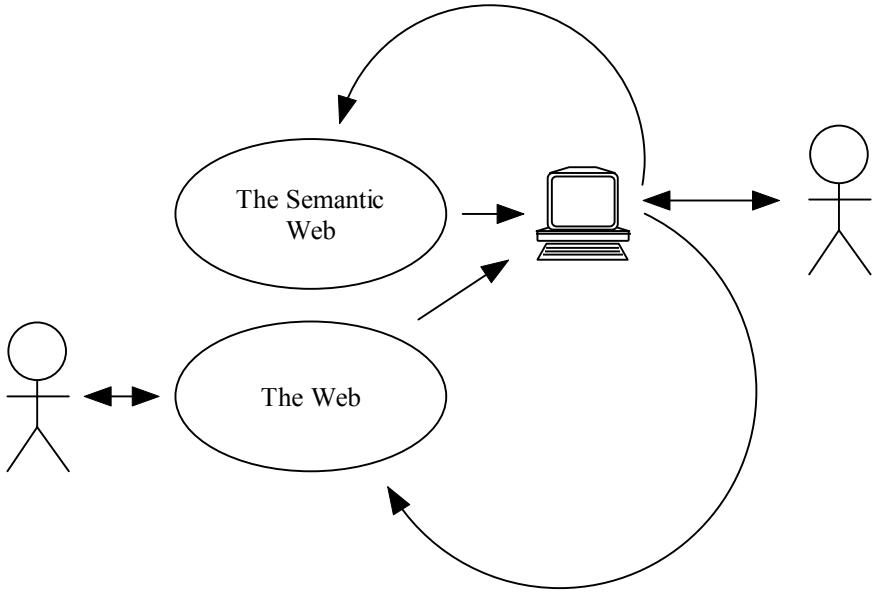


Figure 5.4

Using the Semantic Web directly is out of the question! Put it another way as in [Hendler, 2000] " … ideally most of the users shouldn't even need to know that the Semantic Web exists." We will continue to use Web browsers to instruct machines to help us, and to see the results. Also, machines will create information that is published on Web pages.

To make the Semantic Web happen, there are plenty of things that need to be considered, and problems to be solved. The easiest problem to solve is the precision problem that is present on the Web, and that is explained in the next section. But how do we create information in a way that makes it semantically processable by computers? This is the hardest problem, but this is not a new problem, the Artificial Intelligence community, especially the Knowledge Representation community, has been faced with this problem for a long time. To not "reinvent the wheel", their accomplishment has to be considered and reused as much as possible to find the suitable solution. Sections 5.5 – 5.7 discuss these issues.

5.4 How to achieve precision on the Web

One problem with the information on the Web toady is that there is no precision in the statements (that the information consists of). The information that is created is usually written to a certain audience, in a certain context, in a certain domain, and about a certain topic. All this makes the statements in the information precise, because the words that are used have a precise meaning regarding the special domain, context or audience. This information about the domain, context, and assumptions about audience is usually implicit, e.g. a book uses a book cover that encapsulates the information and label it with title, categorization etc. that archives this precision. Thus, information in a

book has a certain amount of precision, achieved by the books encapsulation. But as soon as the information from a e.g. book is directly published on the Web its precision gets lost since the books physical encapsulation is not possible to represent on the Web. And, the *hypertext* format, "discovered" by Ted Nelson, makes it possible to read information in a non sequential way: therefore one cannot assume that the person that reads a digitalized book starts at the beginning and reads it though to the end. A user that uses a search engine will most probably jump right into a chapter without knowing the domain, the topic, or the intended audience. Therefore, it becomes impossible for a search-engine to diversify the meaning of words, or statements because it does not know the context or domain it was originally written in. Since we cant use physical things like a book-cover to achieve precision we need something else. So, how can we use the Web so that precision in at least a certain aspect is preserved? The answer is quite obvious: use URIs to identify important things, and more generally use *metadata*. As Tim Berners-Lee puts it: "everything of importance should have a URI" [Berners-Lee, TLK]. This does not only include documents, or parts of it, retrievable over the Internet, but also abstract things, or things not digitally representable such as humans, feelings, and properties etc. but most of all *concepts*. This might at first seem a bit odd: "Ugh? Should people have URIs?" To see that it is not so strange, let us look at an example.

Imagine that you are doing "ego searching", i.e. looking for information about your self on the Web. How do you express that in a keyword search engine? First you might use your first and last name, but as the list of results excides tens of thousands of links to people that is not you, you might add your city and country. But that has the effect that pages where only your name is used will not appear. Now, it is quite obvious that if people had a URI, a URN specifically e.g. `people:sv:19740507-2342`, or simply a permanent mail address, the search would simple be to enter my URI instead if the information used it to make precise statements. Then all the results would be "good" answers. It does not mean that my name Benny should be replaced by this URI in the actual text, but there should be metadata that attaches the unique URI to the name. We don't change the way the text appears to human readers, we just attach (e.g. tag it) with URIs making the statement more precise.

```
… I read an article by <Benny="people:sv:19740507-2342"/> about
…
… I read an article by Benny about …
```

In the simplified example above, the first line is how the statement is actually represented at syntax level (how computers will read it), and the line below is how it will be presented to humans reading the document. The statement above has precision in one sense and that is who Benny is.

Since we can't use book covers to encapsulate information, we need to create an imaginary book cover. Attaching metadata to the information does this. A page that is snatched from a book should at least have some metadata about by whom, about what, the topic, etc. Meaning lies in the context and the context is generally also implicit in the information. The context that information was created in was partly the source to the imprecision due to the fact that the context was not represented along with the information. The solution of using URIs makes the thing that they identify context independent. A certain URI means the same thing regardless if you use it in a different country, or in a different domain, and this is the most important thing regarding URIs. To achieve a semantic structure is about putting information items on the Web in

context to one other. The task is about describing what a given information item "mean" in a computer processable way.

The meaning of a document on the Web can be defined more precisely than an ordinary paper document [Berners-Lee, ME]. This stems from the fact that the Web (the URI-space) is a global namespace, wherein the meaning of information items becomes world wide and non-ambiguous if the information declarers which language it was written in and which concepts are used [Berners-Lee, 2000].

What this is all about is to change the way information is expressed on the Web so that the information becomes as worldwide as the Web. Creating "artificial" contexts does this and the demand that always declaring in which (artifact) language it is written in. (Remember that this is not about the information that humans should read, but the information that computers should read.)

5.5 The Semantic Web is not natural language processing

The Semantic Web is not natural language processing: instead of asking computers to understand people's language, it involves asking people to make the extra effort [Berners-Lee, RE]. Teaching computers to process natural language is very difficult. At the beginning of the Semantic Web evolution, computers will not process natural language – but this does not mean that it would not be extremely useful though. The problem of processing natural languages basically comes down to identifying the structure of sentences (syntax), determining the surface meaning of sentences (semantics), and interpreting language in context [Dean et al., 1995]. This is difficult as it is, but in an environment that mixes domains, languages, context in a way that the Web does, it becomes even more difficult. An easier way, but not as general, is to use computer processable and context-less *metadata*. If we could attach computer processable metadata to information pools, a computer would then be able to use the information as a "value" of the type that the metadata declares to use the data in computations. This means that a computer could use the human oriented information in some sense without understanding it. To see this clearly and the use it has, let us look at an example.

When a function is declared in a programming language the parameters and the return type is usually specified, with the function body. This is basically done to make the functions processing "meaningful". It does not always make sense e.g. to make a division with strings if the function interprets the strings as integers. To make functions meaningful in some sense, values are of a certain type, and a function takes and returns certain values of a certain type. What course-grained metadata does is to make information items a member of a certain "type".

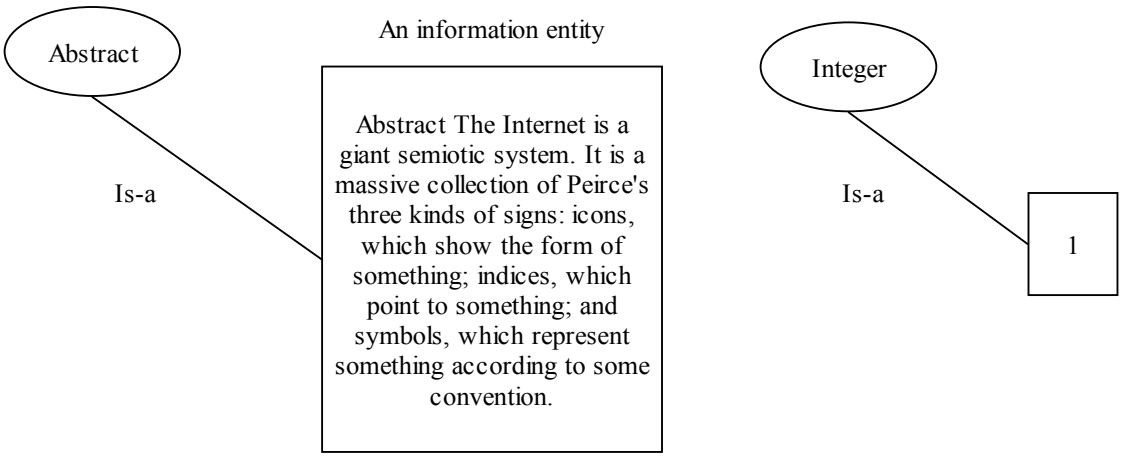


Figure 5.5

On the left in figure 5.5 is an information entity that is the extract of an abstract of a paper. Attaching metadata to this information pool is like making this information pool a member of a type. In the example, the information becomes a value of the type `Abstract` by declaring that the information `is-a Abstract`. In the analogy with programming languages on the right of the figure making "1" a member of the type `Integer` is shown. By explicitly making the information a value of the type `Abstract` makes it possible for a machine to use this value (the information pool), e.g. in a function that searches for specific keywords in abstracts. This makes the information computer processable indirectly. This is, a bit simplified, how the use of metadata could ease the use of the information without "understanding" the actual contents (this would require natural language processing). As with all programming languages, the computer executing does not know what the processing means, this interpretation is the role of humans.

Metadata does not have to be about big pools of information (course gained): metadata can attach properties to "atom" values too (fine grained). As an example consider the following information in a simplified XML syntax.

```
<name>Benny</name>
<age>80</age>
<favorite_quote>Perhaps, but let's not get bogged down in
semantics – Homer</favorite_quote>
```

This pool of information and attached metadata could be depicted as in figure 5.6.

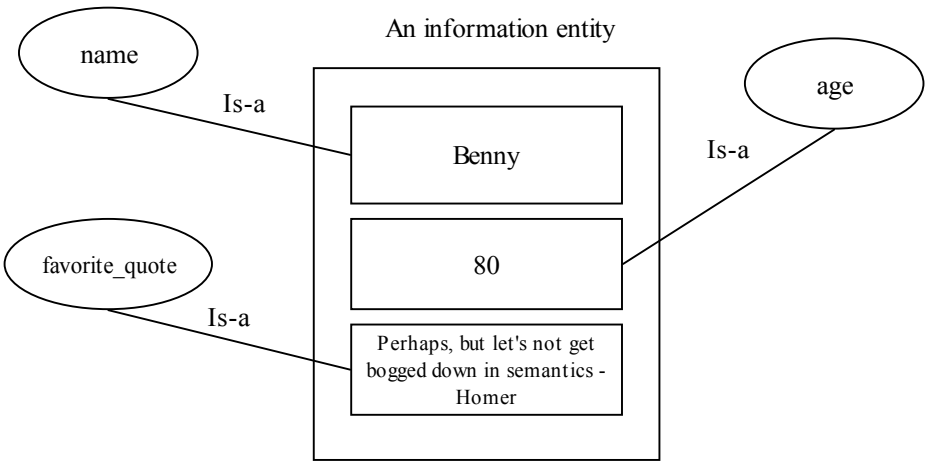


Figure 5.6

What the example should illustrate is that attaching metadata to information entities is just a binary relation. Metadata is a binary relation from a set of "types" to a set of information entities. This is a fact that will be used in latter parts of this thesis, but for now, consider metadata as a way of attaching machine processable information to information.

There is one thing that needs to be clear at this point. Adding the metadata *age* to the value '80' in the example has absolutely no meaning to a machine. The machine does not care if you, as a creator understand what age actually means. To a computer the isolated statement <age>80</age> is as understandable as #¤%¤# 2#¤"# #¤%¤# is to a human. The preachers of XML have always said that XML is "smart data" or "intelligent data" but the people who state this lacks the knowledge that XML is not intended for direct human consumption. Further, labeling meaningless data with meaningless data does not make the meaningless data less meaningless. This odd sentence comes from the fact that: "Metalanguage consists of signs that signify something about other signs, but what they signify depends on what the relationship those signs have to each other, to the entities they represent, and to the agents who use those signs to communicate with other agent." [Sowa, 2000] The way that the <age>80</age> could mean something to a computer is if it is hand-coded into the software that read that statement. The semantics of age is thus explicitly encoded into the software, making the semantics of the information meaningless to other applications. This is not a good thing for interoperability. To be able to find a way to make applications "understand" each other's information, one has to consider how a machine could understand. What we want is to make the meaning of a statement somewhat self-describing; we do not want it to be encoded into the application.

5.6 How a computer "understands" the meaning of signs

A computer by itself is stupid. It has no intelligence. A computer by itself is as useful as a stone. It is only useful as a tool, used by intelligent beings (creating "intelligent" programs). Throwing a stone through a window does not imply that the stone understands the concept of window or what it is doing, and likewise, a computer does not understand the meaning of the information that it processes only because it processes it. It does certain things because it is instructed to do so.

Building the Semantic Web is not all-new technology. There exists a huge amount of knowledge in different areas that is valuable for achieving the vision. The Artificial Intelligence community has tackled the problem of machine understanding in at least 50 years and their theories are very important and useful. Without going into a discussion

about what intelligence is, the take on intelligent programs that are used thought this thesis is *the rational agent approach* [Russell & Norvig, 2001]. This is a bit simplified built on that acting rational means acting so as to achieve one's goals, given one's beliefs.

To se how humans learn concepts is again useful as a comparison on what is needed to teach computers new concepts. In an encyclopedia a concept is explained by a structure of other concepts. The meaning of a concept is explained by how the concept is related to other concepts. This model of how semantics is "created" is the take that needs to be used on computers too.

What computer understanding comes down to is to be able to create a *computational theory* about the entities of interest [Dean et al., 1995]. Because the computers can't ground meaning in physical object or feelings, we have to create an abstraction of the world in a computable form. This abstraction is an approximation, and the theory consists of concepts that are explained with logic and relationships between other concepts. The relationships between concepts act like an encyclopedia to the computers. Such a theory needs, among other things, a formal system which makes explicit the entities and types of roles that plays role in the theory [Dean et al., 1995]. The formal system is often called a *representation* and consists of the parts explained below [Dean et al., 1995]:

1. **Syntax** – the notational conventions associated with a given representation
2. **Semantics** – the meaning of the objects and relations specified in the syntax (denotational conventions)
3. **Computation** – a model specifying how the objects and relations are manipulated in accordance with the semantical conventions

These three parts have some similarity to the parts that the study of signs – *semiotics* – consists of [Sowa, 2000] (quoted):

1. **Syntax –** Syntax is the study that relates signs to one another
2. **Semantics** – Semantics is the study that relates signs to things in the world and patterns of signs to corresponding patterns that occur among the things the signs refer to
3. **Pragmatics** – Pragmatics is the study that relates signs to the agents who uses them to refer to things in the world and to communicating their intentions about those things to other agents who may have similar or different intentions concerning the same or different thing

What this should make clear is that we cannot build a Web of computer processable data by only considering the syntax. For a computer to process information it needs concepts and relationships between concepts. A concept that is only a syntactic construction does not create the semantics that we look for. Thus, using e.g. only XML is not an option since XML only is about syntax and does not offer any semantics.

Without going into details, there are many things that need to be observed when we try to model the world in a computer processable way. One thing is that the information could be considered to be a set of signs, which represent something according to some convention. A sign has three aspects: it is (1) an *entity* that represents (2) another *entity* to (3) an *agent* [Sowa, 2000]. The machines have to "understand" the entities that the sign represents, not the sign itself.

Using the word *understand* in sentences like "the computer understand that this is a car" is in the strict sense wrong (and complicated since it leads into the meaning of

meaning). To generalize and easy the reading of this thesis, a simplified interpretation of understanding related to computers is needed.

A computer (program) knows the meaning of a sign if the sign is described in a computational theory and the producer of the sign uses the exact same theory.

Thus, a computer will never understand the entity that the sign might represent in reality, it will only understand the computable processing instructions that are described in the theory, and this is the essence of a computable theory. There is an important aspect that the sentence above states: The meaning of a set of signs is only possible to know if the computational theory that they belong to is explicitly stated. That is why the language used to describe the computer processable information needs to be declared. This is important, and will be further explained in latter chapters in association with *ontologies*.

The users of the Semantic Web (the applications) should basically provide *automatic reasoning*. Automatic reasoning is any computation that takes as input some representation encoding knowledge about the domain of interest and provides as output conclusions based on that encoded knowledge [Dean et al., 1995]. The Semantic Web does not demand that the computers should understand the actual entities that the sign represents, the computers should be able to process the attached properties according to a computable theory and thereby make use of the sign as data. And, something that needs to be in the back of ones head while continuing to read is: "Yet without the people, the documents and its contents have no meaning." [Sowa, 2000]

5.7 What distinguishes the Semantic Web from other KR approaches?

Knowledge Representation has promised a lot, but has often failed to deliver all the promises. A traditional KR approach in the AI field is based on two assumptions that make them inappropriate to use on a web: they are centralized and closed-worlds in that different systems cannot share worlds directly. Generally, a closed-world ontology consists of a hierarchy of concepts. Two systems developed independently cannot communicate if they don't use the same ontology. If a system demands that one giant hierarchical ontology tree is the only one used, such a system cannot be "webified". In the case of the Semantic Web, the Web itself is the ontology, and it is not a tree with a standardized structure, it's a web.

To reuse a KR approach it at least has to be tweaked to use URI as identifiers. Another thing that needs to be observed is that the Semantic Web is not an inference engine: it is a Web of machine processable data. There will not be a single Semantic Web machine [Berners-Lee, TLK]. There will exist many different kinds of inference engines, all optimized at solving specialized problem, so the Semantic Web does not demand a single Semantic Web machine. There will instead exist simple applications, like today's Web browsers, that use the Web as usual and there will also exist applications that are extremely powerful in terms of logical capabilities.

5.8 Different views of the Semantic Web architecture

As in software engineering, it is often possible to have multiple architectural views of the thing under consideration. Different views are produced to describe the architecture of the system to different stakeholders [Sommerville, 1995]. The different views often describe the system from different aspects to fit the stakeholder or to emphasize a specific aspect of the system. The Semantic Web could be viewed as a big system that consists of interacting subcomponents, the same way as the Web is just a huge system, and its architecture could be described by identifying the components and the

connections between them. That is not how I will describe the Semantic Web, because the focus is on the computer processable information, not the set of applications that provides and uses the information. The architectural description presented in latter chapter describes that architecture of the "Semantic Web language" that will be used to create computer processable information. The type of architectural description that is going to be used regarding the Semantic Web is the layered approach. In general, this approach is an architectural description consisting of a set of layers. These layers could represent a number of different concepts such as languages, information, computations, etc. Basically, these types of descriptions start at the bottom by describing some "atom" feature of the system, and then add layers that represent an abstraction that uses these functions to provide "higher-level" functions. In the context of the Semantic Web there are a few possible architectural views of this layered approach that one could use (some quite related):

1. **An architectural view of the chosen technologies.** This is a view that depicts the different technology pieces that together enable the Semantic Web. Since the Semantic Web is currently under development this view is impossible or very hesitant. This is mainly because there are so many things that are unsure. Currently, this view of the Semantic Web is well defined at lower layers but tends to get more abstract in higher layers, and thus does not describe the technology. This view will become more important as time goes by and when pieces start to fit together. This view is totally excluded in this thesis.
2. **A functional view of the architecture.** This type of view describes the different functions that the individual pieces offer to one other and the functions that the pieces depend on. This could for instance be depicted as a set of layers. A layer *n* uses functions provided by the layer at level *n*-1 and provides functions for the layer at level *n*+1.
3. **An information-model view of the architecture.** By this I mean that one considers the type and properties of the information that are present in the different architectural layers. This view does not consider the technology that support or enables the different states of the information. Basically this type of architectural view often starts at some low-level data model and then creates abstractions that form higher-level data models. Naturally, this view is important in systems that are informational centric. It often presides the first (1) view since it often is technology independent but might call for certain technologies.
4. **Language layers.** This view focuses on the language properties of the architecture. At the lowest level the language may only be possible to express simple things. But as the language develops, as it moves up the layers, more and more power is added to the languages, and thereby its ability to express more powerful things. It is from this perspective that the Semantic Web will be described in this thesis.

These views all could depict different but important aspects of the architecture of the Semantic Web. They are all needed in order to fully understand the Semantic Web. At this point in the development of the Semantic Web, it is important not only to try to describe the architecture by using the first view (above). This thesis will concentrate on the language view (4) because I consider it to be the most fundamental: The use of language reflects the type and properties on the information that it expresses, and the information governs how it can be functionally manipulated, and the language, information, and functions basically call for a certain technology.

5.9 Existing architectural descriptions of the Semantic Web

There exists, to my knowledge, two "architectural descriptions" of the Semantic Web and that is one by Tim Berners-Lee [Berners-Lee, TLK], and one by S. Melnik and S. Decker [Melnik & Decker, 2000] from Stanford University. Berners-Lee's description is a good description, but a bit specific on technology/language syntax in lower layers. His architecture [Berners-Lee, TLK] is depicted in figure 5.7:

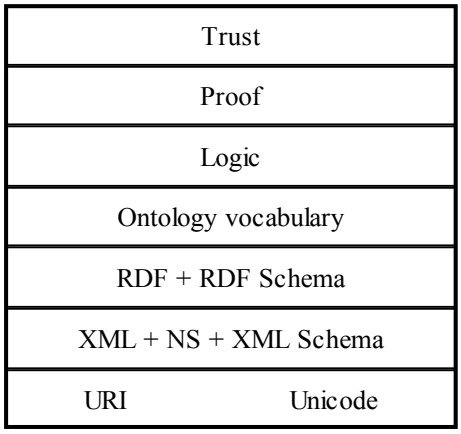


Figure 5.7

This architectural description in figure 5.7 is language oriented, but focuses at lower levels on the actual syntax that the language will have. Even if it is quite possible that these specific languages and syntaxes will implement the lower levels of languages, I will focus on the theories behind the languages that will eventually be used (generally, the theory behind a lot of prospect standards are the same, but they differ in syntax – especially in "syntax sugar"). Of course, the Semantic Web has one concept that its implementation technology must support and this is the URI concept - since it effectively creates the Web. This means that all the technologies that are eventually chosen must support this concept.

S. Melnik and S. Decker take a somewhat different approach. They focus on the data modeling layers. Their view from [Melnik & Decker, 2000] is depicted in figure 5.8.

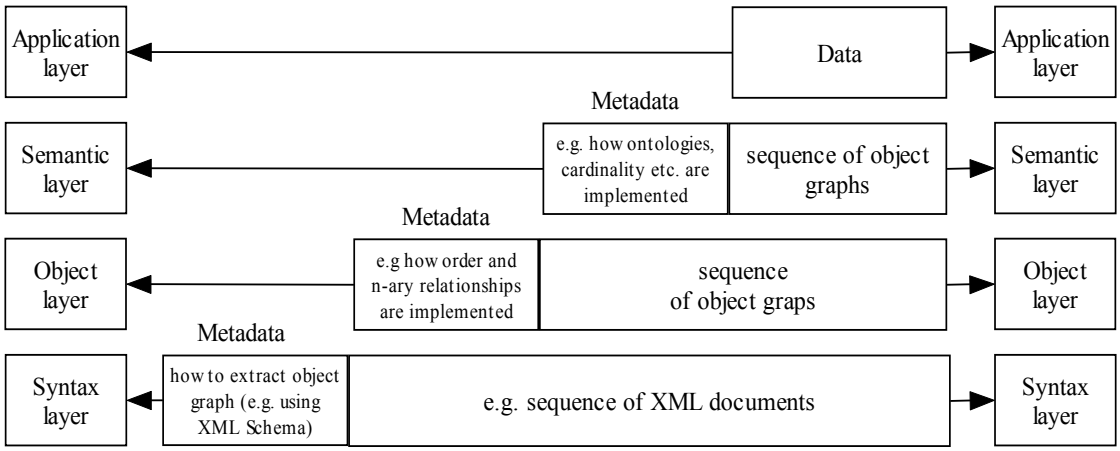


Figure 5.8

This view is more oriented at how the interoperability between applications consist of different layers, as a in the case of a network protocol stack. Applications working in the "Application layer" use the lower levels to enable semantical interoperability. This

view is quite useful when looking at how applications communicate thru the levels of "understanding".

The problem of uncertainty about what technologies, languages syntaxes, and standards that eventually will implement the Semantic Web was illustrated in [Studer et al., 2000] with a picture showing the increase of uncertainty as the "higher" levels of the Semantic Web architecture. Figure 5.9 is a reproduction of that image.

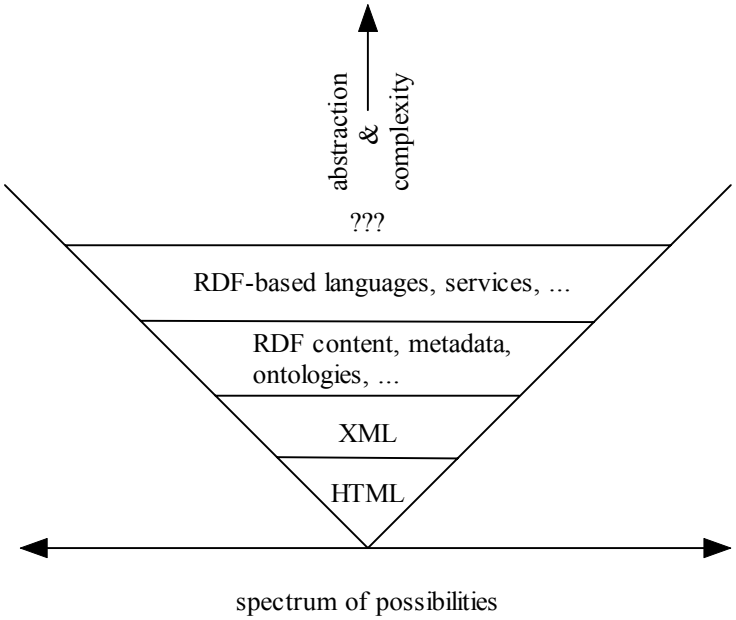


Figure 5.9

What this figure illustrates is that the number of possibilities is too great for a precise description of the Semantic Web architecture which is focusing on syntax, standards etc. This is why I concluded that a theoretical description on the core language would be more valuable than a "survey of prospect standards". The next chapter describes the layers of the language that will be used on the Semantic Web to enable computer processable data and metainformation about the information on the Web.

6 The language layers of the Semantic Web

The Semantic Web is a web of semantically interconnected data units targeted at machine consumption. To be able to express data and the semantics of the data in a way that enables computers to process it requires a special artificial and formal language. The language that the Semantic Web needs is in a way similar to role that HTML has to hypertext. The Semantic Web therefore needs a formal language with powerful capabilities that could express almost any type of information and explicitly declare the semantics of if it. Only when this language exists is it possible to connect and describe units of data forming the Semantic Web. In this chapter, the language described will be limited compared to the "full" language that will be used on the Semantic Web.

This chapter explains the theories and semantics behind the language that will be used on the Semantic Web. The language described here is directly aimed at machines and will be used to annotate human oriented information, e.g. text, pictures, sound etc, so that a machine could use the annotated information to process human oriented information in some sense. Also, and most importantly, the language will enable information to be expressed directly in it, thereby targeting the information directly at machines. Important aspects are that the language is declarative and that the language is self-referential i.e. it could be used to describe parts of itself and introduce new facilities.

The language presented here is built with a set of layers, where each layer uses the semantics and primitives that the lower layer provides to create a more powerful language. The actual syntax will never be considered, only artifacts that will help the discussion. Further, to be able to talk about the language one needs to use a terminology, and the chosen terminology in this chapter is influenced by the Resource Description Framework [Lassila & Swick 1999] (RDF) [Brickley & Guha, 2000] (RDFS). The reason for using this terminology and to a certain extent the same semantics is that the Semantic Web community is using this terminology to communicate, and it would therefore be cumbersome to introduce a completely new vocabulary.

Regarding the semantics of the language, influences have been taken from a diverse field of languages but the theories presented here are more rigid and unifies the different languages. Influences regarding the semantics have been taken from: Resource Description Framework [Lassila & Swick 1999] (RDF) [Brickley & Guha, 2000] (RDFS), Notation 3 (N3) [Berners-Lee, N3], SHOE [Heflin et al., 2000], Ontology Inference Layer (OIL) [Horrocks et al., 2000], and Metalog [Marchiori & Saarela, 1999], Conceptual Graphs (CGs) [Sowa, 2001], and DARPA Agent Markup Language (DAML) [Hendler, 2000]. The semantics of all these languages is extremely similar and is not newly invented but has strong influences from many additional fields such as Artificial Intelligence. As explained previously, the languages developed in the AI community have not been able to scale up to the Web, but are nonetheless useful to consider as input to design decisions.

The artificial language described in this thesis unifies the semantics of all these languages by looking beyond the syntax. Thus, this section actually specifies the semantics of a class of languages. Ignoring the syntax revels that the semantics of the language to be used on the Semantic Web is not that complicated. To realize the Semantic Web the language of course needs syntax, just as natural languages.

As mentioned in section 2.4, the semantics of this language will be described by using first-order predicate calculus, since it provides a solid and commonly understood semantics, and a set theory model. Further, the use of *web-resource* is here as defined in chapter 4.

6.1 Level 1 – Statements and triples

The first level of the language has to be general, un-constraining, and expressive enough for handling any prospect language functions [Berners-Lee, RM]. Further, the language needs an extensibility mechanism so that new features can be introduced, i.e. by higher layers or in the future. As always, a general and un-constraining language does not provide any specific features for direct use, its use is to enable the ability to express any prospect higher-level language.

To see how the fist level should be constructed it is important to see what the language is to be used for, i.e. what it is that the language should be able to express. In the most fundamental aspect there are two uses of this language:

- *Metadata* – The language should make it possible to annotate human oriented information so that a machine could find and use this information to provide better services to humans.
- *Knowledge representation* – The language should make it possible to represent knowledge (as in Knowledge Representation) on the Web targeted at machines.

What these uses of the language have in common is the need to make *statements* (a.k.a. assertions). Metadata in its general use is basically statements about a web-resource representing human oriented information, and knowledge representation is very simplified a set of statements that describes some knowledge in the domain of interest. Thus, at the most basic level the language needs to be able to express statements, i.e. to model assertions. In addition to that, since we are talking about a universal language that is to be used in the Web-namespace we have to use URIs in statements, as explained in previous chapter. **All elements of a statement must be URIs with an optional fragment ID!** (But to ease the reading of this thesis, examples uses often plain names but should be considered to denote a URI.)

6.1.1 Informal features and syntax constraints

This section explains the language in an informal way to easy the understanding. It explains the features, needed primitives, some syntax constraint, and informally the semantics of the language at this level. The section should serve as an introduction to the formalization of the language semantic.

To see what constitutes a statement, it is useful to look at logical languages. As an example lets use the statement "The world is flat". (At this point we don't care what the statement actually means, or if it is true in some sense or context, we only want the ability to express statements.) In logic, a statement is basically a proposition. In propositional logic a statement could be as simple as that proposition `the-world-is-flat`. To model the statement "The world is flat" in predicate calculus (*PC*), the concept of *predicate*s could be used. A predicate is used to describe properties and relations involving arbitrary objects [Dean et al., 1995]. In *PC*, `is(world,flat)` is a statement (atomic formula) where `is` is a binary predicate. It is important to remember that a statement `is(world,flat)` does not mean anything to a computer. The semantics that is implicit is that the previous statement has three elements, i.e. the computer needs to understand the ordering of the elements.

What this should indicate is that a statement, in the context of the Semantic Web, basically is a triple. In the Semantic Web community, the elements of these triples are named *subject*, *predicate* and *object*. To this end, statements will be expressed in the artificial language to be used on the Semantic Web as `(s,p,o)`, (in some syntax) which is short for *subject*, *predicate*, and *object*. (It is also possible to denote this with *object*,

attribute, and *value*, such as `(o,a,v)` but this is less common in the Semantic Web community.) It is important to notice that the use of subject, predicate, and object should not be confused with the use in linguistic contexts. Using triples to express statements is nothing new, it have been used a long time, in AI and especially in natural languages. The terminology subject, predicate, and object are used in both RDF and N3, but the concept of triples is a common ground that many other languages use.

To be able to express these statements the language needs the concept of identifiers. The obvious choice of identifiers is as explained URIs. All elements of the statements will be URIs, most often, identifying a web-resource representing some entity on the Web. Below (6.1) is the general definition of statements, the set of all possible statements.

$$setOfallStatements = \{(s, p, o) \mid s, p, o \in UF\} \tag{6.1}$$

Where $UF = \{x \mid x \in S \wedge URIF(x)\}$ is all valid URIs with an optional fragment ID. The predicate *URIF(x)* describes the URI to be a valid URI plus an optional fragment ID, i.e. it identifies a web-resource. At this point there are no restrictions on the URIs that the *s*, *p*, and *o* take in a statement.

In the following, the 3-ary predicate *statement*(*s*,*p*,*o*) is used to denote that the statement (*s*,*p*,*o*) in the language is present.

A web-resource identified by a URI has a certain role in a statement depending on where in the statement its URI occurs. A web-resource that is identified by the URI used as; subject is the web-resource that the statement is about; predicate has the role of a *property*; object is the "value" of the statement. A bit more formally: There exists a set *R* of web-resources that are all the web-resources and a subset *P* of this set denotes all the web-resources that are *properties*. RDF takes this approach. To diversify theses sets there is a need to introduce two primitives: `type` and `property`. `type` is a primitive property, i.e. it is used in the predicate position and `property` is a primitive object (or value). Making a statement that a web-resource is of `type property` makes that web-resource member of the subset of web-resources that are the set of properties. All web-resources are implicitly stated as being of `type resource`, i.e. they are all considered to be elements of the set of web-resources.

The set of all web-resources that are properties are defined below in (6.2):

$$P = \{p \mid p \in R \wedge statement(p, TYPE, PROPERTY)\} \tag{6.2}$$

The rule above states that all web-resources that is described as having the property `type` valued with `property` constitutes the set *P*. The predicate *statement* means that the statement is true.

As said before, logically, a statement is viewed as an atomic formula involving a binary predicate and two constants. A statement `(s,p,o)` maps to an atomic sentence `p(s,o)`. Thus, e.g. `{6,is_bigger,5}` is logically the atomic sentence with the predicate `is_bigger` and with terms `6` and `5`. At this point the predicate does not mean anything to a computer, a statement should be interpreted as a fact.

To informally explain/introduce the semantics of a statement it should then be compared with the semantics of an atomic sentence involving a predicate. A statement about a web-resource is stated because is should express a statement about the entity that the web-resource represent. Thus, an interpretation has to have a set of entities that we are interested in (a domain) and must map each term (a URI identifying a web-resource) to an element in the domain (a entity). Since a predicate is used to denote

relations or properties, a predicate is a relation and therefore the interpretation must map a predicate onto a relation on the entities of the domain. This is the "core semantics" of the language at this point.

The language now consists of three types of objects: web-resources, properties (a property is also a web-resource), and statements. This is also the case with RDF and N3 at the basic level. This means that the language is highly unconstrained; the only constraint is that all statements are ordered 3-tuples. The only thing that we can do with the language is to make statements, and query on the predicate, subject, or object in the statement. So, in a way, a statement is meaningless but not useless at this level. This general language indicates that to make the language more powerful we need to layer things on top of it.

At this point it is important to stop and think of what the statement is actually about. As an example statement consider something along the line of: (`http://homepage.com/benny/overivew.html`, `http://purl.org/DC/creator`, `http://people.org/benny`). This statement was stated by someone to assert that the `creator` of `http://homepage.com/benny/overview.html` is Benny, where creator and Benny have unique URIs. But what does this mean? At this stage in the language, this is only a fact, the individual elements of the statements does not mean anything. The statement is a simply an ordered set of URIs. Another thing to observe is the fact that the statement consists of URIs. This means that a statement is basically a set of points in the URI-space that are ordered into a 3-tuple. Thus, a statement is a fact about points in the URI-space. What we actually want to use the statements for is to make statements about web-resources. Although this is important to understand, in the following it is assumed that a statement is about the web-resources that the URIs of the statement identifies.

The language also has to be able to express statements about other statements (called *higher-order statements* in RDF). Statements about statements are something that occurs often in real life: "Jim said that the world is flat, but that is false". Statements like these have to be modeled, so there is a need to introduce a set of primitives, since it is not possible to write a statement about a statement with two statements if all elements of a statement should be treated as a first-class object. `((s2,p2,o2)p1,o1)` does not work directly because `(s2,p2,o2)` is not a first-class object (a web-resource) with a URI. The solution is *reification*, used in RDF, N3, and CGs, but is a common feature in KR. To allow statements about statements a statement is *reified*. This means that the statement is represented by another object that has certain properties that represent the reified statement. Reification turns what is an explicit statement into a description of a statement that is not specifically asserted, but it is used to make statement about [Berners-Lee, ST]. Thus, a statement and a reified statement exist independently of each other.

For this to be semantically processable by a computer there are a set of primitives that needs to be "understood" by a computer handling the language at this level. The concept of *type* has been introduced previously, but it is also needed here because the object that is representing another object has to be semantically diversified. Generally, a type is like a *unary predicate* in logic: the `(jim,type,man)` that states that "jim is of the type man" is usually represented as the unary predicate `man(jim)`. But this could equally well be expressed by a binary predicate explicitly declaring that the predicate is "is of type". To this end, there is a need for three primitive predicates (relations): `subject`, `predicate`, and `object`. And to express that an object is representing a statement, there is a need to express that it is of type `statement` [Lassile & Swick, 1999]. These are all needed to represent the statement that is reified. If a web-resource

representing a reified statement is found, the components of the reified statement have to be identifiable.

Before explaining the semantics of all the introduced primitives, table 6.1 below is a summary of language primitives that are introduced and an associated informal explanation of their use. *Predicates* denote a primitive web-resource that is used as in the predicate position in statements (i.e. property), and *Objects* are primitive resources that are used in the object position in statements.

Predicates:	Objects (primitive resources):
`type` Making a statement where the predicate is `type` makes the web-resource in the subject a "member" of the type that the web-resource in the object position represents. This allows e.g. the type identification that is needed in order to identify a web-resource that reifies a statement. This is also needed to be able to state that a web-resource is a `property`.	`statement` A web-resource that is representing a reified statement needs to be identified. A web-resource that has the property `type` valued `statement` is representing a reified statement.
`predicate` Used when reifying a statement. A statement where the predicate is `predicate` states that the object is the reified statement's predicate.	`property` Properties are projections to the second argument of the logical predicate statement. A web-resource that is of `type` `property` makes that web-resource member of the subset of web-resources that are the set of properties.
`object` Used when reifying a statement. A statement where the predicate is `object` states that the object is the reified statements object.	
`subject` Used when reifying a statement. A statement where the predicate is `subject` states that the object is the reified statements subject.	

Table 6.1.

The meaning of all these primitives is implicit, i.e. a machine working with the language at this level has to be hand coded to handle these types of primitives.

Prior to explaining the languages features, there is a need to state a set of logical rules explaining the introduced predicates of the language. In the rules defined below, *predicates* are written with *italics* and with lower-case letters, *PRIMITIVES* are written in *italic* and in capital letters. In this context, i.e. logic description of the languages semantics, a statement is regarded as an atomic formula involving a binary predicate and two constants/variables, and *statement*(*s*,*p*,*o*) is true if (*s*,*p*,*o*) is a valid statement in the language as define in previous definitions and (6.4) below.

The unary predicate *uri* below means that *uri*(*x*) is true if *x* is a URI a valid URI plus an optional fragment ID.

$$\forall s, p, o : statement(s, p, o) \Rightarrow uri(s) \wedge uri(p) \wedge uri(o) \qquad (6.4)$$

Rule 6.4 simply says that if an ordered triple is a statement, then every element of the triple is a URI.

The binary predicate *type* means that *type*(*x*,*y*) is true if *x* is of type *y*.

$$\forall s, p, o : statement(s, p, o) \Rightarrow type(p, PROPERTY) \tag{6.5}$$

Rule 6.5 states that if a statement is given then the web-resource in the predicate role, are of type property.

To create a reified statement of a statement (*s*,*p*,*o*) four statements are required as rule 6.6 [Conen & Klapsing, 2000] describes. Not that the statement that the reified statement models need not to be present (since a statement and a reified statement exists independently – in this language).

$$\begin{aligned} &\forall s', s, p, o : statement(s', TYPE, STATEMENT) \wedge statement(s', SUBJECT, s) \wedge \\ &statement(s', PREDICATE, p) \wedge statement(s', OBJECT, o) \Rightarrow reifies(s', s, p, o) \end{aligned} \tag{6.6}$$

The predicate *reifies*(*s′*,*s*,*p*,*o*) means that the web-resource *s′* represent the reified statement (*s*,*p*,*o*).

$$\forall s', s, p, o : reifies(s', s, p, o) \Rightarrow reifyingStatement(s') \tag{6.7}$$

Rule 6.7 is an abstraction used in 6.8

$$\begin{aligned} &\forall s' \exists s, p, o : reifyingStatement(s') \Rightarrow \\ &statement(s', TYPE, STATEMENT) \wedge statement(s', SUBJECT, s) \wedge \\ &statement(s', PREDICATE, p) \wedge statement(s', OBJECT, o) \end{aligned} \tag{6.8}$$

6.8 above states that if we find a web-resource representing reifying statement, the only thing that is valid to conclude is that there exists four statements about this resource, modeling the modeled statement. It does not mean that there necessarily exists a statement as the one modeled. This comes from the fact that reification is also used as an extensibility mechanism and that there should not be call for global consistency. Therefore, e.g. [Coen & Klapsing, 2000] defines another predicate *reifies_fact* as below, but this is not used in this thesis.

$$\forall s', s, p, o : reifies(s', s, p, o) \wedge statement(s, p, o) \Rightarrow reifies\_fact(s', s, p, o) \tag{6.9}$$

To ease the understanding of the language it is often useful to use some form of graphical representation for the language. RDF and N3 uses directed labeled graphs (DLG), and Concept Graphs (CGs) also uses a form of graph. Since the language at this level is practically equivalent to RDF and CGs in the following examples the use of directed labeled graphs will be used. The concept of statements maps directly onto a DLG – as described in figure 6.1.

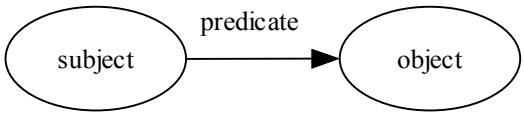


Figure 6.1

(Making statements about predicates that is used in other statements does not directly map onto DLGs since an arc has to be drawn from another arc to a node.)

Example 1. “Tim knows Sara”

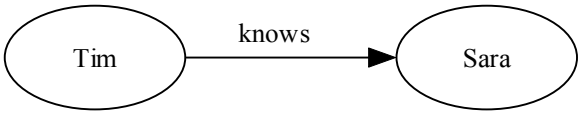


Figure 6.2

Example 2. “Tim knows Sara, Stefan, and Jim, and Sara also knows Jim”

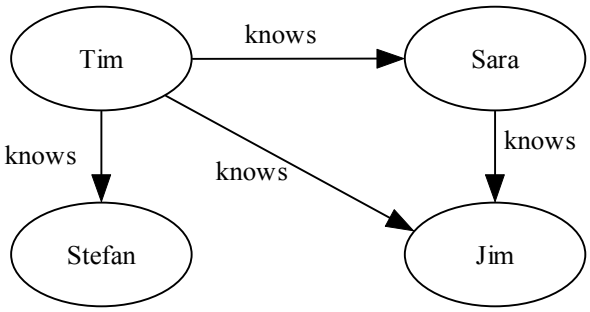


Figure 6.3

Example 3. “Someone states that Tim knows Sara”

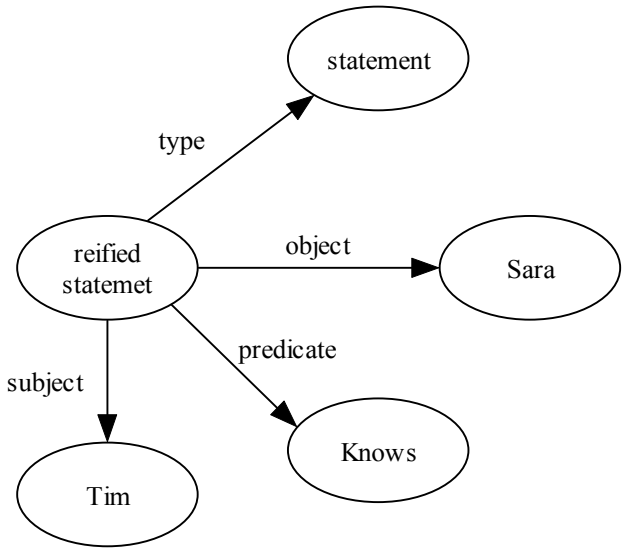


Figure 6.4.

At this point the language has only the concepts of statements and reification (not negation or implication). This means that the language is quite limited (e.g. not Turing complete). It is easy to find a proof, i.e. a statement, if there exists one. It is only possible to search on statements or on the components of a statement. But since the model is so general and the existence of a reification mechanism, things can be modeled on top and thereby create more powerful languages [Berners-Lee T, RM]. Generally, to

extend the languages power, new predicates or property values is introduced in the next languages layers.

To see the, at this point, the limited capabilities of the language let us assume that there is a search engine that traverses the Web and collects statements. The statements could e.g. be translated into a knowledge base in Prolog and then be queried. (The last statement is in the form that will actually be found on the Web, i.e. all elements are URIs.)

```
S1: (Benny, knows, Martin)      "Benny knows Martin"
S2: (Ronny, father_of, Benny)   "Ronny is the father of Benny"
S3: (Benny, lives_in, Sweden)   "Benny lives in Sweden"
S4: (Ronny, lives_in, Sweden)   "Ronny lives in Sweden"
S5: (http://someserver.com/sw.html, http://purl.org/DC/Creator,
mailto:semanticweb@operamail.com)
```

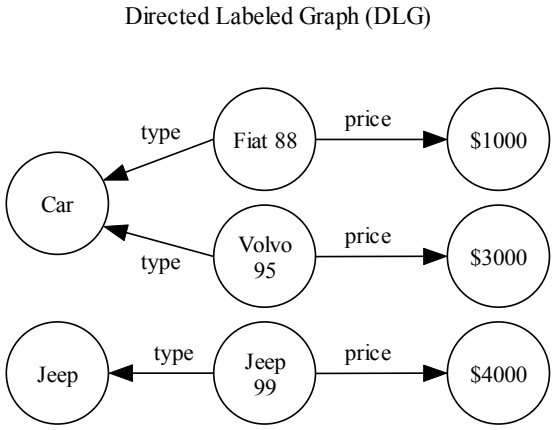
The questions we could ask, if the statements are represented in e.g. a prolog knowledge base, are:
"Does anybody know martin?" (*, knows, Martin)
"Who does Benny know?" (Benny, knows, *)
"Is there any statement about Benny" (Benny, *, *) or (*, *, Benny)
"Who is the creator (http://purl.org/DC/Creator) of http://someserver.com/sw.html?

These are all quite useful, but there are some obvious limitations. A computer does not know what `knows` mean, it simply is treated as a value. Is the `knows` relation symmetric? Does Martin know Benny (i.e. is (knows, Martin, Benny) true?)? All the semantics of the used predicates has to be known by the user of the search engine or explicitly encoded into the search engine if such easy deductions should be possible, and this does not provide much of interoperability. At this point there is not much help we can get from the computers, other that searching. We want machines to make deductions, but the language at this level does not provide the needed power.

It is important to consider how sets of statements should be treated. All statements that occur in the same context is considered to be implicitly and conjunctively concatenated [Conen & Klapsing, 2000]. A context is in this case e.g. a web page or a part of a web page. On the Semantic Web there will be a huge amount of statements, some of them will always be true, some always false, but there will exist statements that are true in certain *contexts*. The context of a statement is indispensable to its use [Berners-Lee, ST]. If an agent is sweeping the Web for information to archive a goal and finds two statements that are in conflict of each other, how shall the agent interpret this? The solution is to reify every statement, i.e. creating logical *contexts* [Guha, 1995]. A context makes it possible to consider a statement as true in a certain context. An agent "picking" up statements from the Web has to treat them as true in a certain contexts. This could be done by registering the URL from where the statements where extracted, i.e. creating a 4-tuple internally, or/and use the possible present digital signature to decide context. This means that the agent could possibly come to conflicting conclusions depending on which statements from which context it is using. Choosing what statement or context to believe is a matter of trust and should be utilized by software. It should be possible to state that "I trust all information from www.w3c.org, but I don't trust anything from www.microsop.com". It could also be a user activity: the agent could prompt the user asking: "I've come up to the conclusion that the U2 has five members from www.u2fan.com, but according to www.tripod.com/users/nisse U2 has four members. Who do you trust?" This could also be automatically done – if you trust your agent.

6.1.2 Triples

A data model is a syntax-neutral way of representing language statements. RDF uses this concept to evaluate equivalence in meaning: two language statements are equivalent if and only if their data model representations are the same [Lassila & Swick, 1995]. This section describes how the actual statements will be represented in the context of the Semantic Web. The language described so far is capable of making statements. The use of the statements is to be able to model metadata and express information directly in the language. For the language to be useful it has to be possible to map almost any data model onto the language data model. There is lots of information present in e.g. relational database so therefore, the Semantic Web needs a model of great generality when it comes to what data models can be mapped onto it. As implied by the language at this level, the data model is *triples*. A statement maps, of course, directly onto triples, but to see how triples are general figure 6.5 shows how directed graphs and relational data maps onto triples.



Relational data

	Type	Price
Fiat 88	Car	$1000
Volvo 95	Car	$3000
Jeep 99	Jeep	$40000

Set of triples

(type,Fiat 88,Car)
(type,Volvo 95, Car)
(type,Jeep 99, Jeep)
(price,Fiat 88, $1000)
(price,Volvo 95, $3000)
(price,Jeep 99, $4000)

Figure 6.5

All these three data models (DLG, relational data, and triples) are basically equivalent.

In [Berners-Lee T, RM] Berners-Lee states that in order to be able to represent any prospect data format, a general data model is needed on the Semantic Web. He later proclaims that the RDF provides such a global model. As explained previously, I do not intend to use any specific standard, syntax, or technique, so I will only use the underlying data model. The RDF data model is simply triples. The RDF triple is one of several *formal models* offered by the core RDF spec, it and consists of an ordered 3-tuple of URIs.

$$statement(s, p, o) \Rightarrow triple(s, p, o) \tag{6.8}$$

Thus, all information that will be "understandable" to machines will be in the form of triples encoding statements in the “statements language”. This represents the lowest level of the information, i.e. the smallest entities of information that are used to build higher information levels. Of course, in order to represent triples in an unambiguous and universal way we need technologies (serialization syntaxes and utilities) that support this.

6.1.3 Semantics

This section develops a, sometimes simplified, model for the language described in this thesis. The framework for the model has influences from [Peacock, 1988] who developed an abstract model (set theory model) for RDF WD 16 Feb 1998. The model developed here is different in a number of places from Peacocks, partly because this model is not dedicated to RDF, but especially the interpretation of properties.

To relate the model to the language described so far let us first consider the vocabulary for the language.

$$V = URIF \cup \{R_0, R_1, R_2, R_3, R_4, c_0\} \tag{6.9}$$

The vocabulary for the language consists of the set *URIF* of all possible URIs with possibly a fragment id appended. R_0 is a relation symbol (denoting a 3-ary relation), R_1 – R_4 are binary relation symbols, and c_0 is a constant symbol. In the previous sections, various names were used to talk about the different language primitives. To simplify the understanding of this section, the relation between the vocabulary and the terms used in the previous sections could be: R_0 is `triple`, R_1 is `type`, R_2 is `subject`, R_3 is `predicate`, R_4 is `object`, and c_0 is `statement`. Note that the actual syntax is not important. Different syntaxes could be used; the model and semantic is syntax independent.

The model *M* for *L* is:

$$M = \langle WEBRES, \alpha_{obj}, \alpha_{rel} \rangle \tag{6.10}$$

WEBRES is a set of objects that is the universe of the model. An element of this set could correspond to a URI in the language syntax under a certain interpretation, but the objects could otherwise be considered to be general. α_{obj} is the *object interpretation* and α_{rel} is the *relation interpretation*. This distinction of these interpretations is necessary because a property could be the subject of a statement and the statement could be about either the object that represents the property or the relation. E.g. there is an interpretation issue in statements like "The creator of property *P* is Benny" and "Property *P* is transitive". The first statement is under one interpretation about the object that represents the property and under another about the relation/set. The second example is a statement about the set. Thus, statements that have properties as subject or object could sometimes be interpreted in two ways and this distinction has to be made at the syntax level.

6.11 states the object interpretations on the language and 6.12 the relation interpretations (lowercase words are constants in the universe and capitalized words are sets/relations on the universe).

$$\begin{aligned}
&\alpha_{obj}(R_0) = triple \\
&\alpha_{obj}(R_1) = type \\
&\alpha_{obj}(R_2) = subject \\
&\alpha_{obj}(R_3) = predicate \\
&\alpha_{obj}(R_4) = object \\
&\alpha_{obj}(c_0) = statement
\end{aligned} \tag{6.11}$$

$$\begin{aligned}
\alpha_{rel}(R_0) &= STATEMENT \\
\alpha_{rel}(R_1) &= PROP_{type} \\
\alpha_{rel}(R_2) &= PROP_{subject} \\
\alpha_{rel}(R_3) &= PROP_{predicate} \\
\alpha_{rel}(R_4) &= PROP_{object}
\end{aligned} \tag{6.12}$$

All object interpretations in 6.11 map onto elements in the universe *WEBRES*, i.e. $type, subject, predicate, object, statement \in WEBRES$.

To be able to express the semantics of the other primitives a set of axioms and relations has to be created.

The model handles objects, statements about objects, statement about statements, etc. The set *C*, that is to be defined below, is a set that contains both objects and statements. *C* is divided into subspaces, identified by a subscript, e.g. C_0. The different subspaces will be referred to as levels, e.g. C_0 is level 0. At level 0 *C* will only contain elements of *WEBRES*. Then, if a set of statements is made about the elements in level 0, these statements will be considered to be objects at level 1. Further, statements about these statements (objects) will be considered to be objects at level 2 and so on. 6.13 and 6.14 below defines this. Note that *T* is defined in 6.17-6.19. Note that if nothing else is specified, the variables *i, j,* and *k* are considered to be natural numbers.

$$C_0 = WEBRES \tag{6.13}$$

$$C_i = \cup_{\forall \sigma \in C_{i-1}} [T_{\sigma,1}(C_{i-1})] \tag{6.14}$$

C_i defines all the objects at level *i*, i.e. objects that are statements about objects at level *i*-1 if $i > 0$. 6.15 then define the set of all objects (an object at higher levels represents a statement).

$$\overline{C} = \cup_{\forall i} C_i \tag{6.15}$$

As with *C* the set of primitive properties of *P* is defined to be the set P_0. P_0 contains the objects that 6.16 define.

$$P_{rel_0} = \{PROP_{type}, PROP_{subject}, PROP_{predicate} PROP_{object}, PROP_{subject}\} \tag{6.16}$$

$$P_{obj_0} = \{type, subject, predicate, object, subeject\} \tag{6.17}$$

$P_{obj_0} \subseteq C_0$

Now it is time to define the operator *T* that is used in 6.14.

$$T(C_k) = \{(t_1,t_2,t_3) | t_1 \in (C_k \cup P_{rel_0} \cup P_{obj_0}), t_2 \in P_{obj_0}, t_3 \in (C_k \cup P_{rel_0} \cup P_{obj_0})\} \tag{6.18}$$

6.18 expresses all statements at level C_k.

$$T_{a,j}(C_k) = \{(t_1,t_2,t_3) | t_j = a, (t_1,t_2,t_3,) \in T(C_k)\}, \ 1 \le j \le 3 \tag{6.19}$$

6.19 makes is possible to isolate those statements in C_k that are made about a particular object *a* in a particular role *j*.

Now it is possible to define the set of possible statements at each level. 6.20 below states the property of the set of statements at level 0, and 6.21 possible statements at level *i*.

$$STATEMENT_0 \subseteq (C_0 \cup P_{rel_0} \cup P_{obj_0}) \times P_{obj_0} \times (C_0 \cup P_{rel_0} \cup P_{obj_0}) \tag{6.20}$$

$$STATEMENT_i \subseteq (C_x \cup P_{rel_0} \cup P_{obj_0}) \times P_{obj_0} \times (C_y \cup P_{rel_0} \cup P_{obj_0}), \quad \forall x, y \in N \wedge x, y \leq i \wedge (x = i \vee y = i) \tag{6.21}$$

Now that the set of statements have been defined, it is also possible to express the relation interpretation on $R_1 - R_4$. 6.22-6.25 (below) does this.

$$PROP_\tau = \{(x, y) \mid (x, \tau, y) \in STATEMENT\}, \tau \in P_{rel_0} \tag{6.22}$$

$$PROP_{type} = \{(x, y) \mid (x, type, y) \in STATEMENT\}, \tau = type \tag{6.23}$$

All rules that were specified on the informal description of the language could be specified in this model but are left out on the account of this sections scope.

The presentation of the model is quite limited and general, e.g. does not define the satisfaction relation, but should indicate that the semantics of the language could be very well defined in this model.

6.1.4 The language and the Web

The triples encoding statements will stem from a number of sources: from annotations of Web pages, derived from databases, or conversation between Semantic Web agents etc. To ease the understanding, the figure 6.6 below illustrates that the set of triples (statements) that is possibly extracted from an annotated Web page, or returned from a triple database, has relations to Web resources. These relations are created due to the use of URIs.

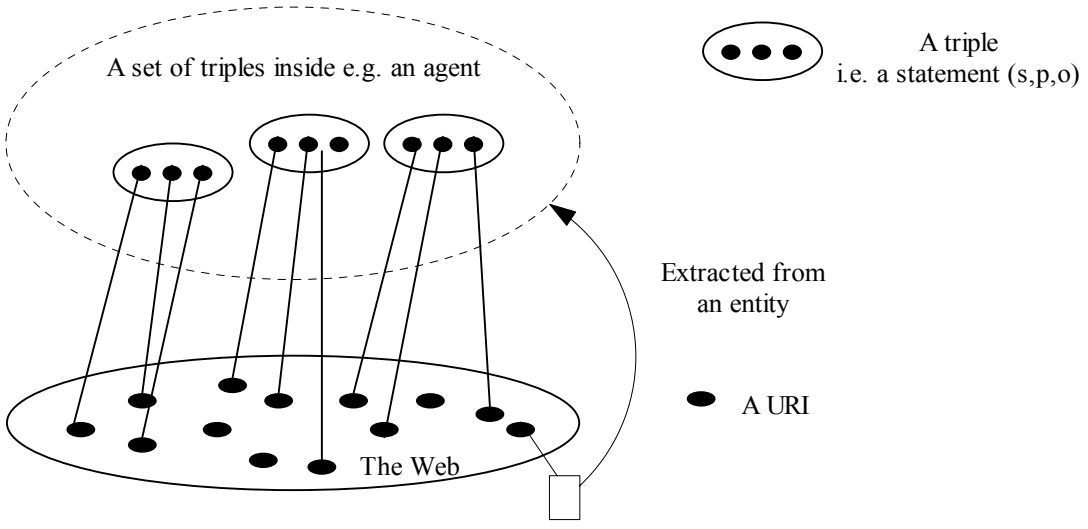


Figure 6.6

Statements attach properties to resources, i.e. create relationships between resources. By using properties that have a hand-coded meaning to machines, it becomes possible to automate the processing of these resources – but with a very limited interoperability. This is not how it should be done, but as said before, at this point we cannot express formally in the language what a property mean.

6.2 Level 2 – Concept description language

By using the simple "statement language" described previously, we can express binary relations. A set of binary relations could be used for a number of things; to attach properties to resources (i.e. metadata); model statements; model relational data; or build a complete computing language. In the context of the Semantic Web, the semantics of the relation has to be machine "understandable". If I make the statement "`http://www.people.org/benny` is the author of `http://www.docs.com/SWintro`" a machine can't, unless it processes natural languages, understand what *is the author of* formally means. This relation between `http://www.people.org/benny` and `http://www.docs.com/SWintro` does not make any sense to a machine. It is the semantics of the relation that makes the statement meaningful to a machine. And, there is another important thing to consider, and that is the precision of the relation. As said previously, anything of importance should have a URI and this also includes relations and properties. That means that the *is the author of* predicate should have a URI. But this only increases the precision; a machine still can't understand the relation. How do we formally define the "meaning" of a relation so that a machine can process it (and not hand-code it)?

To better see how we don't want to do it, let us look at how this is usually done in standalone applications nowadays. When we are developing an application that uses local information, e.g. from and to a file, we basically hand-code the application so that it "understands" a set of concepts. If we are designing a word processor application we could use a set of special characters for character formats, tables, lists, etc that form the concepts that the application "understands". All these special characters that are present in a file that the program has created has a certain meaning, but the meaning is hand-coded into the application. When the application reads a file that it has created it could interpret the special characters and then process it. The program uses these special characters to refer to some concept that are hand-coded in the application. This is quite similar to the way people understand words, because the meaning of this word: "balloon" is encoded in your brain, the word "balloon" is not self explained. If someone that has not the same hand-coded meaning between words and concepts reads "balloon", it does not mean anything. The same goes for word processor applications: If word processor A creates and saves a document, another word processor B will most likely not understand the file because the semantics of the files components is hand-coded into application A. This is absolutely not a scenario that we want on the Web!

When I use the word "understands" when talking about machines, I don't mean that the machine actually understands something. It means that the machine can use the word as a parameter for how the information could be processed. This is basically the only way a machine could understand something: it understands by mapping a concept to an algorithm that means that it could process the information. If we hand code the semantics, the interoperability between the applications becomes very limited. But if we explicitly declare the semantics of the information inside the actual information it becomes possible to achieve a greatly increased interoperability. In the case of the Semantic Web we need to specify explicitly what the semantics of the information is

because different applications will create information that other applications wants to consume.

Therefore, the information that is going to be present on the Semantic Web has to be self-describing. And, the semantics has to be "understandable" by a machine. But there exists another very important requirement: the meaning has to be global! If an application in Japan stumbles across information that an application in Sweden has created it should make no difference. This is archived by using URIs. In order not to reinvent the wheel, the approach taken by the AI community should first be considered.

6.2.1 Ontology

In the philosophical sense the term *ontology* is used as to mean: "The science or study of being; that department of metaphysics which relates to the being or essence of things, or to being in the abstract" [Oxford English Dictionary]. But the term has surfaced in the computer science community, especially in knowledge engineering, natural language processing, cooperative information systems, intelligent information integration, and knowledge management [Fensel, 2000], and its purpose is there described to offering: a shared and common understanding of some domain the can be communicated across people and application systems [Fensel, 2000] or enabling knowledge sharing [Gruber, 2000]. That is precisely what the Semantic Web needs. Thus, the concept of ontologies sounds reasonable to use. Though, it is important to always keep in mind that the Web is a universal and highly heterogeneous environment.

There exist numerous descriptions and definitions of what an ontology is as used in the computer science community: "a shared and common understanding of a domain that can be communicated between people and heterogeneous and distributed systems." [Fensel, 2000]; "a computer model of some portion of the world." [Huhns & Singh, 1997]; "a specification of a conceptualization." [Gruber, 2000]. Although the one by Gruber is most commonly used, and a bit further explained by him, a need for a more formal definition is needed to avoid confusions and to fully understand what constitutes an ontology.

6.2.2 Definition of ontology and related concepts

The definition of ontology that is used throughout is the one presented by [Guarino, 1998] that extends Gruber's definition, and it is as follows:

> "An ontology is a logical theory accounting for the *intended meaning* of a formal vocabulary, i.e. its *ontological commitment* to a particular *conceptualization* of the world. The intended models of a logical language using such a vocabulary are constrained by its ontological commitment. An ontology indirectly reflects this commitment (and the underlying conceptualization) by approximating these intended models."

This definition is also illustrated in figure 6.7 below, reproduced from [Guarino, 1998].

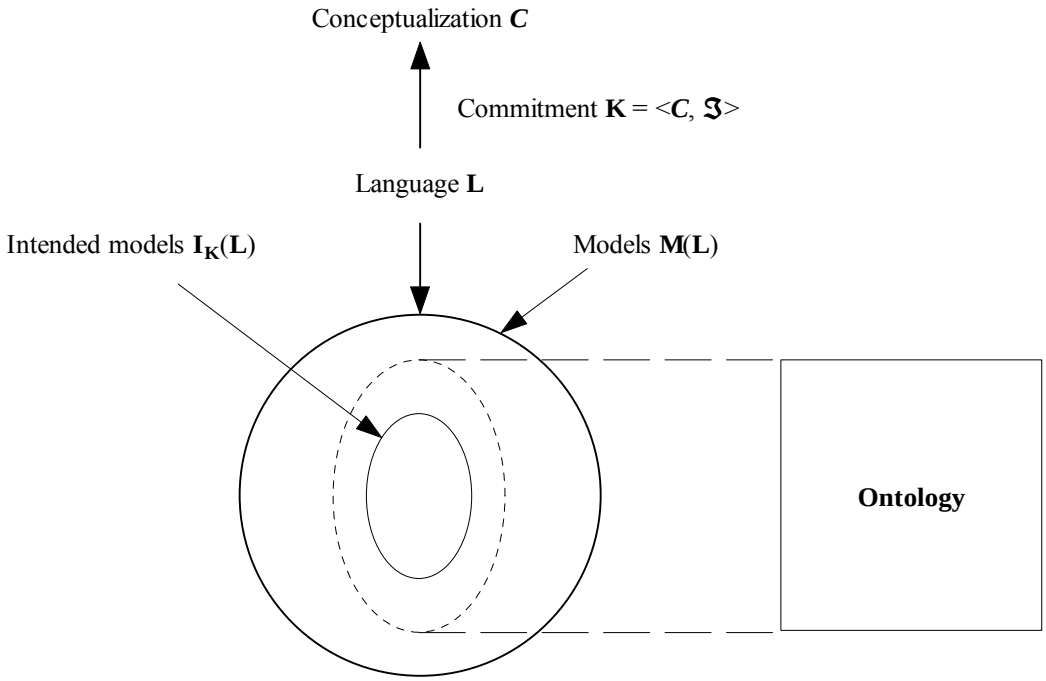


Figure 6.7

The definition clearly states very important properties that an ontology has. Each word used (for a relation or property) must explicitly declare its ontological commitment. In the case of the Semantic Web this means that we have to use URIs to achieve precision *and* to show its ontological commitment in the case of properties. The ontological commitment governs the use and meaning of the word, and as a consequence the applications' use of it. Further, an ontology is only a formal approximation of the world. Another important property that Guarino further states is that the ontology is language-dependent while the conceptualization is language-independent. This is in my opinion extremely important to notice. Two different ontologies might share the same conceptualization of a thing, e.g. the concept of price, but might map different words from their vocabulary to that concept. This means that it will become possible to achieve interoperability between different ontologies if the conceptualization is the same in some sense. But it also means that if two applications use the same words it does not mean that they map to the same conceptualization.

Every application could be considered to have its own conceptualization, and if two applications are going to be able to exchange "meaning" they have to have overlapping conceptualizations. As [Guarino, 1998] says, two systems A and B using the same language L can communicate only if the set of intended models $I_A(L)$ and $I_B(L)$ associated to their conceptualizations overlap. This is depicted in the left side of the figure 6.8 below:

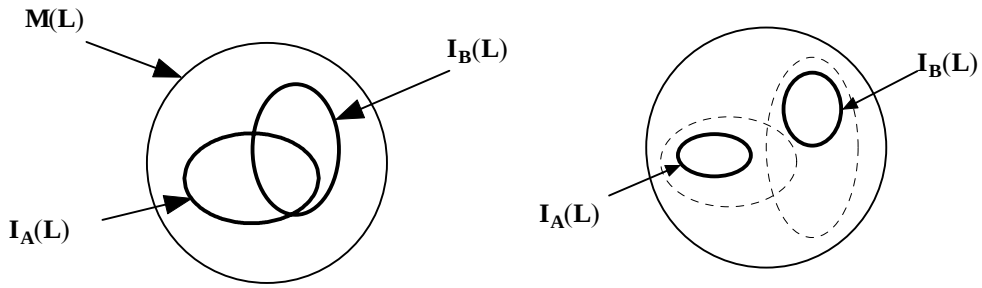


Figure 6.8

On the right side of figure 6.8 it is showed that even if the intended models of two different ontologies do not overlap, the axiomatizations (set of axioms) might overlap. An example of this if one ontology describes human – man relations and states a number of axioms, this entire set of axioms might be the same as another ontology that describes vehicle – car relations. This does not mean that the intended models overlap, and the applications committing to those ontologies cannot communicate (meaningfully).

But there exist a few notable limitations: an computer application can't generally decide if two concepts overlap, this has to be decided by humans and explicitly described by a human expert by explicitly express equal-statements that connects word with different ontological commitments. This is important to consider.

There exists a small confusion, which is important to bring order into, and that is the use and scope of ontologies. In [Studer et al., 2000] the authors state that: in order to reuse an ontology as much as possible, ontologies should be small modules with a high internal coherence and a limited amount of interaction between modules. This is in my opinion totally wrong. (Although is it true considering software reuse.) Every concept and every relation and every property should have a URI. Thus, using its URIs enables the reuse of an ontology. And most importantly, the true semantical interoperability will be possible IF ontologies are highly interconnected. The Semantic Web will reach its true potential if different ontologies that share concepts are interconnected. As [Hendler, 2001] says there is most probably not going to be one big ontology that everyone is using, rather it will be a big set of different possibly interconnected smaller ontologies. (Although, I would rather use *highly* instead of *possibly* interconnected.) Of course, an ontology in the biological domain will be very different from an ontology in the philosophical domain, but they might have concepts that overlap (e.g. the author of a document) thus connecting them.

How does this ontology concept relate to the language at level 1? All properties and classes of objects should be described by an ontology. An ontology is a bit simplified a set of interconnected properties and a set of logical properties. But in order to create this sort of property-networks we need a set of atom primitives, and set of standard properties that constitute the building blocks for connecting properties. Using these constructs to build a network of properties and then use this network to map properties of statements onto is what using an ontology is all about. Also, if we construct a property-network partial understanding is possible. If a set of statements uses a set of properties (i.e. a set of nodes in the properties network) partial understanding could be possible if another application uses a set of properties that are "near by" or closely related. By following routes in the network there might be a common base property that facilitates the partial understanding. This also means that isolated networks of properties are of lesser use than a big combined network (i.e. do not invent ad hoc properties without first looking for existing ones). What is generally needed in order to create an

ontology is the ability to model classes of meanings and their relations [Huhns & Sing, 1997]. Thus, we need the possibility to create a kind of dictionary that defines the terms that will be used in statements and gives specific meaning to them.

6.2.3 Informal features and syntax constraints

When designing layered applications or languages there are always a number of decisions that have to be made on how to divide the application or language into suitable and intuitive layers. The reason for small (thin) layers considering languages is that the engineering task becomes easier and it makes is possible for simple applications to use only parts of the language and still manage to use it. One big layer would make all the applications that use the language fairly complex. The ontology concept requires that the language could express logical axioms or rules that describe behavior and the relationship between concepts. Including logic into a language makes it very powerful and quite complex, therefore this level will only handle the structuring of an ontology, which also makes it possible to test a set of statements of their ontological commitment. This will anyway become a language of surprising usefulness and power without the ability to directly express logic.

What we want to model in an ontology, besides logical properties, is basically the types of entities that exist in our ontology and using the concepts of classes, subclasses, properties, subproperties and instances usually does this. A *concept description language* (CDL) is a language that allows us to represent classes of objects, instances of classes, subclass relationship, and properties of classes and instances [Dean et al., 1995]. The reason for creating CDL is that general theorem proving in e.g. predicate calculus is computationally complex, therefore one often wants to use a subset that is expressive enough to represent a given knowledge but has less complex computational properties [Dean et al., 1995]. These properties are very suitable for the language at this level. We don't what to be to be as expressive as full predicate logic yet, but still want to capture the "description" of the ontology and use it to make small deductions.

There are basically two different entities used to describe the structure of an ontology: classes and properties. A class is a type of entity that we are interested in representing. Stating that an object is an instance of a class of objects makes it possible to treat different object from the same class in the same way, and this is very useful as will be explained later. The same goes for properties. A certain property is an instance of a property "class", and this is also useful when comparing objects that have the same property types. To this end we need the concept of web-resource classes and class instances, and the concept of property hierarchies.

As an example of the concept of CDL and its use, a small set of concepts and their relations is illustrated in the example in figure 6.9.

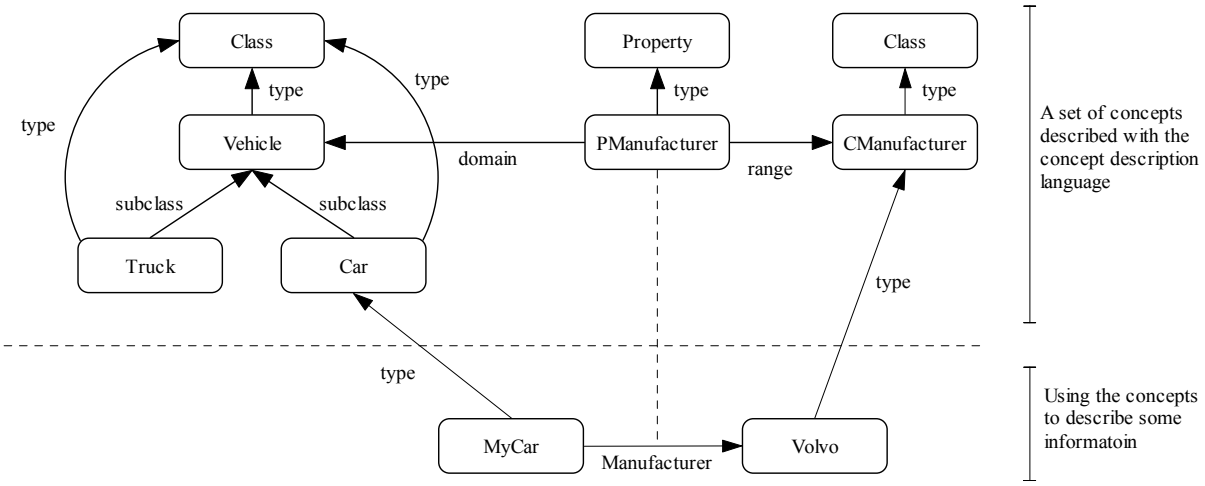


Figure 6.9

The part above the horizontal (and dotted) line is a description of a set of concepts: *Vehicle*, *Truck*, *Car,* and *CManufacturer* are classes, and *PManufacturer* is a property, all described by the CDL. These concepts are then used by the lower part of figure 6.9 to describe some information. The statement below the line states that the resource *MyCar* is of the type *Car* and has the property *PManufacturer* with the value *Volvo*.

In figure 6.9 a concept node (a property or class) is represented by a resource in the "webified" concept description language described here.

To be able to create these concepts (properties and resource types) a set of primitives has to be introduced. In the example the introduced primitives are `Class`, `Property`, `subClassOf`, `domain`, and `range`. All these primitive are used in RDFS [Guha, & Swick, 2000], a language for expressing RDF vocabularies. The semantics of the primitives in RDFS is the same as used here not to conflict with the common use in the Semantic Web community. The `type` primitive was introduced in the previous language layer as a primitive in order to model reified statements but will have an important role here too. This set (as small as possible) of necessary primitives has to be known by every application that needs to use them (i.e. the semantics of the primitives has to be hand-coded into the application – therefore as few as possible). This is the only way to create higher-level languages.

The difference between class and property is that an instance of a property can be used as an element in any position in a statement. But an instance of a class can only appear as the object or subject, not as the predicate of a statement. The fact that a resource could be both the property (in the predicate position of a statement) and the object of a statement makes reification and self-reference possible [Champin, 2000]. Others state that this does not make the language able self-referential but self-expressible [Conen & Klapsing, 2000].

Another important fact is that we need primitives to be able to convert a set of words in an ontology into another set of words from another ontology (e.g. a *equivalent* primitive). This will be explained later.

There has been work done to formalization of these primitives, e.g. in [Campin, 2000], [Conen & Klapsing, 2000] and [Fikes & McGuinness, 2001]. There are also efforts done to extend the logical possibilities of RDF, e.g. [Boley, 2000] and [Staab, 2000], on model ontologies. These papers are all focused on RDF but offer a great deal of help in stating the semantics of the language presented here since much of RDFS model is reused.

6.2.4 Classes

Resources on the Web represent a wide diversity of entities. A resource could represent a person, a company, an introduction, a novel, a car, and so on. If all these resources are assumed to have nothing in common with each other, machines will have a difficult time trying to help us humans. If we want our agent to find the cheapest car within a ten mile radius, how could this be possible is we could not use a general concept that encompasses the general concept of car? To make things easier for machines we need to classify resources, so that similar resources can be treated in similar ways. The most common way in computer science is to introduce the concept of classes (types). A class is used to denote a commitment. If a resource declares itself to be a member of a certain class (an instance of a class) it also makes a commitment that the resource will "obey" the rules and properties that a certain class declares. (This is similar to commitment regarding ontologies.) If the concept of classes and instances of classes is used, searching for the cheapest car is much easier since the agent knows that it can treat all resources that are declared to commit to a certain class (e.g. car) in the exact same way. Classes are also useful for the partial understanding, i.e. if the agent finds a resource that is a subclass of a class that the agent can process, it can at lest treat the found resource in the more general way, as defined by the class that the agent can process. For the sake of reuse it is also important that a class could be refined, i.e. to reuse the common declaration by to extend it. But the real benefit of sub classing is the partial understanding that becomes possible. This will be explained in the latter parts of this section.

A class is a resource denoting a set of resources by the property `type` [Champin, 2000].

$$\forall s : statement(s, TYPE, CLASS) \Rightarrow class(s) \tag{6.26}$$

According to (6.26) a statement that states that a resource *s* has the property `type` valued with the primitive value `class` is defined to be a class.

$$\forall s,o : statement(s, TYPE, o) \wedge class(o) \Rightarrow instanceOf(s,o) \tag{6.27}$$

A statement that states that a resource *s* has the property `type` valued with a resource *o* that is also a class, then *s* is an instance of the class *o*. This is similar to how the instantiation of classes in object oriented programming works. To create the structure that an otology needs, the concept of subclass us used.

$$\forall s,o : statement(s, SUBCLASSOF, o) \wedge s \neq o \wedge class(o) \Rightarrow subClassOf(s,o) \tag{6.28}$$

A statement that states that the resource *s* has the primitive property subClassOf valued with a resource that is a class creates a subclass *s* to *o*. The *subClassOf* predicate should be interpreted as a subset relation between the relations they denote [Champin, 2000] and therefore it is also transitive. This means that $subClassOf(s,o) \Rightarrow s \subseteq o$. In RDF it is not allowed to have circles in the subclass or property hierarchy.

$$\forall c_1,c_2,c_3 : subClassOf(c_1,c_2) \wedge subClassOf(c_2,c_3) \Rightarrow subClassOf(c_1,c_3) \tag{6.29}$$

There has been some debate whether cycles should be allowed in concepts graphs, i.e. if the requirement $c_1 \neq c_3$ is too strong. Some, e.g. [Champin, 2000] states that cycles could be used to express equality between concepts, i.e. as defied below:

$$\forall c_1,c_2 : subClassOf(c_1,c_2) \wedge subClassOf(c_2,c_1) \Rightarrow equal(c_1,c_2) \qquad (6.30)$$

Since subClassOf is a subset relation, the predicate *equal*(c_1, c_2) implies that $c_1 \subseteq c_2$ and $c_2 \subseteq c_1$. If the predicate subClassOf(c_1, c_2) is true and predicate *equal*(c_1, c_2) is false that means that $c_1 \subset c_2$. As the language explained here is only going to use the smallest possible set of primitives the above definition of the predicate *equal* is used. But then there is a problem of how indirect cycles should be interpret. Looking at subClassOf as a subset relation states that a cycle of more than two concepts that are not equal is not meaningful. The set relationships $c_1 \subset c_2 \subset c_3 \subset c_1$ is not valid, i.e. it makes no sense. Therefore, cycles are only allowed to express equality between two concepts (or sets of equal concepts).

Like object oriented programming, an instance of a class is also an instance of all of its super classes. This makes partial understanding achievable.

$$\forall i,c_1,c_2 : instanceOf(i,c_1) \wedge subClassOf(c_1,c_2) \Rightarrow instanceOf(i,c_2) \qquad (6.31)$$

6.2.5 Properties

The properties are resources used as predicates in the statements, and the semantics of a statement (s,p,o) therefore depends heavily on the predicate [Champin, 2000]. A resource that is used as a predicates is regarded as a certain property. Since a property is a resource and resources are treated as first-class objects, a property is also a first class object [Champin, 2000]. It is possible to make assertions about properties, i.e. a resource that is a property could be used as predicate, subject, and object of a statement. As indicated in figure 6.9 previously, properties and classes can be defined independently of each other [Champin, 2000]. The rules below have similar meaning as the class primitive, and basically, properties and classes are very similar, the point in which they differ is in the positions they occur in statements.

$$\forall s : statement(s,TYPE,PROPERTY) \Rightarrow property(s) \qquad (6.32)$$

$$\forall s,o : statement(s,SUBPROPERTY,o) \wedge s \neq o \Rightarrow subPropertyOf(s,o) \qquad (6.33)$$

The *subPropertyOf* should be interpreted as a subset relation between the relations they denote [Champin, 2000] and therefore it is also transitive.

$$\forall p_1,p_2,p_3 : subPropertyOf(p_1,p_2) \wedge subPropertyOf(p_2,p_3) \Rightarrow subPropertyOf(p_1,p_3) \qquad (6.34)$$

A property denotes a relation between two resources. To facilitate the Semantic Web, the machines have to understand what the property means, i.e. what computations could be done on the resources that have a certain relation. At the current stage, the property concept is too general. Imagine that we specify a property fatherOf that is intended to mean that the subject of a statement that has the predicate as fatherOf represents the father of the child that the object of the statement represents. Further, we specify a class `humans` that the resources representing the father Jim and the daughter Sara are instances of. Using this, the statement (Jim, fatherOf, Sarah) is meaningful regarding the concept that we want the relation to model. But how should a statement (Sweden,

fatherOf, smorgasbord) be interpreted? Of course the statement is meaningless according to the concept that the fatherOf relation denotes, but how could a machine know this? Erroneously, the machine interpreting this might deduce that Sweden and smorgasbord are instances of the `humans` class. What would be useful for a machine is if the use of properties were constrained to be applicable to certain classes. Then, if a property stated that is only usable if the subject and object where of declared classes, then a machine could use it to deduce if a statement is meaningful.

To make the properties more useful the concept of *domain* and *range* is introduced as in RDFS. The domain restricts to what classes the property might be asserted to and the range restricts the classes that might appear as object in those statements.

$$\forall s, o, i : statement(s, DOMAIN, o) \wedge type(o, CLASS) \Rightarrow domain(s, o) \qquad (6.35)$$

The rule above states that if a statement asserts that the domain of a property *s* is *o* and *o* is of type class then *o* is the domain of *s*. The semantics of range is very similar as expressed below.

$$\forall s, o, i : statement(s, RANGE, o) \wedge type(o, CLASS) \Rightarrow range(s, o) \qquad (6.36)$$

The use of domain and range is really useful for the partial understanding.

Table 6.2 below informally describes the primitives that have been introduced at this level of the language.

Predicates:	Objects (primitive resources):
`subClassOf` Used to creates hierarchies of classes	`class` This primitive is used with the `type` property to declare that a resource is of type class
`subPropertyOf` Used to creates hierarchies of properties	`property` This primitive is used with the `type` property to declare that a resource is of type property
`domain` Restricts the types that a property can be applied to. It is the subject that is restricted.	
`range` Restricts the types that a property can be applied to. It is the object that is restricted.	

Table 6.2

This language that is used to describe properties has a little more power but is not Turing complete [Berners-Lee, TLK]. Even if the language is not so powerful at this level, it is extremely useful and can provide a structure that enables agents to process information that it was not intended to from the beginning. There is also an ability to achieve partial understanding, which is very important. The concepts that provide this are the subclass, subproperty, and the equal concept. The concepts introduced in this layer do not need to be understood by all agents on the Semantic Web. Specialized agents limited to using a predefined language will not [Champin, 2000]. If an agent that is hand-coded to process a certain set of classes and properties stumble across information that is described in a way that the agent cannot directly process it might be

possible to achieve partial understanding. Agents processing metadata will be able to trace the origins of the concept descriptions that they do not understand back to concept descriptions that they do understand so they could process it even though they weren't directly designed to do that. If a robot knows the concept of parent and finds a resource that is a father and also finds a statement that says that father is a subproperty to parent, the agent could treat the resource that is a father as a parent.

6.2.6 Semantics

The model that was presented in section 6.1.3 needs to be extended with a few primitives and rules to handle that language at this second level. After the introduction of the primitives that have been explained in the previous sections we consider the language to be:

$$L = \{URIF, R_0, R_1, R_2, R_3, R_4 R_5, R_6, R_7, R_8, c_0, c_1, c_2\} \tag{6.37}$$

As before, the language L consists of the set *URIF* of all possible URIs with possible a fragment id appended. R_0 is relation symbol (denoting a 3-ary relation), $R_1 - R_8$ are binary relation symbols, and $c_0 - c_2$ are constant symbols. To simplify the understanding of this section, the relation between the language L and the terms used in the previous chapter could be: R_0 is `triple`, R_1 is `type`, R_2 is `subject`, R_3 is `predicate`, R_4 is `object`, R_5 is `subClassOf`, R_6 is `subPropertyOf`, R_7 is `domain`, R_8 is `range`, c_0 is `statement`, c_1 is `class`, and c_2 is `property`.

The model M for L is as before:

$$M = \langle WEBRES, \alpha_{obj}, \alpha_{rel} \rangle \tag{6.38}$$

In the following only the added elements, sets, and rules will be shown. 6.39 and 6.40 shows the added object and relation interpretations.

$$\begin{aligned}
&\alpha_{obj}(R_5) = subClassOf \\
&\alpha_{obj}(R_6) = subPropertyOf \\
&\alpha_{obj}(R_7) = domain \\
&\alpha_{obj}(R_8) = range \\
&\alpha_{obj}(c_1) = class \\
&\alpha_{obj}(c_2) = property
\end{aligned} \tag{6.39}$$

$$\begin{aligned}
&\alpha_{rel}(R_5) = PROP_{subClassOf} \\
&\alpha_{rel}(R_6) = PROP_{subPropertyOf} \\
&\alpha_{rel}(R_7) = PROP_{domain} \\
&\alpha_{rel}(R_8) = PROP_{range}
\end{aligned} \tag{6.40}$$

The most important difference between this model and the model that was explained in the previous language level is the ability to create new properties. It is now possible to make a statement that creates a property and then make statements about this property that expresses constraints on how it could be used. The properties could then be used at

higher levels to express the properties of objects. So the model now has to incorporate this. If nothing else is stated the variables *i*, *j*, and *k* are natural numbers. The set *C* is defined as before.

$$C_0 = WEBRES \tag{6.41}$$

$$C_i = \cup_{\forall \sigma \in C_{i-1}} [T_{\sigma,1}(C_{i-1})] \tag{6.42}$$

6.43 then define the set of all objects (an object at higher levels represents a statement).

$$\overline{C} = \cup_{\forall i} C_i \tag{6.43}$$

As with *C* the set of primitive properties of *P* is defined to be the set P_0. P_0 contains the objects that 6.44 define.

$$\begin{aligned} P_0 = \{&PROP_{type}, PROP_{subject}, PROP_{predicate} PROP_{object}, \\ &PROP_{subClassOf}, PROP_{subPropertyOf} PROP_{domain}, PROP_{range}\} \end{aligned} \tag{6.44}$$

To define *P* for other values than 0, the operator *T* defined below has to be used.

$$P_i = \{\delta \mid \delta \in T_{type,2}(C_i) \wedge \delta \in T_{property,3}(C_i)\} \quad i > 0 \tag{6.45}$$

$$\overline{P} = \cup_{\forall i} P_i \tag{6.46}$$

The operator *T* looks a bit different since properties can be created, and thus, be present at levels other that 0. 6.47-6.49 defines *T* for this level of the language. (The modification is basically that a variable for selecting "property level" is included.)

$$T(C_{k,i}) : C_k \rightarrow (C_k \cup P_i) P_i (C_k \cup P_i) \tag{6.47}$$

$$T(C_k, i) = \{(t_1, t_2, t_3) \mid t_1 \in (C_k \cup P_i), t_2 \in P_i, t_3 \in (C_k \cup P_i)\} \tag{6.48}$$

$$T_{a,j}(C_k) = \{(t_1, t_2, t_3) \mid t_j = a, (t_1, t_2, t_3,) \in T(C_k, k)\}, \; 1 \leq j \leq 3 \tag{6.49}$$

It is now possible to define the set of possible statements at each level. This can be done by using the operator *T* or with *P* and *C*. 6.50 below states the set of possible statements at level 0, and 6.51 possible statements at level *i*.

$$STATEMENT_0 \subseteq (C_0 \cup P_0) \times P_0 \times (C_0 \cup P_0) \tag{6.50}$$

$$\begin{aligned} &STATEMENT_i \subseteq (C_a \cup P_b) \times P_c \times (C_d \cup P_e), \forall a,b,c,d,e \in N \wedge \\ &a,b,c,d,e \leq i \wedge (a = i \vee b = i \vee c = i \vee d = i \vee e = i) \end{aligned} \tag{6.51}$$

Now that the set of statements have been defined, it is also possible to express the relation interpretation on $R_5 - R_8$. 6.52-6.55 (below) does this.

$$\alpha_{rel}(R_5) = PROP_{subClassOf} = \{(x, y) \mid (x, subClassOf, y) \in STATEMENT\} \quad (6.52)$$

$$\alpha_{rel}(R_2) = PROP_{subPropertyof} = \{(x, y) \mid (x, subPropertyOf, y) \in STATEMENT\} \quad (6.53)$$

$$\alpha_{rel}(R_3) = PROP_{domain} = \{(x, y) \mid (x, domain, y) \in STATEMENT\} \quad (6.54)$$

$$\alpha_{rel}(R_4) = PROP_{range} = \{(x, y) \mid (x, range, y) \in STATEMENT\} \quad (6.55)$$

This section does not specify the semantics of the operations that where presented informally in the previous section or the satisfactory relation. They could be rather easily moved into this model from the descriptions in the previous sections.

6.2.7 The language and the Web

There is no difference in the actual representation of the data at this level. Everything is expressed with triples as on the previous level. To relate all this to the Web, and to see the relationships between the different roles that a resource can play in a statement, figure 6.10 below depicts the conceptual and abstract view of how triples (statements) extracted from an entity are related to other triples in different roles.

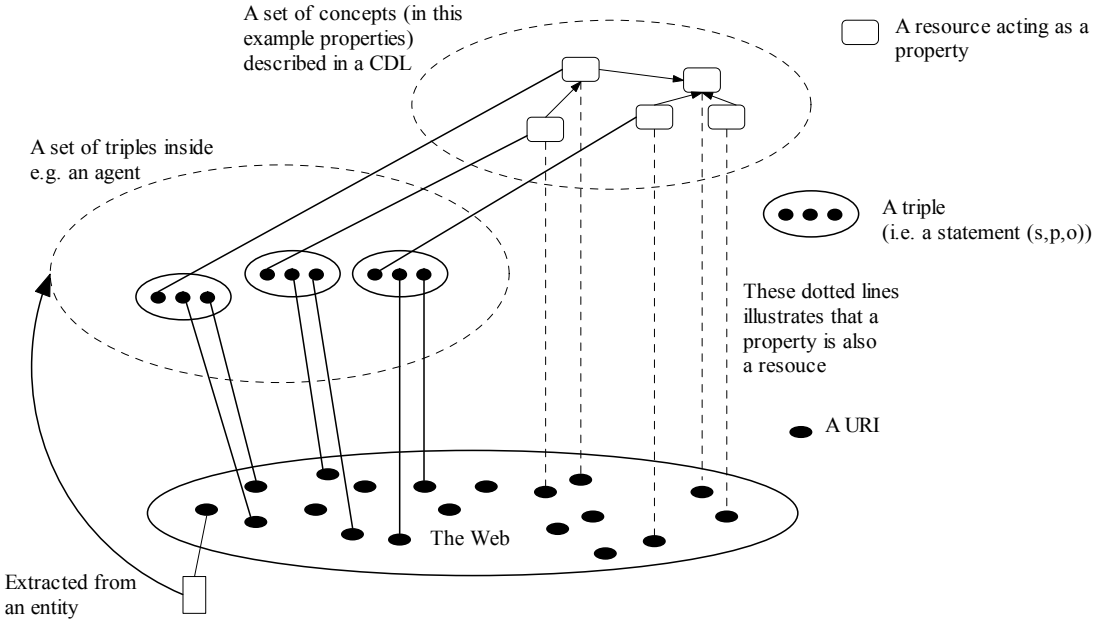


Figure 6.10

What figure 6.10 illustrates is that a triple that represents a statement has one of its components (the predicate) associate with a resource that act as a property. The other two components (the object and subject) are associated with resources. The types of resources that they could be associated with are determined by the domain and range of the property. As the image illustrates, it is possible to make statements *about* properties, as they also are resources. Thus, a resource of the type property can occur as predicate, object, or subject of a statement.

6.3 Level 3 – Logic language

The Semantic Web is a universal space for anything that can be expressed in classical logic [Berners-Lee, LL]. The previous level enabled us to create a concept description in the artificial language. A concept description is not a complete ontology since it is not possible to express logical axioms or rules that govern the logical properties of the

concepts. At the previous level, it was possible to state that a property *x* is symmetric, but the meaning (logically) had to be hand-coded into the application that uses the information for it to be useful. Therefore, the previous level made it possible to create a web of properties and restrict to what resources they could be applied, but it was not possible to express what they meant logically. Even if the language could not express logic it is a very useful language, so the decision not to include logic is very reasonable at that stage. The next step in this language level is to introduce a set of primitives that makes it possible to express logic. If this is introduced, a property is not only a name with an implicit meaning (hand-coded), but also a name and a logical declaration of its intended meaning. It would then be possible to state that this property is symmetric by describing it logically. An agent that read a statement that says that the `http://company/property/coworker` property is `http://company/logic/symmetric` could, by retrieving the `http://company/logic/symmetric` document, conclude that `http://company/logic/symmetric` is equal to $\forall x, y : f(x, y) \Rightarrow f(y, x)$ where *f* is `http://company/logic/symmetric`. Then the statement that says that `http://company/employees/jim` has `http://compaly/property/coworker` `http://anothercompany/people/Sara` implies that "Sara" has coworker "Jim". This might at first not seem so powerful, but what is also possible is to declare logical properties that are far more complicated. This is a huge step forward for interoperability. What is needed is a common language for expressing this logic, and that is what this level should introduce.

6.3.1 Existing logical Web languages

There exist a few systems that use RDF as an underlying framework for expressing logic. Metalog [Marchiori & Saarela, 1999] is a three-layered system where the first layer extends RDF with logical connectives and facilitates the creation of complex inference rules that encodes logical reasoning. The second layer of Metalog is simply an interpretation of the RDF language, analogous with the rules expressed in the previous parts. The third layer is a language interface to writing structured data and reasoning rules [Marchiori & Saarela, 1999]. Although the system is not fully developed, and not an active project, it is useful to look at the design decisions that they made and take that into account as this section also should enrich the language with logical capabilities.

The reason for enriching RDF with logic in Metalog is to enable to encode dynamic reasoning rules as in logical programming [Marchiori & Saarela, 1999]. To this end, [Marchiori & Saarela, 1999] states that in order to do so one needs at least the connectors `and`, `or`, `implies` and variables. To ease the use, further connectives are introduced, e.g. `not` and math operators. As for the operators `and`, `or`, `implies` these are very simple to introduce and there is no need for important design decisions. On the on the other hand, the logical interoperation of the `not` operator is a bit more problematic. All logic-programming systems have the basic property that something that cannot be proven to be true is assumed to be false, and this is a closed-world assumption [Louden, 1993]. The implementation of the `not` operations in this context is often "negation as failure" (NAF) which means that the goal `not(X)` succeeds whenever the goal `X` fails. Using NAF in closed-worlds can have the property that adding information reduces the number of things that can be proven from the set of information, which leads to nonmonotonic reasoning [Dean et al., 1995]. Metalog interpret not as NAF, i.e. Metalog opted for a closed-world assumption with the motivation that they consider the Web to be a distributed knowledge basis with possible partial information available [Marchiori & Saarela, 1999]. This type of decision does

not need to be made when "designing" the language that has the role of a declarative knowledge language. The language does not need a specific inference engine.

Another useful consideration taken with the Metalog system is the one regarding the computability. Metalog extends RDF to first order predicate calculus, i.e. increases the expressibility, but does not impose computability restrictions since its primary goal is to provide expressibility means to codify logical data and relationships [Marchiori & Saarela, 1999]. But as a facility, there is a possibility to add values to a tag that expresses that the stated information uses a restricted syntax: *datalog* or *logic programming*. As an example, datalog queries always terminate, i.e. it uses a finite amount of time. As this indicates, Metalog is a purely declarative language [Marchiori & Saarela, 1999]. This type of decisions could be considered later in the language design, e.g. at the syntax creation stage.

SHOE [Heflin, 2000] is a language aimed at the Semantic Web that is not built on top of RDF; instead it is built by extending HTML. Even so, it is still very useful to see examine the logical capabilities of SHOE and what design decisions and compromises that the language design faced. SHOE combines features from knowledge representation, Datalog, and ontologies, but also from predicate logic and frame systems [Heflin, 2000]. As the general Semantic Web language described here, SHOE is not an inference system; it is simply a language that could be used to express facts that could be used by an inference engine to derive new facts. This is something that the language described here shares with SHOE.

The meaning of the information that an instance of the SHOE language describes is connected to the use of an ontology. The interpretation of an ontology in SHOE is that it is a tuple $< V, A >$ where V is the vocabulary and A is the set of axioms that governs the theory [Heflin, 2000]. Further, V is basically a set of predicate symbols with arbitrary arity while the set A is a set of *definite program clauses* that have the standard logical semantics [Heflin, 2000]. A definite program clause is a Horn clause that has at least one antecedent and exactly one consequent. Since the ontology uses axioms in Horn clauses SHOE is restricted to a subset of predicate calculus. An important statement that is made in [Heflin, 2000] is that they envision a Web in which SHOE search engines differentiate themselves by the degree of inference supported and the size of the underlying knowledge base. This statement is generally extendible, and it is possible to predict two extreme cases of applications:

- General applications with large knowledge bases with very limited inference capabilities (as an example of an application with a huge "knowledge base" and no inference capabilities at all is an ordinary search engine on the Web)
- Domain targeted and highly specialized applications with a small knowledge base but extremely high inference capabilities (an example could be an agent that tries to find out as much as possible).

6.3.2 The logical language

This section is mainly intended to explain how logical primitives could be introduced in the language to make it more powerful. The resulting language is only a limited version of the "full" FOPC logical language that could be developed. The use of the words "true" and "false" are simplified since the interpretations in a model are left out in the text.

To develop the language there are a few decisions that have to be made, especially on how powerful logic we need. If we intend to be able to express meaning as a natural language we need higher-order logic (HOL) [Dean et al., 1995], and the agents that uses the information certainly sometimes need modal logic to manipulate the different

contexts that the statements occur in. But as this is a layered approach, it seems reasonable to introduce primitives so first propositional logic and then predicate calculus could be expressed. This would take the language to a very expressive level. If the expressiveness of a language increases, the computational properties generally get worse. As an example, there is no algorithm that is guaranteed to terminate in a finite number of steps that determines if a statement follows from a set of statements in *PC* [Dean et al., 1995]. The restriction of the language to *PC* in this elaboration of the Semantic Web language is the fact that the value that *PC* offers and the problems of HOL. (The language could be extended quite easily at a latter stage.)

To move the language up to propositional calculus *P* we need `AND` or `OR` or `imply`, and `NOT`. From any of these two set of primitives it is then possible to e.g. create `NAND` or `NOR` which is operators that are complete by them self [Aho & Ullman, 1995]. `AND` is already included, implicitly, in the language by juxtaposition, i.e. two statements stated one after the other are both equally trusted in the context that they occurred. Two statements, S1 and S2 that are true in context C is the AND of {S1, S2} [Berners-Lee, TB]. The use of context makes it possible to group statements, by using reification and grouping statements, and thereby create AND sets. To make the language as powerful as *P* we need to introduce NOT. Using reification and a set of primitive properties we could create all new features that are needed in the language. To see this, imagine that we are searching to make a negation of a statement S1 below:

```
S1: {Jim, has_access_to, Area01}
```

In the following, statements will be illustrated with a graph. Generally, a statement consists of two nodes and a labeled arc, where the label is the predicate, the node at the tail of the arrow is the subject, and the nod at the head of the arrow is the object. (The previous image 6.1 illustrates this.) S1 is illustrated in figure 6.11 below.

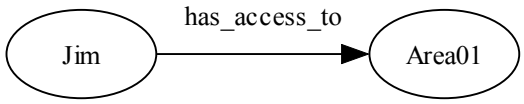


Figure 6.11

Which states that Jim may access area01. If we instead want to express that Jim may *not* access area01, we have to model the statement S1 as a reified statement and then create a property, e.g. `truth` or `not`, and make the value of the property false (i.e. NOT true). This means that we make a statement about a statement, which does not explicitly exist as an assertion. Statements S1 – S4 below are used to represent the statement that we want to negate, and S5 attaches the property NOT with the value true. The value true might seem a bit odd, but is quite arbitrary, the importance is the NOT property.

```
S1: {s, type, statement}
S2: {s, subject, Jim}
S3: {s, predicate, has_access_to}
S4: {s, object, Area01}

S5: {s, NOT, true}
```

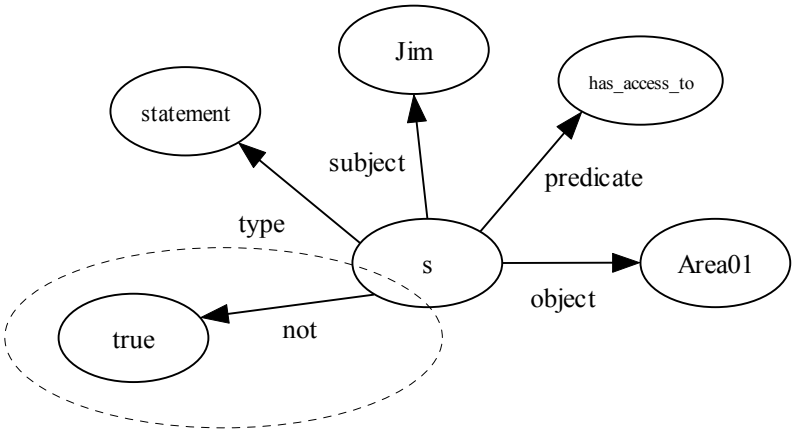


Figure 6.12

Figure 6.12 illustrates what this set of statements looks like modeled in a graph. The representation of statement S5 is in the dotted line. What we have done here to introduce NOT is to make an assertion about the statement that is to be negated. To enable this, one primitive has been introduced: a binary predicate (property) `not`. The resource true is strictly not needed as 6.22 below indicate.

6.21 is the rule from 6.6 in the first level of the language that is reproduced here to help the understanding of the `not` predicate.

$$\forall s', s, p, o : statement(s', TYPE, STATEMENT) \wedge statement(s', SUBJECT, s) \wedge statement(s', PREDICATE, p) \wedge statement(s', OBJECT, o) \Rightarrow reifies(s', s, p, o) \quad (6.21)$$

Using that rule, we can now specify what the not property means, 6.22 descries the rule.

$$\forall s', s, p, o, x : reifies(s', s, p, o) \wedge statement(s', NOT, x) \Rightarrow \neg statement(s, p, o) \quad (6.22)$$

6.22 states that if a statement s' reifies a statement `(s,p,o)` and that s' has a property `not`, that the statement `(s,p,o)` is not true.

At this point the language has AND and NOT, but to increase the expressiveness that language needs quantifiers, i.e. the ability to express statements about a set of objects.

To introduce a universal quantifier (or existential) in this language is quite simple. This is done by creating a new primitive property and using reification. In figure 6.13 the property `true_for_all` has been introduced and connected to the subject to the reified statement.

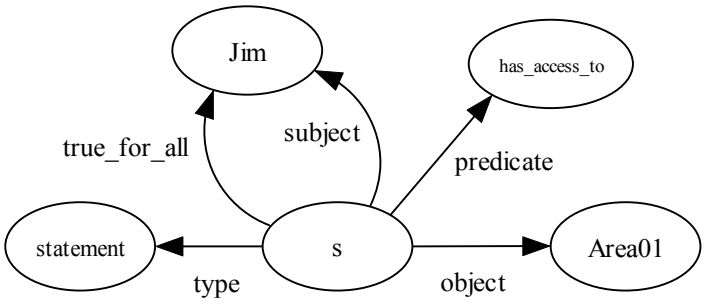


Figure 6.13

The semantics of this is that we have a statement `(Jim, has_access_to, Area01)` that we have reified with statement s, and by stating the statement `(s, true_for_all, Jim)`, the interpretation is that all statements that have `has_access_to` as predicate and `Area01` as object are true for all subjects (i.e. everybody can access Area01). 6.23 descries figure 6.13 logically.

$$\forall Jim : statement(Jim, has\_access\_to, Area01) \tag{6.23}$$

A more useful example is to model the statement that if someone works on project 01 then they have access to area 01. This statement is modeled by introducing a new primitive property: `imply`. This property has the properties of the logical imply operation. The statement that anybody that works on project 01 can access area1 is illustrated in image 6.14 below.

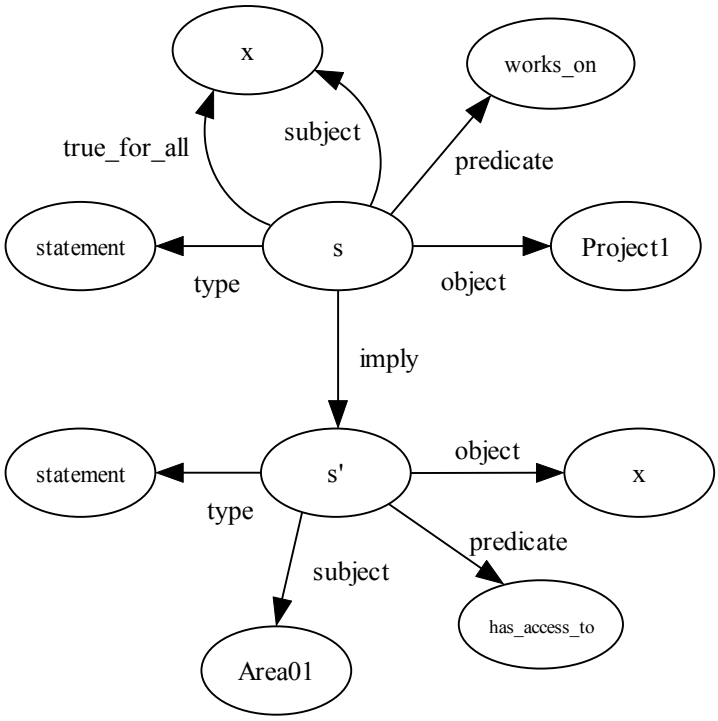


Figure 6.14

The logical meaning of this set of statements is given in 6.24. This general construction of this statement is two reified statements s and s'. These two resources, s and s', can be though of as the body and the head of an ordinary imply operator (i.e. s $\Rightarrow$ s').

$$\begin{aligned} &\forall x : statement(x, works\_on, Project1) \Rightarrow \\ &statement(x, has\_access\_to, Area01) \end{aligned} \tag{6.24}$$

The general logical rule for this type of statements is given below. Rule 6.25a states what is true when one of the statements components are used in a true_for_all predicate, while 6.25b states what is true what two statement components are used in the true_for_all predicate. The other cases are simple to derive from those.

$$\forall s,s_1,p_1,o_1,s',s_2,p_2,o_2,: reifies(s,s_1,p_1,o_1) \wedge reifies(s',s_2,p_2,o_2) \wedge$$
$$statement(s,true\_for\_all,s_1) \wedge statement(s,imply,s') \wedge s_1=s_2 \Rightarrow \quad (6.25a)$$
$$statement(s_1,p_2,o_2)$$

$$\forall s,s_1,p_1,o_1,s',s_2,p_2,o_2,: reifies(s,s_1,p_1,o_1) \wedge reifies(s',s_2,p_2,o_2) \wedge$$
$$statement(s,true\_for\_all,s_1) statement(s,true\_for\_all,o_1) \wedge \quad (6.25b)$$
$$statement(s,imply,s') \wedge s_1=s_2 \wedge o_1=o_2 \Rightarrow statement(s_1,p_2,o_1)$$

The quantifier scope is, when it is unconstrained, universal but by including contexts, i.e. group statements into contexts, it is possible to have more control on the scope. Since this section is mainly intended to show how logical capabilities could be layered on top of the language the scope issue is not further discussed.

There are other statements that one wants to make that describe the logical properties of a property. The graph below (6.15) states that if *x* knows *y*, then *y* knows *x*. This symmetric property of knows is now possible to express in the language as illustrated below in 6.15

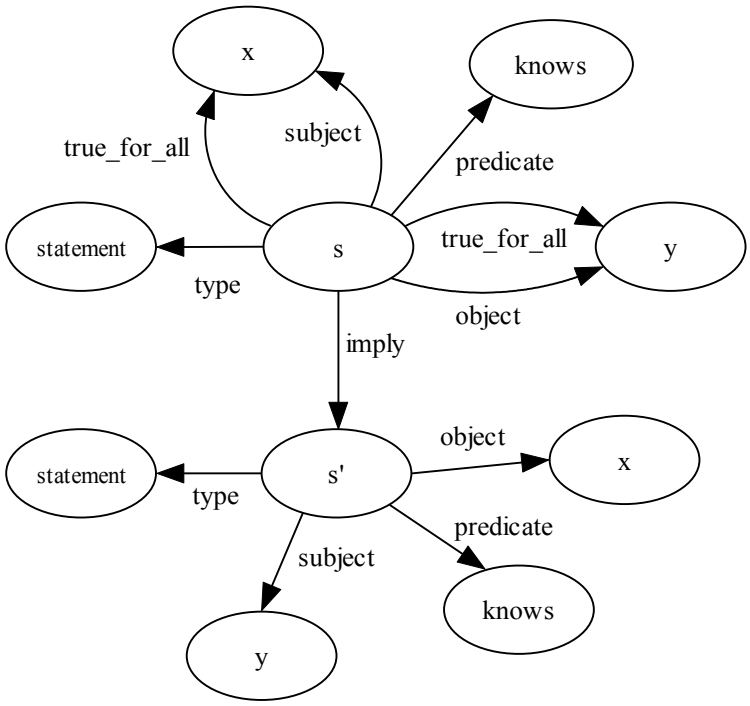


Figure 6.15

The logical meaning of this statement illustrated in figure 6.15 is described in 6.26.

$$\forall x, y : statement(x, knows, y) \Rightarrow statement(y, knows, x) \quad (6.26)$$

Rule 6.25b previously stated the general rule for the conclusion made in 6.26.

There are cases when the statements that we want to express are like: “if this and that are true, then this is true”. In the previous examples the body of the implication was only one statement, but often there is a need for more. The example below in figure 6.16 is an illustration of a statement that expresses that the `bigger_than` property is transitive. To model this there is a need to introduce one more primitive. In this case the `body` primitive is introduced to with the logical interpretation that the node at the tail of

the arc is an “argument” to an implication or future operations. The `body` property is also needed to be able to group several statements into the body of the implication. Another point of the introduction of this primitive is that a computer reading the statements needs to be able to see if other statements govern that statement. In the example there is also a resource `head1` that simply is used as a bag that collect a set of body arguments.

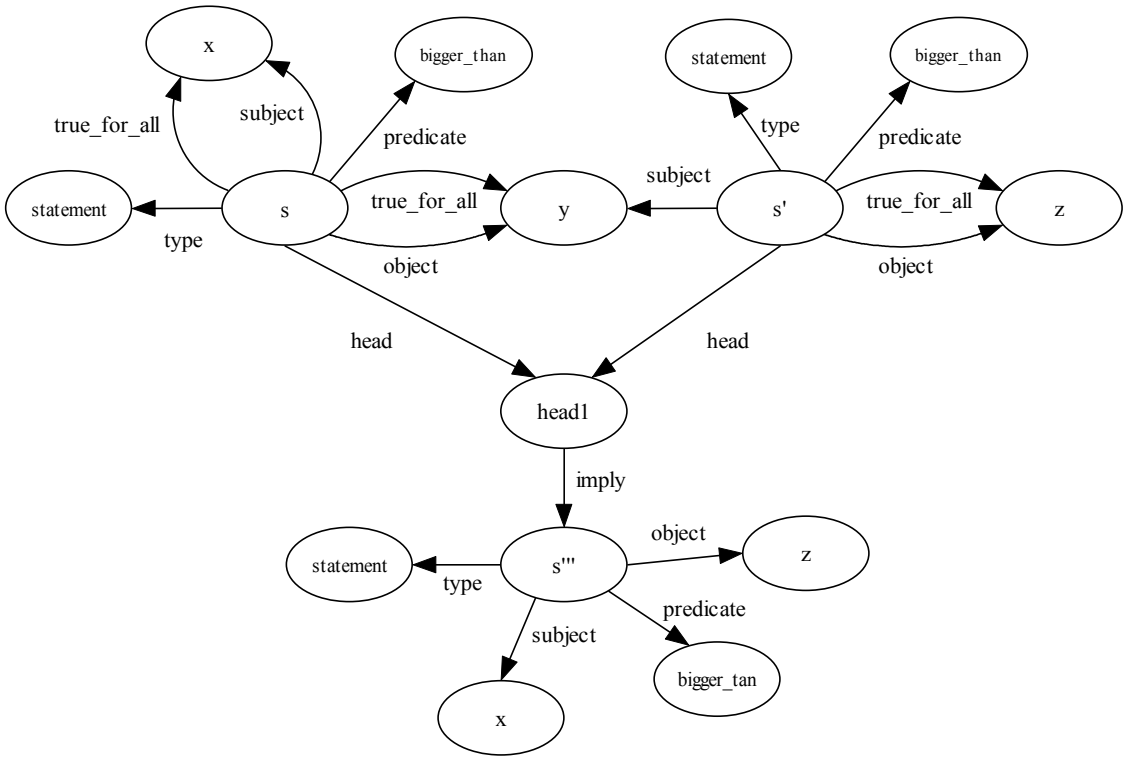


Figure 6.16

The logical interpretation of this example is in 6.27.

$$\forall x, y, z : statement(x, bigger\_than, y) \wedge statement(y, bigger\_than, z) \Rightarrow statement(x, bigger\_than, x) \quad (6.27)$$

The general rule for this conclusion is rather long and excluded, but essentially is a concatenation of two rules as 6.25a or 6.25b (depending on the number of quantified variables) and a control that there is no other statements that is in the head of the imply predicate to come to the conclusion.

The figure in 6.16 is quite verbose, but remember that this is not how the language will be processed, only how the semantics of the language is connected.

Table 6.3 below is an informal summary of the primitives that have been introduced this section.

Predicate:	Objects: (primitive resources)
`not` The not primitive is introduced to negate statements that have been reified.	
`true_for_all` This predicate is used to express that one	

of a reified statement components should be treated as a universally quantified variable.	
`imply` The imply predicate is used to make more that one reified statement the body of a implication.	
`head` The head predicate is make it possible to explicitly state that a reified statement is acting in a head of a implication or similar future operations.	

Table 6.3

This layer has the ability to define your own logical primitive and explain what they mean – logically. This means that the language is really powerful in terms of expressiveness. All this expressiveness is achieved by introducing fifteen primitives that also enables further features to be expressed thanks to the use of reification. Expressing information on this level makes it possible for a machine that operates at this level to learn new concepts by looking at logical declarations of the concepts, and since it comprehend all the primitives that are used, it can process it. The language could be further extended to introduce other features but the core semantics will almost always be expressible in this language.

6.3.3 Semantics

The model does not differ fundamentally on this level from the other levels and is therefore not discussed. Generally, all primitives (predicates and subjects) have to be introduced in the model and defined with the model constructs. One important thing to consider is that the model is more "powerful", in some sense, than the language since the language cannot make statements about statements without using reification. The model can make statements bout statements without reification, and can model reification.

6.3.4 The language and the Web

To relate the data that this level could express to the Web and the previous level figure 6.17 below illustrates how all the languages levels conceptually creates different networks (Webs) of connected concepts.

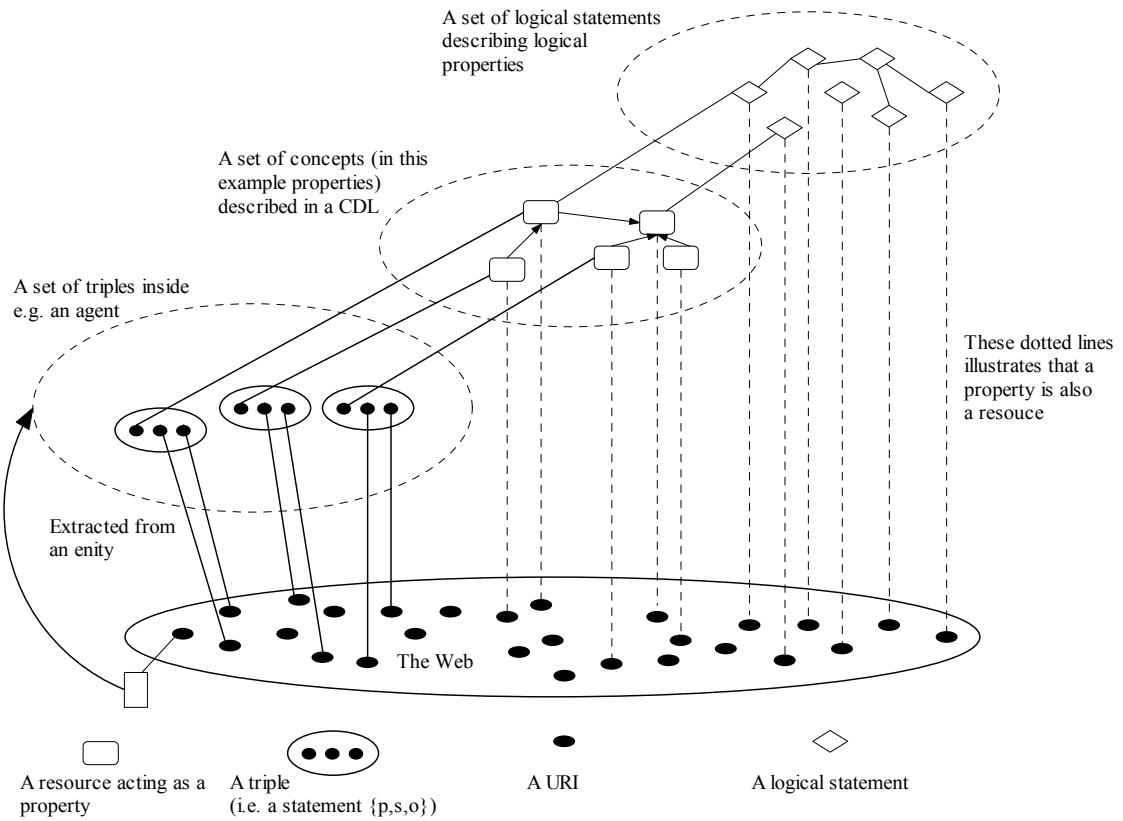


Figure 6.17

6.4 Summary of the language and the Semantic Web it could create

The language that has been described in the previous sections is the language to be used on the Semantic Web, and therefore is aimed at machine consumption. Further, the language was described by looking beyond the syntax, concentrating on the semantics of the language. Once the semantics of a language is settled, the specifications that maps language constructs onto syntax is much easier, than to capture the semantics of a "syntax language". To practically use the language syntax needs to be developed, but that is not in the scope of this thesis, and would not offer any real value.

One important aspect is that the user of this language will not process it as pure triples. Internal representations that increase the effectiveness of the language will be used, but the core semantics of the language will always be described in that way.

Another thing that is important is that the Semantic Web in not an inference system is it a declarative knowledge base to be used by tailored inference engines. Different types of inference engines could use this knowledge base, and use it in the different ways.

The various figures that has been shown previously that shows the relationship between the Web and the data that the Semantic Web language produces can now be used to explain what the Semantic Web is and its conceptual difference from the Web. The figure 6.18 below shows the Semantic Web.

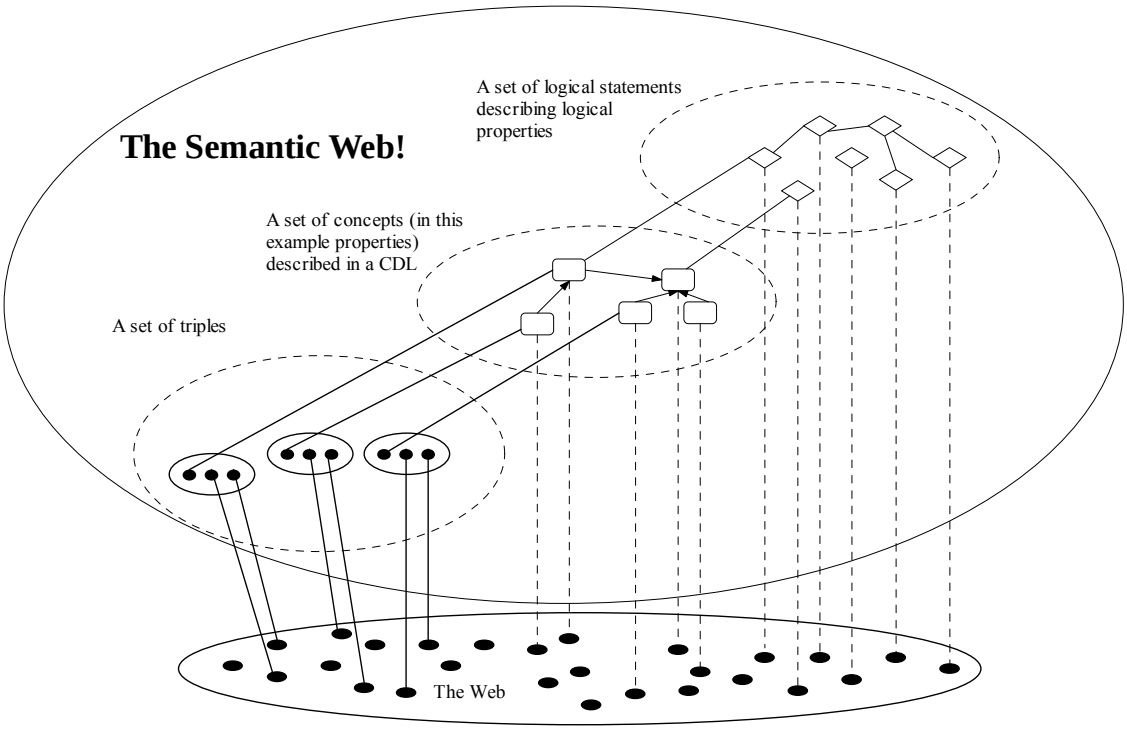


Figure 6.18

What this should indicate is that the Semantic Web language that is described in this thesis *creates* the Semantic Web. Therefore, to fully understand the Semantic Web one needs to know what the Web is and the semantics of the language. This thesis delivers formalization of both these issues.

7 Conclusion

The main results of this thesis are formal definitions of the Web and related concepts, but also a rigorous explanation and discussion regarding the semantics of the language that is to be used on the Semantic Web. The former result was a necessity to ground later results. The following sections state the most important observations and results that have been achieved in this thesis.

7.1 The World Wide Web and Uniform Resource Identifiers

The need for a formal and rigorous definition of the Web was apparent as it is often regarded as a fuzzy creation. There was also some confusion of the Web and its relation to the Internet and how entities could be represented. Furthermore, the use of the Web in the Semantic Web vision in an extension to how it has traditionally been used. Current standards of the Uniform Resource Identifier [RFC2936] and [RFC1630] is too ambiguous, particularly when it comes to *what* a URI actually identifies. The use of URIs as identifier has been to identify "resources", but the concept "resource" never been fully and formally defined, i.e. the concept "resource" is explained in [RFC2936] as "anything that has identity" and this is far to abstract and general because it is used as a formal definition. The lack of formal definitions have then been more apparent as other applications/languages have been constructed by using this unclear understanding of a resource, e.g. Resource Description Framework [Lassila & Swick, 1999]. As in the case of RDF, the wide interpretation of what a resource is and what a URI really identifies has been a major hindrance to RDF and similar applications evolution.

The problem with the previous "definitions" is that they were both regarded as formal definitions and as abstract informal descriptions. The formal definition of the concept of URIs and the Web is far more complex than one might first assume.

What the definitions in this thesis provide is a formal definition of the Web, and what a URI identifies. The definition makes it possible to better communicate unambiguously, and provides an abstraction concept that is to be used as the second part of the thesis.

The grounding definition is the one about the Web, and that it is the URI-space. Older definitions often regarded the Web as an abstract information space, but that is more what it enables, or what it is used for than a definition of the Web. Regarding the Web as the URI-space has not been stated explicitly before, only implicitly in informal descriptions. The definition of the Web as the URI-space also grounds what a URI is: a point in the URI-space. Further, there is a common belief that a URL is a direct mapping to a representation of some information, but that is not entirely correct since that hides that fact that a URL has to be mapped onto an algorithm first. The URL *and* the algorithm then maps onto a representation of some information. This makes it possible to state what a URI represent, and that is nothing directly. But a URI that has a relation to an algorithm might represent some information entity. This is important to know, but it is too cumbersome to use practically when talking about the Web, and the use of fragment identifiers in the Semantic Web community conflicts with the conservative use of URIs. Therefore, an abstraction *resource* is used to explain what a URI identifies, and what a resource represents. Using this abstraction it is clear that a URI identifies a resource, and a resource represent an entity on the Web. Thus, resources are the only entities that are "on" the Web. An entity is represented on the Web if there is a URI that identifies a resource that has a function that maps onto that entity.

This might seem a bit too theoretical but as the Semantic Web basically is a set of statements about URIs, what a URI represents is very important to know.

7.2 The language on the Semantic Web

The Semantic Web is a vision that in certain contexts sounds too futuristic and a bit indefinable. Discussions on what could be done on this ”future web” are often similar to science fiction in some sense. Even if these visionary thoughts are important, it is also dangerous if it is the only explanation that exists of the Semantic Web. What the Semantic Web comes down to is to create a machine targeted language that is to be used on the Web, creating a Semantic Web for the machines. This Semantic Web of machine processable and semantically interlinked data is then to be used so as to ease the human consumption of information from the Web. The Semantic Web is built for human by humans, but machines do the tedious work on the Semantic Web.

The focus in this thesis is on the language that is to be used to create the Semantic Web. This language is for the Semantic Web what HTML is for the Web, but a bit more complex and directly aimed at machines. The description of the language described in this thesis is focused on the semantics of the language. The reasons for this approach is that at this early stage too much is uncertain to be able to discuss the syntax of the language, and the mapping from semantics to syntax is easier than the other way around.

In order to not “reinvent the wheel” the semantics of the language described in this thesis borrows ideas from a wide set of similar and dissimilar languages such as RDF [Lassila & Swick, 1999], RDFS [Brickely & Guha, 2000], Notation 3 (N3) [Berners-Lee, N3], SHOE [Heflin et al., 2000], Metalog [Marchiori & Saarela, 1999], KIF [Genesereth & Fikes, 1992], OIL [Horrocks et al., 2000], CG [Sowa, 2001]. To a large extent, all these languages/models have a common semantics, but differ in syntax and terminology, and in some cases as to whether they are adapted to the Web or not. Therefore, the focus was to explain the semantics of the language on the Semantic Web, not any particular syntax. As the Semantic Web community uses the terminology in RDF(S) that terminology where chosen in this thesis.

The language description consists of a set of layers, where each layer provides a set of primitives with formal semantics that higher layers use, combined with introducing new language primitives to produce a more powerful language. The benefit from this layered approach is that applications could be developed to only comprehend a part of the language stack, naturally limiting the capabilities. The language description provided here consists of three layers; statements, concept descriptions, and logic as depicted in figure 7.1

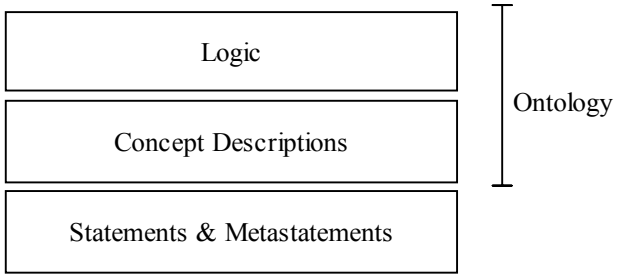


Figure 7.1

What the semantics of the language shows is that the language on the Semantic Web is basically a mesh of old technologies, modified to be usable on the Web. The technologies and theories are quite simple, and have a strong theoretical background in e.g. Artificial Intelligence. The use of the result of this thesis is as a theoretical background for those who need to understand the future language on the Semantic Web, and it shows how the language creates the Semantics Web.

References

[Aho & Ullman, 1995]
Aho A & Ullman J, 1995: *Foundations of Computer Science*, Computer Science Press

[Berners-Lee, 2000]
Berners-Lee T, 2000: *Weaving the Web*, Harper Business

[Berners-Lee, AU]
Berners-Lee T, Axioms of URIs, URL: http://www.w3.org/DesignIssues/Axioms.html

[Berners-Lee, AN]
Berners-Lee T, 2001: RDF Anonymous nodes and quantification, URL:
http://www.w3.org/DesignIssues/Anonymous.html

[Berners-Lee, GR]
Berners-Lee T, 2000: Generic Resources, URL:
http://www.w3.org/DesignIssues/Generic.html

[Berners-Lee, IE]
Berners-Lee T, 2000: Rules and Facts: Inference engines vs. Web, URL:
http://www.w3.org/DesignIssues/Rules.html

[Berners-Lee, LL]
Berners-Lee T, 2000: The Semantic Web as a language of logic, URL:
http://www.w3.org/DesignIssues/Logic.html

[Berners-Lee, ME]
Berners-Lee T, 2000: Meaning, URL: http://www.w3.org/DesignIssues/Meaning.html

[Berners-Lee, N3]
Berners-Lee T, 2001: Notation 3, URL://www.w3.org/DesignIssues/Notation3

[Berners-Lee, RE]
Berners-Lee T, 1998: What the Semantic Web can represent, URL:
http://www.w3.org/DesignIssues/RDFnot.html

[Berners-Lee, RM]
Berners-Lee T, 1998: Semantic Web Road map, URL:
http://www.w3.org/DesignIssues/Semantic.html

[Berners-Lee, ST]
Berners-Lee T, 1999: The Semantic Web Toolbox: Building Semantics on top of XML-RDF, URL: http://www.w3.org/DesignIssues/Toolbox.html

[Berners-Lee, TLK]
Berners-Lee T, 2000: XML 2000 Keynote Slides, URL:
http://www.w3.org/2000/Talks/1206-xml2k-tbl/

[Berners-Lee, WA]

Berners-Lee T, 1998: Web Architecture from 50,000 feet, URL: http://www.w3.org/DesignIssues/Architecture.html

[Berners-Lee, WM]
Berners-Lee T, 1998: The Web Model, URL: http://www.w3.org/DesignIssues/Model.html

[Brickley & Guha, 2000]
Brickley D & Guha R, 2000: Resource Description Framework (RDF) Schema Specification 1.0, URL: http://w3.org/TR/rdf-schema

[Dalen, 1980]
Dalen D, 1980, *Logic and Structure*, Springer-Verlag Berlin Heidelberg

[Champin P et al., 2001]
Champin P, Euzenat J, Mille A, 2001: Why URLs are good URIs, and why they are not, URL: http://www710.univ-lyon1.fr/˜champin/urls/urls.pdf

[Champin P, 2000]
Champin P, 2000: RDF Tutorial, URL: http://www710.univ-lyon1.fr/˜champin/rdf-tutorial/rdf-tutorial.pdf

[Conen & Klapsing, 2000]
Conen W & Klapsing R, 2000: A Logical Interpretation of RDF, Linköping Electronic Articles in Computer and Information Science ISSN 1401-9841 Vol. 5 (2000) nr 013 URL: http://www.ep.liu.se/ea/cis/2000/013/

[Dean et al., 1995]
Dean T, Allen J, Aloimonos Y, 1995: *Artificial Intelligence – Theory and Practice*, Addison-Wesley

[Ebbinghaus et al., 1996]
Ebbinghaus H, Flum J, Thomas W, 1995: *Mathematical Logic*, Springer Verlag New York

[Fensel, 2000]
Fensel D, 2000: Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce, URL: http://www.cs.vu.nl/%7Edieter/ftp/paper/silverbullet.pdf

[FNC, 1995]
FNC Resolution: Definition of the Internet, URL: http://fnc.gov/Internet_res.html

[Genesereth & Fikes, 1992]
Genesereth M & Fikes R, 1992: Knowledge Interchange Format, Version 3.0 Reference Manual, Technical Report Logic-92-1, Computer Science Department Stanford University, URL: http://www-ksl.stanford.edu/knowledge-sharing/papers/kif.ps

[Gruber, 2000]
Gruber T, 2000: What is an Ontology?, URL: http://www-ksl.stanford.edu/kst/what-is-an-ontology.html

[Guha, 1995]
Guha R, Contexts: A Formalization and Some Applications, Stanford PhD Thesis URL: http://www-formal.stanford.edu/guha/guha-thesis.ps

[Guarino, 1998]
Guarino N, 1998: Formal Ontology and Information Systems, Amended version of a paper appeared in N. Guarino (ed.), Formal Ontology in Information Systems, Proceedings of FOIS'98, Trento, Italy, 6-8 June 1998, Amsterdam, IOS Press, pp. 3-15 URL: http://www.ladseb.pd.cnr.it/infor/Ontology/Papers/FOIS98.pdf

[Halsall, 1995]
Halsall F, 1995: *Data Communications, Computer Networks and Open Systems*, Addison-Wesley

[Hansen, 1994]
Hansen K, 1994: *Grundläggande logic*, Studentlitteratur

[Heflin et al., 2000]
Heflin J, Hendler J, Luke S, 2000: SHOE: A Prototype Language for the Semantic Web, Linköping Electronic Articles in Computer and Information Science ISSN 1401-9841 Vol. 6 (2001) URL: http://www.ep.liu.se/ea/cis/2001/

[Hendler, 2000]
Hendler J, 2000: DARPA Agent Markup Language, URL http://www.darpa.mil/iso/DAML/DAML_ISO_World.ppt

[Hendler, 2001]
Hendler J, 2001: Agents and the Semantic Web, URL: http://www.cs.umd.edu/users/hendler/AgentWeb.html

[Horrocks et al., 2000]
Horrocks I, Fensel D, Broekstra J, Decker S, Erdmann M, Globe C, Harmelen F, Klein M, Staab S, Studer R, Motta E, 2000: The Ontology Inference Layer, URL: http://www.cs.vu.nl/~dieter/oil/Tr/oil.pdf

[Humns & Singh, 1997]
Humns N & Singh M: Ontologies for Agents, IEEE internet Computing, Nov – Dec 1997, http://computer.org/internet

[Lassila & Swick, 1999]
Lassila O & Swick R, 1999: Resource Description Framework (RDF) Model and Syntax Specification, URL: http://www.w3.org/TR/REC-rdf-syntax

[Louden, 1993]
Louden K, 1993: *Programming Languages – Principles and Practice*, PWS

[Marchiori & Saarela, 1999]
Marchiori M & Saarela J, 1999: Towards the Semantic Web: Metalog, URL: http://www.w3.org/RDF/Metalog/CIKM-050299.html

[Melnik & Decker, 2000]

Melnik S & Decker S, 2000: A Layered Approach to Information Modeling and Interoperability on the Web, In proceedings of the Workshop "ECDL 2000 Workshop on the Semantic Web" URL: http://www.ics.forth.gr/proj/isst/SemWeb/proceedings/session1-2/paper.pdf

[Oxford English Dictionary]
Oxford English Dictionary, URL: http://www.oed.com

[RDF-Interest Group]
URL (archive): http://lists.w3.org/Archives/Public/www-rdf-interest

[RDF-Logic]
URL (archive): http://lists.w3.org/Archives/Public/www-rdf-logic

[RFC1630]
Berners-Lee T, 1994: Universal Resource Identifiers in WWW
Network Working Group, Request for comments: 1630

[RFC2396]
Berners-Lee T et al., 1998: Uniform Resource Identifiers (URI): Generic Syntax,
Network Working Group, Request for comments: 2396

[Russell & Norvig, 2001]
Russell S & Norvig P: Artificial Intelligence: A Modern Approach, Introduction, URL: http://www.cs.berkeley.edu/~russell/intro.html

[Sommerville, 1995]
Sommerville I, 1995: *Software Engineering*, Addison-Wesley ISBN: 0-201-42765-6

[Sowa, 2000]
Sowa J, 2000: Ontology, Metadata, and Semiotics, Published in B. Ganter & G. W. Mineau, eds., Conceptual Structures: Logical Linguistics, and Computational Issues, URL: http://www.bestweb.net/˜sowa/peirce/ontometa.htm

[Sowa, 2001]
Sowa J, 2001: Conceptual Graphs, URL:
http://www.bestweb.net/˜sowa/cg/cgstandarg.htm

[Studer et al., 2000]
Studer R, Decker S, Staab S, 2000: Situation and Perspective of Knowledge Engineering http://www.db.stanford.edu/˜stefan/paper/2000/ios_2000.pdf

[URI]
URL (archive): http://lists.w3.org/Archives/Public/uri

[W3C DesignIssues].
URL: http://www.w3.org/DesignIssues