

State of the Art Paper

The Foundation for Semantic Interoperability on the World Wide Web

Marut Buranarach

Submitted in partial fulfillment
of the requirement for the degree of
Doctor of Philosophy

**Department of Information Science and Telecommunications
School of Information Sciences
University of Pittsburgh**

November 8, 2001

Table of Contents

ABSTRACT.....	5
OVERVIEW	6
1. INTEROPERABILITY IN INFORMATION SYSTEMS.....	10
1.1 <i>Definition</i>	10
1.2 <i>Approach to achieve interoperability</i>	11
1.2.1 Standards	11
1.2.2 Layered approach	11
1.2.3 Mediators.....	12
1.2.4 Wrappers	13
1.3 <i>Toward semantic interoperability in information systems</i>	13
2. SEMANTICS AND ITS REPRESENTATION	15
2.1 <i>Semantics</i>	15
2.1.1 Definition	15
2.1.2 Declarative semantics vs. Procedural semantics.....	15
2.1.3 Intension vs. Extension	16
2.2 <i>Knowledge Representation</i>	17
2.2.1 Definition	17
2.2.2 Logic and computation.....	18
2.2.3 Ontology.....	21
2.2.4 Approaches in representing knowledge	22
A. Frame-based systems.....	22
B. Semantic Networks.....	22
C. KL-ONE	25
D. Description Logic.....	28
E. Nonmonotonic Reasoning: KR under uncertainty.....	30
3. INTEROPERABILITY ON THE WEB.....	33
3.1 <i>Markup Languages</i>	33
3.2 <i>Hypertext Markup Language (HTML)</i>	35
3.3 <i>Extensible Markup Language (XML)</i>	37
3.3.1 XML Document	38
3.3.2 XML as a Data Interchange Format for the Web.....	41
3.3.3 XML Schema	42
4. WEB SEMANTICS.....	45
4.1 <i>Metadata</i>	45
4.1.1 Definitions.....	45
4.1.2 Dublin Core.....	45
4.1.3 Warwick Framework.....	47
4.1.4 Resource Description Framework (RDF)	50
A. RDF Design Goals.....	50
B. RDF Data Model	53
C. RDF Syntax	56
D. RDF Schema.....	57
4.1.5 Topic Maps.....	59
4.2 <i>Semantic Web</i>	61

4.2.1 Definition	61
4.2.2 The roles of XML and RDF to the Semantic Web.....	61
5. SEMANTIC INTEROPERABILITY ON THE WORLD WIDE WEB	63
5.1 <i>Ontology on the Web</i>	63
5.2 <i>An ontology language for the Web</i>	65
5.2.1 Simple HTML Ontology Extension (SHOE).....	67
5.2.2 Ontology Inference Layer (OIL).....	71
5.2.3 The DARPA Agent Markup Language (DAML)	77
5.3 <i>Ontology in distributed environment</i>	83
5.4 <i>Ontology in Electronic Commerce</i>	86
APPENDIX.....	88
A. <i>Description Logic</i>	88
B. <i>Nonmonotonic Logic and Reasoning</i>	94
C. <i>An DAML+OIL (March 2001) Ontology Example</i>	98
D. <i>World Wide Web Consortium (W3C) Recommendation Track</i>	104
REFERENCE LIST	105
ACKNOWLEDGEMENT.....	121

Table of figures

Figure 1: Semantic Web architecture by Berners-Lee	8
Figure 2: Mediator architecture	12
Figure 3: Wrapper architecture	13
Figure 4: Heterogeneity in information systems	14
Figure 5: Example of semantic networks	23
Figure 6: Example of a KL-ONE concept	27
Figure 7: Map of built-in data type in XML Schema	43
Figure 8: Warwick Framework architecture	49
Figure 9: RDF data model for a basic statement	53
Figure 10: RDF data model for a high-order statement	55
Figure 11: Ontology Spectrum	64
Figure 12: OIL's layered language model	72
Figure 13: Ontology Integration methods	85
Figure A1: TBox, ABox and its relationship	89
Figure A2: Reasoning services in Description Logics	92
Figure D1: Possible transitions of the Recommendation track	104

Table of tables

Table 1: Syntax rules of first-order logic in Backus-Naur Form (BNF)	19
Table 2: An analogy of RDF terms to OOP, Frame systems and Description Logics	66
Table 3: SHOE ontology example	68
Table 4: SHOE instance example	70
Table 5: An OIL ontology example	73
Table A1: Syntax rule of \mathcal{FL} language	90
Table A2: Syntax rule of \mathcal{ALC} language	91

Abstract

This paper reviews and describes works considered a foundation for semantic interoperability on the World Wide Web (WWW). The paper focuses on the semantic interoperability efforts that build on the WWW standards. The paper is divided into five parts. The first part summarizes general approaches to achieve interoperability in information systems. The second part provides a review of literature and research works in Knowledge Representation. The third part discusses the history and current achievement of interoperability on the WWW. The fourth part reviews and describes the efforts to provide semantics for WWW documents. The final part reviews research attempts to bring these efforts together in order to achieve the semantic interoperability on the WWW. Ontology is considered a key toward this achievement.

Overview

The exchange of information and the integration of information between systems is known as *Interoperability* (Vckovski, 1999). Interoperability becomes more important when the number of interacting systems increases. There are currently two major approaches toward interoperation of information systems. One is by having an agreement about interaction. This is known as standardization. When a single standard cannot be agreed to, a modular standard may be used. The modular, or layered, approach alleviates the difficulty by breaking the standard into multiple levels. By making the standard modular, the impact of changes in one standard can be isolated from others (Spring, 1996). When it is not possible to have everyone agree on a single or layered standard, transformation of data between systems may be necessary. This is the second approach. Transformation may make use of mediator (Wiederhold, 1992) and wrapper architectures. When transformation is involved, semantics becomes more important (Sheth, 1999).

The representation of semantics is important to the creation and exchange of meaning between communicating parties. Knowledge Representation (KR) studies how to transform the expression of meaning into a formal representation such that a machine can use it as knowledge. There have been two major approaches of semantics modeling: declarative and procedural semantics. Using declarative semantics, a program can determine simple logical consequences of what it is told from what it previously knew. This is regarded as similar to human common sense (McCarthy, 1960). Major knowledge representation approaches include frame-based representation (Minsky, 1975), semantic network (Quillian, 1967), KL-ONE (Brachman & Schmolze, 1985) and Description Logics.

The World-Wide-Web (WWW) is a huge distributed information system. With billions of Web pages and millions of users, interoperability is a key to its success. The Hypertext Markup Language (HTML) is the de facto standard for Web documents. HTML is a Document Type defined by Standard Generalized Markup Language (SGML). As such,

and by nature of its design, HTML is not extensible (Bosak, 1997). While SGML provides the features lacking in HTML, it was considered too complex for use in Web tools. This has led to the development of the Extensible Markup Language (XML) as “the universal format for *structured documents* and *data* on the Web” (W3C, 2000). XML is viewed as an important development for the Web, but it needs to address interoperability at the semantic level. The basic approach to define semantics for interoperability is the use of metadata.

There have been several attempts to use metadata to represent the semantics of Web documents. The cataloging information used by traditional printed materials is one example of metadata (Wool, 1998). One of the most promising attempts to define a standard for Web metadata is the Dublin Core Metadata Element Set (Dublin Core Metadata Initiative, 1999) or, in short, the Dublin Core. However, the Dublin Core is not sufficient to represent the semantics of Web documents unless it can interoperate with other metadata standards. This has led to the framework for metadata interoperability, known as the Warwick Framework (Lagoze, 1996). However, the Warwick Framework does not provide a unified data model and syntax, which is crucial to the interoperability between different metadata standards. The Resource Description Framework (RDF), which is considered an implementation of the Warwick Framework, tries to overcome these limitations by defining a uniform data model for representing metadata. It also defines a syntax to represent this data model using XML. The design of RDF is strongly influenced by the KR community. The RDF data model is designed to represent the semantics of Web documents, using metadata, in the form that is easy for a machine to interpret and make use of in an intelligent way (Lassila, 1998).

The Semantic Web is a vision of Tim Berners-Lee, the director of the World-Wide-Web Consortium (W3C). It is “*a Web of data that can be processed directly or indirectly by machines*” (Berners-Lee, 2000d). Berners-Lee defines the architecture of the Semantic Web as shown in the Figure 1 (Berners-Lee, 2000b). Both XML and RDF play major roles in the Semantic Web (Decker et al., 2000b). In addition, they require the ontology layer and logic layer on top of them. Ontology is needed to resolve disparate terminology

in the domain knowledge. The Logic layer is needed to provide formal semantics to enable reasoning services. The Web of trust, using digital signatures is a mechanism to prevent inconsistency in the Semantic Web (Berners-Lee, 2000a).

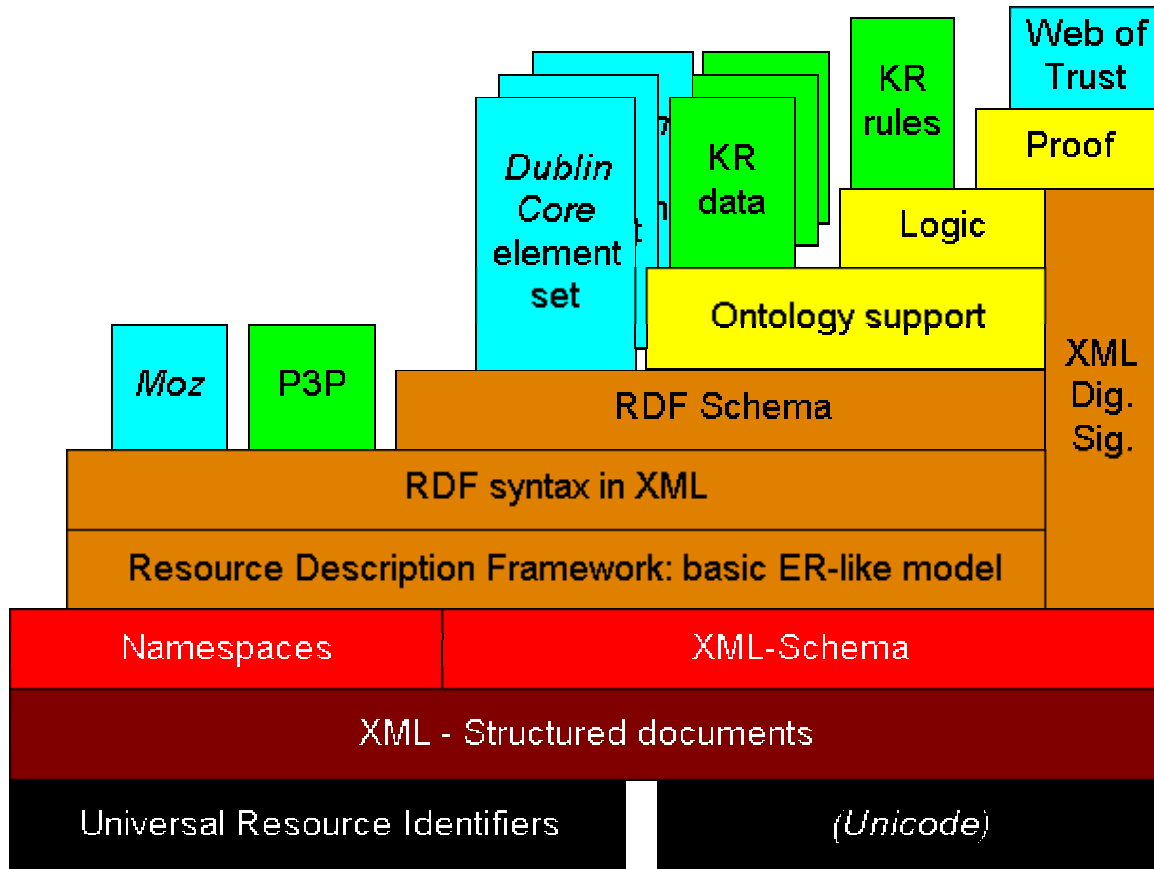


Figure 1: Semantic Web architecture by Berners-Lee

In order for the machine to process the knowledge in the WWW knowledge base, there needs to be an agreement on the representation of knowledge. Ontology is a formal specification of conceptualization (Gruber, 1993). Ontology provides the common understanding of definitions of terms or vocabulary in the domain. Ontology is key to the achievement of the semantic interoperability on the Web. This creates a need for a unified framework for creating and sharing ontology on the web. This has led to the design of knowledge representation languages with sufficient expressive power for creating and sharing ontology on the Web. These research efforts include the Simple HTML Ontology Extension (SHOE) (Heflin & Hendler, 2000b), the Ontology

Interchange Layer (OIL) (Broekstra, Klein, Decker, Fensel, & Horrocks, 2000; Decker et al., 2000a), and the DARPA Agent Markup Language (DAML) (Hendler & McGuinness, 2000). These efforts build on XML and RDF. The formal semantics of the designed ontology languages are usually provided in the form of logic. This is to enable the support for machine-automated processing such as reasoning service. These research efforts fit nicely into the three layers on top of RDF layers in the Berners-Lee's Semantic Web architecture.

1. Interoperability in Information Systems

1.1 Definition

Interoperability, as defined by the Institute of Electrical and Electronics Engineers (IEEE) (Institute of Electrical and Electronics Engineers, 1990), is “the ability of two or more systems or components to exchange information and to use the information that has been exchanged”. A system (or component) can interoperate with any other system (or component) as long as they can understand the information that has been exchanged between each other.

Brodie (Brodie, 1992) gives a functional definition of *Interoperability* in information systems as follows:

“Interoperability: Two components (or objects) X and Y can interoperate (are interoperable) if X can send requests for services (or messages) R to Y based on a mutual understanding of R by X and Y, and Y can return responses S to X based on a mutual understanding of S as (respectively) responses to R by X and Y.” (p.13)

In short, system X can interoperate with system Y if X’s requests can be responded to appropriately by Y.

Under the definitions from Brodie and IEEE, interoperability between information systems is not determined by the physical location. Thus, interoperability between two information systems (or components) can occur within the same machine or between two different machines. It is also not determined by the purpose of interoperation. Interoperability is only determined by the success of information exchange between two information systems.

Interoperability, as used in this paper, refers, consistent with Brodie and the IEEE, to the success of information exchange between systems and is independent of how the data is

stored in each individual system. The paper will focus on the interoperability of systems in a distributed environment.

1.2 Approach to achieve interoperability

1.2.1 Standards

Interoperability is difficult when the number of different systems involved is high. The number of data conversions necessary for a system to communicate with n systems is equal to $n \times (n-1)$. However, if there is an agreed-on set of rules, aka standards, for information exchange, the number of data conversions necessary is reduced to $2n$ or $O(n)$. Thus, standards are critical to the success of interoperability of heterogeneous systems.

Vckovski has defined the requirements for a good standard as follows: expressivity, unambiguity, extensibility and acceptance (Vckovski, 1999).

1.2.2 Layered approach

Agreement is the goal of every standardization process. However, the standardization process is usually costly and time-consuming. This has led to the layered approach, where standards are layered into multiple levels. Spring (Spring, 1996) suggested that, by making the standards modular by layering, the impact of changes in one standard can be isolated from others. The layered approach has been widely used in telecommunications for internetworking, i.e. OSI reference model, TCP/IP reference model.

Unlike telecommunications, currently there is no well-defined separation between data modeling layers (Spring, 1996; Melnik & Decker, 2000). This lack of clear separation has led to the redundant features on different layers. Melnik and Decker (Melnik et al., 2000) have proposed the Information Model Interoperability (IMI) reference model to identify separation between data modeling layers. In the IMI model, data modeling can be divided into three layers: *Syntax*, *Objects* and *Semantics*.

1.2.3 Mediators

Mediator architecture, first introduced by Wiederhold (Wiederhold, 1992), is another approach toward interoperation of heterogeneous information systems. Mediator architectures are also known as “information brokers”, “knowbots” and “software agents” (Kashyap & Sheth, 1994). The mediator is a layer that sits between the user application layer and the data source layer (figure 2). The role of the mediator is to perform necessary transformation and data mapping between different data sources.

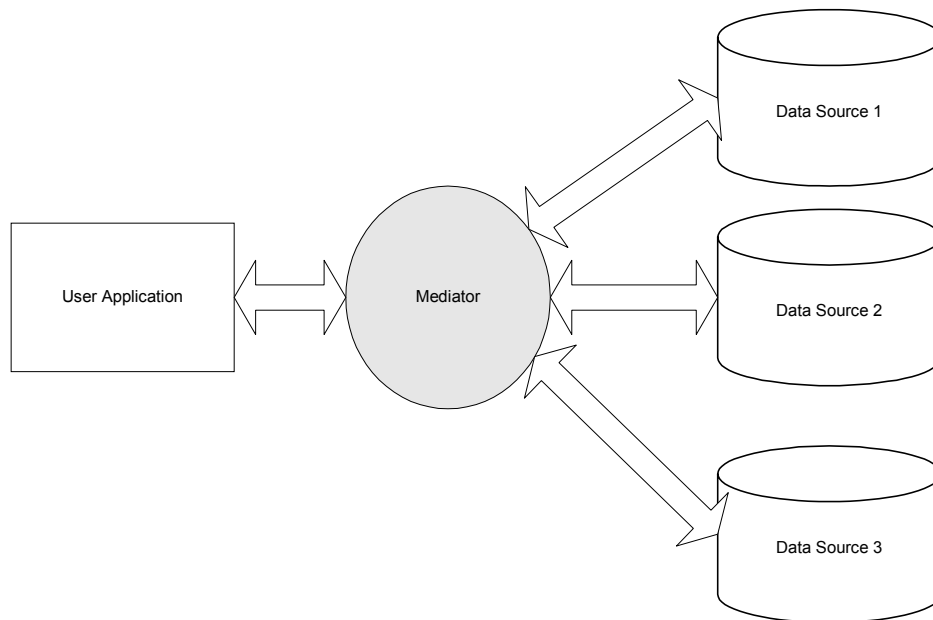


Figure 2: Mediator architecture

Mediator hides the heterogeneity of different data sources from the user applications. Thus it allows user applications to be independent of data sources. In order to perform data mapping, the mediator needs to have knowledge of user applications and data sources. For example, a mediator needs to understand the query formats used by user applications and data sources in order to map user’s queries to the query formats that are required by data sources.

1.2.4 Wrappers

When the number of data sources involved is high, the mediator's knowledge about the data format of each data source can become unmanageable. In order to simplify the task of mediation, wrappers can be placed between the mediator and the data sources (figure 3) by placing a wrapper around each data source. With wrappers, data sources will become homogeneous to the mediator. An example wrapper architecture can be found in (Schwarz & Roth, 1997). Melnik (Melnik, 1999) describes the use of a canonical wrapper in combination with a mediator and layered architecture to facilitate the integration of heterogeneous information systems.

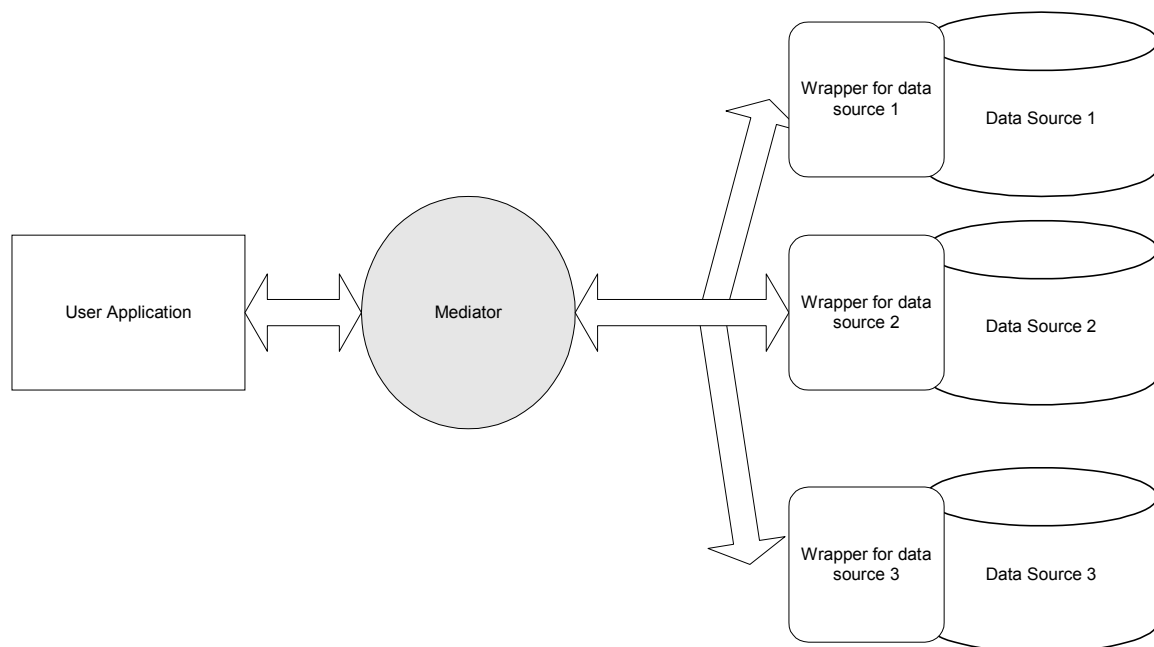


Figure 3: Wrapper architecture

1.3 Toward semantic interoperability in information systems

Sheth (Sheth, 1999) defined the level of heterogeneity in an information system at two levels: *information system heterogeneity* and *information heterogeneity*. Heterogeneous information systems require some solutions for *system interoperability*. Information heterogeneity requires solutions for interoperability at the *syntactic*, *structural* and

semantic level. This can be illustrated in the following figure (adapted from (Sheth, 1999)).

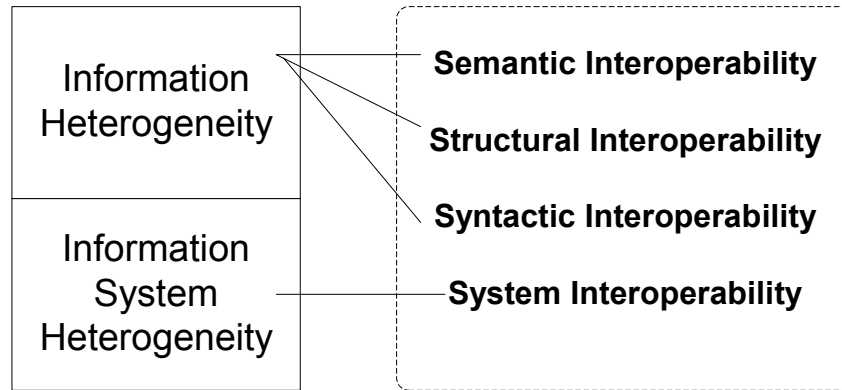


Figure 4: Heterogeneity in information systems

Sheth (Sheth, 1999) defines three generations of interoperable systems. The first generation is from the beginning of computer systems to the mid of 1980s. The focus of this generation is on interoperability among computer systems with different hardware architectures, different operating systems or different DBMSs. The focus of this generation was on system interoperability. The data is usually structured, such as relational data, and the communication is usually within a local area. The second generation began in the mid 1980s and went to the mid 1990s. During this period, the data is more varied and not limited to structured data but also included semistructured data, such as text files, and unstructured data, such as image or audio files. The focus is more on the integration of data, which requires interoperability at syntax and structure levels. This includes schema transformation and data conversion. The third generation begins with the emergence of the World-Wide-Web (WWW). The data becomes more varied and more heterogeneous. The need for more effective retrieval requires the system to understand the semantics underlying user queries or requests. The focus of this generation is moving toward semantic levels of interoperability. There is currently a change of focus on interoperability in information systems from the system, syntax and structure levels to the semantic level (Sheth, 1999).

2. Semantics and its representation

2.1 Semantics

2.1.1 Definition

According to Woods (Woods, 1975), *semantics* means different things to different researchers. The linguists usually think of “semantics” as interpretations of natural language into some kind of *formal representation*. The philosophers usually think of “semantics” as the meaning of these *formal representations*. Artificial Intelligence (AI) may involve both positions.

According to Woods, in order to determine whether two expressions are equivalent, it would be desirable to transform these two expressions to an intermediate form, known as *canonical form*. If the two expressions have the same canonical form, we consider these two expressions equivalent. The transformation function is a canonical function, C (*expression*). Thus, expressions $e1$ and $e2$ are equivalent if

$$C(e1) = C(e2)$$

The definition of the term “*semantics*” used throughout this chapter comes from Woods’ view of semantics as a form of *formal representation* of human expression.

2.1.2 Declarative semantics vs. Procedural semantics

There have been two main approaches in modeling semantics for machine consumption: procedural (or imperative) and declarative semantics. With the procedural approach, the machine is given instructions or procedures. These instructions are usually low-level. The outcome from the program is obtained by instructing the program to execute these instructions or procedures. This approach focuses on building semantics or knowledge of how to obtain the outcome. The *declarative* approach gives meaning by providing the program with facts it knows. This allows the machine to understand the meaning of a new thing by relating it to previous knowledge.

Even though the procedural approach of modeling semantics or knowledge is generally computationally faster, the declarative approach has several advantages, as suggested by McCarthy (McCarthy, 1960). Using the declarative approach, the program can take advantage of previous knowledge. The program can determine the logical consequences of what it is told from what it previously knew. The meaning of declaratives is also less dependent on their order, thus making it easier to modify, i.e. after-thoughts. The declarative approach is also the form that is frequently used in human exchange; where the procedural form is currently used primarily to instruct machines. McCarthy regarded the ability of the program to deduce immediate consequences of anything it is told, from what it already knows, as similar to human *common sense* (McCarthy, 1960). The declarative semantics framework attempts to formalize semantics and reasoning using logic.

2.1.3 Intension vs. Extension

Philosophy distinguishes the expression of meaning into two categories: “*intension*” and “*extension*”. The distinction is sometimes referred as “*sense*” and “*reference*” (Frege, 1892). From a philosophical point of view, the *extension* is the set of all objects in the "actual" world that fall under the concept, whereas the *intension* is the set of objects that fall under the concept in "all possible worlds." In other words, intension could be considered as abstract meaning while extension could be considered as every individual that falls under the abstract meaning. For example, the intension of “human” is the characteristics that make an entity “human”, while the extension of “human” consists of every person in this world. From a computer programming viewpoint, the distinction between intension and extension has been adopted in object-oriented programming model, where *Class* is analogous to “intension” while the set of *Objects* instantiated from the class is analogous to “extension”.

2.2 Knowledge Representation

2.2.1 Definition

Davis et al. (Davis, Schrobe, & Szolovits, 1993) defined Knowledge Representation (KR) in terms of its five basic roles. The five basic roles of KR are fundamental properties that exist in all the invented representations. They are:

- 1) **A KR is a surrogate.** KR is a stand-in for things that exist in the real world. It is an attempt to find representations or surrogates of things. According to Davis et al. (Davis et al., 1993), one consequence of attempting to describe the natural world is that we must inevitably lie, by omitting the complexity of the real world. Thus KR provides surrogates, accepting that surrogates are, by definition, imperfect.
- 2) **A KR is a set of ontological commitments.** Ontological commitments are a set of decisions about what and how to represent the world. These decisions inevitably introduce the focusing/blurring effect. To represent an object in the real world, we have to choose what to focus on and what to ignore.
- 3) **A KR is a fragmentary theory of intelligent reasoning.** Intelligent reasoning has been studied by various fields, including Mathematics, Psychology, Biology, Statistics, and Economics. Each field has a different approach in modeling intelligent reasoning. Examples of these models are mathematical logic (Mathematics), human behavior (Psychology), connectionism (Biology), probability theory (Statistics) and utility theory (Economics). According to Davis et al. (Davis et al., 1993), in specifying a representation, it is also necessary to specify how to reason intelligently from it. “A knowledge representation *is* a theory of intelligent reasoning” (p.13).
- 4) **A KR is a medium for efficient computation.** In order for machines to use a representation, i.e. reason about it, it must be able to make computations. Thus computational efficiency is critical to the effectiveness of a representation.
- 5) **A KR is a medium of human expression.** A representation is a language that humans can use to talk to the machine about the world. It is a medium of

expression and communication from human to the machine. The representation that is expressive and easy to use usually results in effective communication.

Sowa (Sowa, 2000) considers *Logic*, *Ontology* and *Computation* as three major components of KR. Logic provides the formalized language for the representation. Ontology provides the meaning and taxonomy of terms in the domain of interest. Computation is the implementation and manipulation for the computer.

2.2.2 Logic and computation

Logic was first introduced by Aristotle (384-322 B.C.) in the form of syllogism as a simple form of inference. Leibniz (1646-1716) proposed an idea of using Mathematics to formalize logic. In 1879, Frege introduced *quantifiers* to allow the concise expression of facts about objects without enumerating them. First-order logic (FOL) or first-order predicate calculus (FOPC) has been the most fundamental foundation for formalisms based on logic. Because of its expressiveness, logic has been widely used as a formalized language to represent knowledge.

FOL has been known as one of the most expressive and well-understood knowledge representation languages. A FOL *sentence* represents a fact, while an FOL *term* represents an object. It provides *logical operators (connectives)* for forming complex sentences. It also provides *quantifiers* to allow for the concise expression of facts about objects without enumerating them. Facts about objects are expressed in terms of *predicates*. An object can be referred to in term of its relation to other objects using *functions*. Symbols in FOL can be *variable* or *constant*. The syntax rules of FOL are shown in Table 1 (Russell & Norvig, 1995a). Genesereth & Nilsson (Genesereth & Nilsson, 1987) and Russell & Norvig (Russell et al., 1995a) provide good introduction and explanation of the syntax and semantics of FOL.

<i>Sentence</i> →	<i>AtomicSentence</i> <i>Sentence</i> <i>Connective</i> <i>Sentence</i> <i>Quantifier</i> <i>Variable</i> , ... <i>Sentence</i> ¬ <i>Sentence</i> (<i>Sentence</i>)
<i>AtomicSentence</i> →	<i>Predicate</i> (<i>Term</i> , ...) <i>Term</i> = <i>Term</i>
<i>Term</i> →	<i>Function</i> (<i>Term</i> , ...) <i>Constant</i> <i>Variable</i>
<i>Connective</i> →	⇒ ∧ ∨ ⇔
<i>Quantifier</i> →	∀ ∃
<i>Constant</i> →	<i>A</i> <i>X</i> ₁ <i>John</i> ...
<i>Variable</i> →	<i>A</i> <i>x</i> <i>s</i> ...
<i>Predicate</i> →	<i>Before</i> <i>HasColor</i> <i>Raining</i> ...
<i>Function</i> →	<i>Mother</i> <i>LeftLegOf</i> ...

Table 1: Syntax rules of first-order logic in Backus-Naur Form (BNF)

For example, from the following two FOL sentences:

(1) $On (BookOf(John), BookOf(Mary)) \vee On (BookOf(Mary), BookOf(John))$

(2) $\forall x \forall y On(x,y) \Rightarrow Above(x,y)$

The first sentence states that either the book of John is on book of Mary or the book of Mary is on book of John, where *On* is a predicate, *BookOf* is a function, *John* and *Mary* are constants and \vee is the disjunction connective. The second sentence states that one thing that is on another thing implies one is above the other, where \forall is universal quantifier, *x* and *y* are variables, *On* and *Above* are predicates.

Logic provides an ability to deduce new logical sentences from existing sentences using logical computation. This capability is referred to as logical inference or logical reasoning. A generalized pattern of inference is known as *inference rule* or *inference procedure*. Given an inference rule, one can derive a *conclusion* if the *condition* is met.

Modus Ponens (MP) is an example of inference rule. Using Modus ponens, if A implies B ($A \Rightarrow B$) and A is known to be true, one could draw the conclusion that B is true. From the second sentence in the previous example, if '*On(BookOf(John),BookOf(Mary))*' is known to be true, one can infer '*Above(BookOf(John), BookOf(Mary))*'. Other inference rules include Modus Tolens (MT), And Elimination (AE), And Introduction (AI), Universal Instantiation (UI), Existential Instantiation (EI), etc. [see (Genesereth et al., 1987) for details].

Evaluation of an inference procedure is given in term of its *soundness* and its *completeness*. An inference procedure is *sound* if and only if every sentence derived from the inference procedure is logically implied from the knowledge base. An inference procedure is *complete* if and only if all the sentences that could possibly be implied can be derived from the inference procedure. This means that the *complete* inference procedure is not only able to generate new logical sentences that make sense, but it must also be able to discover every logical sentence that could possibly be implied. The completeness of inference procedure is harder to achieve than soundness. Although all of the inference rules mentioned in the previous paragraph are *sound*, none of them are *complete*. Gödel, in his completeness theorem (1930-1931), showed that there exists a *complete* inference procedure in FOL. However, the procedure itself was not discovered until 1965 when the *resolution* algorithm was introduced (Robinson, 1965). The resolution algorithm proves a statement by showing that the negation of the statement produces a contradiction with the statements that are known to be true in the knowledge base [see (Genesereth et al., 1987) and (Russell et al., 1995a) for the explanation of resolution algorithm and examples].

For a given KR system, the task of determining whether a statement can be inferred from the given statements in the knowledge base can be computationally trivial or completely unsolvable. This computational ability is different from one knowledge representation language to another. This difference basically depends on the level of *expressiveness* of the language. It is harder to reason correctly with a representation language when the degree of expressiveness of the language is high. This is known as a fundamental tradeoff

between expressiveness and computational tractability of a knowledge representation language (Levesque & Brachman, 1985). It is one of the most fundamental issues in the design and evaluation of a knowledge representation language.

Due to the high degree of expressiveness in FOL, reasoning in FOL is known to be an undecidable and intractable problem. Levesque & Brachman claimed that reducing the expressiveness of FOL to frame description form could lead to a better computability (Levesque et al., 1985). As a result, Brachman & Levesque (Brachman & Levesque, 1984) introduced a logic which aims to achieve computability by limiting the form of expression of FOL to frame description form. This has become a startup for a new branch of FOL that focuses on describing things and reasoning by determining the *subsumption* relationship. The logic is known as *description logic*.

2.2.3 Ontology

In philosophy, ontology is the study of the nature and relations of being (Merriam-Webster Collegiate® Dictionary, 2001). The term is used by KR community as a way of describing and representing things. The term is often associated with *formal ontology* or *mathematical ontology*, which is a symbolic description of things in a domain. Ontology in KR is a somewhat vague term, with different definitions given by various researchers. One of the most referenced definitions of Ontology in KR is “an explicit specification of a conceptualization” (Gruber, 1993). Gruber considers ontology as a representation of the knowledge of a domain, where a set of objects and their relationships are described by a representational vocabulary. Neches et al. (Neches et al., 1991) shares a similar viewpoint that ontology defines basic terms and relations using a vocabulary of a topic area as well as rules for extending the vocabulary. Ontology is used in KR to facilitate knowledge sharing and reuse. The effectiveness of the sharing and reuse depends on *ontological commitment*, which is an agreement to use the vocabulary in a coherent and consistent manner. Ontology is often related to taxonomy. According to Swartout et al. (Swartout, Patil, Knight, & Russ, 1996), Ontology is “a hierarchically structured set of terms for describing a domain that can be used as a skeletal foundation for a knowledge base”. Sowa (Sowa, 2000) provides a history and review of ontology work in KR.

2.2.4 Approaches in representing knowledge

A. Frame-based systems

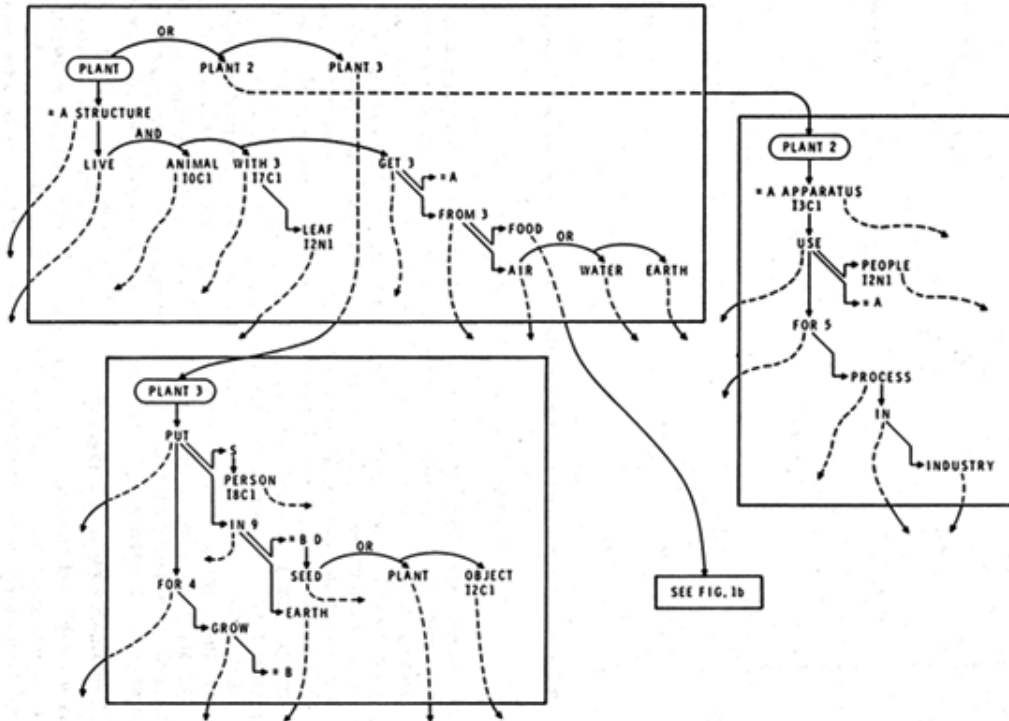
One of the most influential knowledge representation schemes is *frames*. The notion of frames was first introduced by Minsky (Minsky, 1975). Frames were among the first *structured* knowledge representation approaches. A frame consists of slots (or attributes). Attached to each slot can be descriptions or procedures. The value attached to each slot can be filled by default or can have a value restriction. Collections of frames are organized and interconnected in frame systems.

Frames have been criticized for being too flexible and for lack of formalism. The formalism of frames has later been defined using *description logic*, a branch of FOL that combines the structured representation scheme of frame with the formalism of logic.

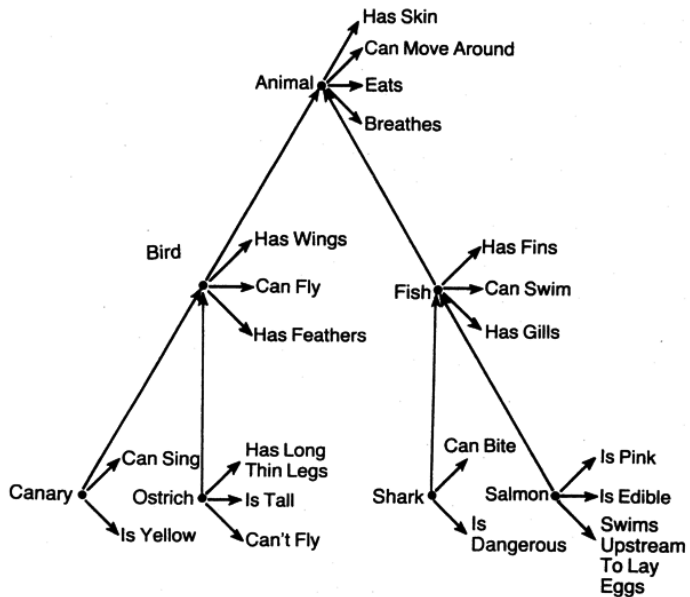
B. Semantic Networks

The semantic network, or semantic net, was first introduced in 1966 by Quillian (Quillian, 1967). Semantic networks were used to represent word concepts in human memory. Semantic nets use a structured representation approach similar to frames. Semantic network can not only represent facts, but also associations between them. In essence, semantic networks contains links between facts. Examples of semantic networks are shown in Figure 5.

- PLANT. 1. Living structure which is not an animal, frequently with leaves, getting its food from air, water, earth.
 2. Apparatus used for any process in industry.
 3. Put (seed, plant, etc.) in earth for growth.



a) a semantic network representing three meanings of “Plant” (Quillian, 1967)



b) a semantic network representing a part of a simple animal hierarchy (Collins & Quillian, 1970)

Figure 5: Example of semantic networks

Woods (Woods, 1975) argued that, “there is currently no “theory” of semantic networks”. Woods suggested that people look at nodes and links in the semantic network without determining whether the meanings upon them are “abstract” (intension) level or “instance” (extension) level. The distinction between intension and extension would help in avoiding unnecessary disagreement on the semantics of the network. Woods also made a distinction between structural links and assertional links, which depend on the sense of meaning being represented (intension or extension). Woods pointed out some limitations of semantic networks, including representation of relative clauses and quantification information. The “relative clause” problem relates to how to represent a reference to an entity that has already been referred to in the network. The quantification issue deals with the need to express quantifier information. The original semantic network is not expressive enough to represent these kinds of information.

Brachman (Brachman, 1979) attempted to clarify the problem of not having uniform semantics for semantic networks, the issue that had been raised by Woods. Brachman suggested that the problem arises because various research efforts in semantic networks are based on different levels of primitives. According to Brachman, there are four different levels of semantic network primitives that previous semantic net research efforts are based on. These levels can be described as follows:

Implementational: Some programming-oriented research work views semantic nets at the level of processing units, i.e. data structures, pointers.

Logical: Most foundation research work in semantic nets treats logic and predicates (predicate calculus) as primitives.

Conceptual: Some research work deals with semantic nets at conceptual level, i.e. semantics and concept of words. This level is independent of language, taking the “thought influences language” viewpoint.

Linguistic: This view, in contrast to the conceptual view, uses the semantic net that is language-specific.

Brachman also suggests a new level, “*Epistemological*”, lying between the logical and conceptual levels. This level gives more a conceptual view to the logical level by adding the notions of inheritance, classification, etc. It also gives more formalism to the conceptual level by adding more structure to it.

Brachman designed a language, named KL-ONE, to reflect the idea of the epistemological level. KL-ONE proposes a formalism that operates at the epistemological level. One of the design goals of the KL-ONE language was to make the semantics of the language clear and well understood.

C. KL-ONE

KL-ONE was originally known as the Structured Inheritance Networks (SI-Nets). It was first introduced in 1977 in Brachman’s Ph.D. dissertation. KL-ONE represented knowledge at the “epistemological level”, where the network is well structured and the type of each node is clearly defined. KL-ONE is not only a knowledge representation language; it also provides a utility for creation and query of the knowledge base. Thus KL-ONE can be considered as a knowledge representation “system”. Brachman et al. (Brachman et al., 1985) provides an overview of the KL-ONE system and its underlying framework. The description of KL-ONE that follows is based on it.

KL-ONE separates *assertion* expression from *description* expression. The *assertion* expression is a mechanism to make statement about things, such as the statement “I have a pen”. The *description* expression deals with the description or definition of “object”, such as the meaning of “pen”. Assertion about things can be independent of the description of things. For example, asserting, “I drop a pen” or “I use a pen” does not change the description of “pen”. The clear separation between assertional and descriptonal component is one of the unique features of the KL-ONE language. The KL-ONE language focuses on the expression of *description*.

KL-ONE is an “object-centered” language, where “object” in KL-ONE can be one of the following three types: ***Concept***, ***Role*** and ***Individual***.

Concept. In KL-ONE, a concept can be *primitive* or *defined*. A concept is a *defined* concept, if it can be described necessarily and sufficiently in term of previously known concepts, otherwise it is a *primitive* concept. For example, if “bicycle” can be necessarily and sufficiently described in terms of the concept “vehicle” with 2 wheels, i.e. $\text{bicycle} \Leftrightarrow \text{2 wheel vehicle}$, then “bicycle” is a defined concept. In contrast, the concept of “taxi” can be necessarily described in terms of the concept “vehicle” with 4 wheels, i.e. $\text{Taxi} \Rightarrow \text{4 wheel vehicle}$, but not vice versa (4 wheel vehicle is not necessarily a taxi). In this case, “taxi” is a *primitive* concept.

Role. A role is property of concept. Role can be represented as attribute-value pairs. The value of each role attribute can be expressed in terms of relationships to known concepts and individuals. The constraints of these values can be given in terms of *Value Restrictions* (V/R). For example, the number of wheels of vehicle must be value-restricted to a number. The role’s value restriction is similar to data type constraint of variable in programming languages.

Individual. An individual or an individual concept is similar to concept but can only be used to describe at most one individual. Individual can be considered as representation of instance of concept. For example, “John’s bicycle”, which refers to a specific instance of the “bicycle” concept, is an individual concept.

Figure 6 shows an example of the primitive concept of an e-mail message (MESSAGE) in KL-ONE. The diagram reads “A MESSAGE is, among other things, a THING with at least one Sender, all of which are PERSONs, at least one Recipient, all of which are PERSONs, a Body, which is a TEXT, a SendDate, which is a DATE, and a ReceivedDate, which is a DATE.” (Brachman et al., 1985)

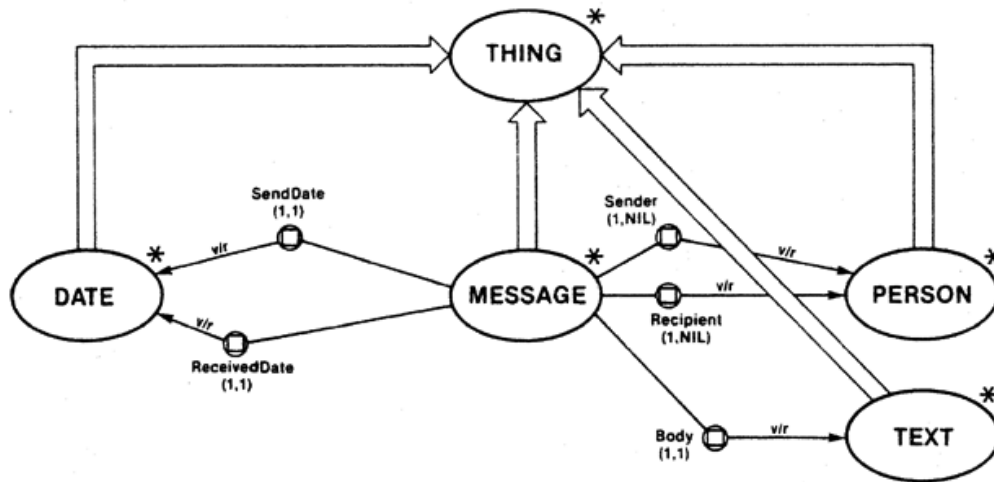


Figure 6: Example of a KL-ONE concept

A KL-ONE knowledge base can be considered as a type of semantic network with hierarchical organization representing inheritance relationships between concepts. The hierarchical organization is performed by determining *subsumption* relationships between concepts. From the definition, Concept A subsumes Concept B if every individual of Concept B can be described by Concept A. Although representing inheritance relationships is possible in both traditional semantic networks and KL-ONE networks, KL-ONE has a major advantage of having a formal language, while a semantic network usually deals only with graphic representation. Thus, the inference mechanism in KL-ONE is more natural and simpler than in the traditional semantic network.

KL-ONE has contributed to the field of Knowledge Representation in many aspects. These include the “epistemological” primitives, which provide clearer semantics than the traditional semantic network. It also introduced the idea of clear separation between *assertions* and *descriptions* of objects, These contributions have set out a new research framework to overcome some of the limitations in the traditional semantic network representation. KL-ONE has been one of the most influential knowledge representation systems in AI. A number of Knowledge Representation research efforts have been developed based on the KL-ONE framework. These KL-ONE successors are known as

the KL-ONE family. A summary of the main features and themes of KL-ONE and its successors can be found in (Woods & Schmolze, 1992).

D. Description Logic

The ideal characteristics of computational logic include expressiveness, decidable and efficient reasoning and sound and complete inference procedures. However, it has long been known that there is a fundamental tradeoff between expressiveness and the computability of reasoning procedures (Levesque et al., 1985). FOL has a high degree of expressiveness, which makes inferences undecidable and inefficient. Description Logic (DL)¹ focuses on computability while maintaining a considerable degree of expressiveness. The semantics of DL is based on the structured representation of KL-ONE, which is based on frames. Description logic can be considered as a unifying formalism for structured representation, such as frames, or can be considered as a *structured* fragment of FOL.

Description logic theory is divided in two parts: TBox and ABox. The “T” in TBox represents “terminological” and “A” in ABox represents “assertional”. TBox deals with the definitions of concepts, while ABox deals with assertions over facts. The clear separation between definition and assertion had been introduced in KL-ONE and was later emphasized and formalized into two separate boxes in KRYPTON (Brachman, Fikes, & Levesque, 1983). TBox allows the establishment of taxonomies of structured terms. Expression in TBox is equivalent to the level of noun phrases in natural language. ABox allows the establishment of descriptive facts about the domains of interest. Expression in ABox is equivalent to the level of sentences in natural language. For example, an expression of “a person with at least 3 children” would go to TBox while an expression of “Every person with at least 3 children owns a car” would go to ABox.

In order to achieve high computability, Brachman & Levesque (Brachman et al., 1984) introduced the reduced version of FOL, which limits the form of expression to frame description form. The logic is known as \mathcal{FL} . The reasoning service for \mathcal{FL} , is provided

¹ Also known as Terminological logic or Concept language

by determining subsumption relationships between concepts. For example, *Student* concept is subsumed by *Person* concept if every member of *Student* concept is also a member of *Person* concept. The computation of this reasoning task in \mathcal{FL} is decidable in polynomial time (Brachman et al., 1984). The syntax rules of \mathcal{FL} are shown in Table A1 on page 90.

Schmidt-Schauß & Smolka (Schmidt-Schauß & Smolka, 1991) introduced the \mathcal{AL} language, which improves the expressiveness of the \mathcal{FL} while maintaining its computability feature. \mathcal{ALC} , a version of the \mathcal{AL} language, is one of the simplest propositional DL. The syntax rules of \mathcal{ALC} are shown in Table A2 on page 91. The reasoning services of the \mathcal{AL} language are provided by determining subsumption relationships between concepts as well as by satisfiability (or consistency) checking. By satisfiability checking, a concept of “*Student* and *not Person*” is not satisfiable if there is no member of *Student* concept that is not a member of *Person* concept. The computation of these reasoning procedures is based on the Tableaux calculus, whose computability is also possible in polynomial time. The detailed description of the description logic family as well as its reasoning service is provided in [Appendix A](#).

Description logics have been deployed in various kinds of applications. One of the most popular KR systems based on DL is CLASSIC (Borgida, Brachman, McGuinness, & Resnick, 1989). CLASSIC has been used by various projects at AT&T. One of the most successful implemented systems is PROSE (Product Offerings Expertise). PROSE, combined with QUESTAR, another CLASSIC based system, was used for over four billion dollars worth of AT&T and Lucent products (Wright et al., 1993). Recently, the implementation of PROSE has been demonstrated in a simpler kind of application, a home entertainment configurator. The purpose of the demonstration was to show how CLASSIC could be used in consumer industry. More details can be found in (McGuinness, Resnick, & Isbell, 1995; McGuinness & Wright, 1998). Apelon (<http://www.apelon.com/>) demonstrates another commercial use of DL in managing terminologies for clinical information systems. According to Apelon, description logic

provides automatic classification features that simplify the development and maintenance of terminologies.

E. Nonmonotonic Reasoning: KR under uncertainty

There are always the cases where the expression of exceptions and uncertainty are necessary. However, classical logic is not expressive enough to describe exceptions or uncertain things. Classical logic does not allow one to draw a new conclusion that conflicts with existing axioms in the knowledge base. This makes classical logic *monotonic*. The monotonic feature is not practical in the real world where few things are certain. As exceptions or changes are discovered, one would want to retract previously drawn conclusions from the knowledge base. In a classic example, if we know that all birds can fly ($\forall x. Bird(x) \Rightarrow Fly(x).$) and we know that Tweety is a bird ($Bird(Tweety)$). FOL would draw the conclusion that Tweety can fly ($Fly(Tweety)$). However, if it is later discovered that Tweety is an ostrich, since an ostrich can not fly we would have to conclude that Tweety can not fly. The FOL would not allow one to draw the conclusion that Tweety can not fly as it would conflict with the first conclusion in the KB. Nonmonotonic logic and reasoning is introduced to overcome this limitation. Nonmonotonic logic allows one to retract the previous conclusion if it is later known to be untrue.

One of the well-known uses of nonmonotonicity in KR is the closed-world assumption (CWA), introduced by Reiter (Reiter, 1978). The closed-world assumption, in short, states that anything that is unknown to the knowledge base is assumed to be untrue. The closed-world assumption eliminates the needs to explicitly make statements about things that are believed to be false. This leads to nonmonotonicity in the knowledge base because once a new fact is obtained, the previous belief that it is false is no longer true. CWA is widely used in database application and logic programming.

There have been three major approaches to formalizing nonmonotonic reasoning using logic. These approaches are **Circumscription**, **Default logic** and **Modal logic**. **Circumscription** (McCarthy, 1980) is the only approach among the three that does not

extend classical logic. This has an advantage that it does not introduce incompatibility with classical logic. One of the motivations of circumscription is that humans usually jump to the conclusion that a thing usually works unless “something” prevents it (abnormality). This reasoning process minimizes the need for a large number of axioms that needed to be stated about exceptions. Circumscription is a formalism for this human informal reasoning process. It gives the minimal model of what we can infer from the given facts. Thus when a new fact is introduced, the new circumscription would invalidate the previous minimal model. This characteristic is a form of nonmonotonic reasoning. **Default logic** (Reiter, 1980) provides an inference rule for nonmonotonic reasoning. It could be considered as a variation of modus ponens. Default logic makes default assumptions when dealing with incomplete knowledge. These default assumptions can later be modified by subsequent evidence. **Modal logic** (McDermott & Doyle, 1980; Moore, 1985) extends classical logic by introducing a new symbol to express uncertainty. An overview of these nonmonotonic logic approaches is provided in [Appendix B](#).

Nonmonotonic logic and reasoning has been criticized on the basis of computability. The nonmonotonic reasoning systems need to deal with consistency checking. Determining consistency is known as an undecidable problem while determining inconsistency is decidable but intractable (Morgenstern, 1999). Thus implementing an effective nonmonotonic reasoning system is known to be a difficult task. Although there is an extensive literature on nonmonotonic logic and reasoning, applications of nonmonotonic reasoning systems in the real world are still very small in number. Logic programming is one of the efficient implementations of nonmonotonic reasoning. An overview of nonmonotonic techniques in Logic programming can be found in (Minker, 1993). Ginsberg (Ginsberg, 1987) provides a good review of Nonmonotonic logic and reasoning.

The approach in nonmonotonic reasoning mentioned in this section has taken the symbolic or qualitative approach. There are other research areas that use a numeric or quantitative approach to deal with nonmonotonicity. One of the areas is Probabilistic

Reasoning. Much of the research in this area uses Bayesian Networks to represent knowledge under uncertainty. The review of the numeric approach is beyond the scope of this paper. Shafir (Shafir, 1999) provides a good overview of probabilistic reasoning. See (Russell & Norvig, 1995b) for explanations and examples of representing and calculating probabilistic reasoning using Bayesian Networks.

3. Interoperability on the Web

The Web was designed with the assumption that the data formats for the Web would proliferate (Berners-Lee, 1998c). The Hypertext Transfer Protocol (HTTP) (Berners-Lee, Fielding, & Frystyk, 1996), the Web communication protocol, was designed to support various kinds of data formats via the Multipurpose Internet Mail Extensions (MIME) types. Despite the generalized design, the Hypertext Markup Language (HTML) has become the de facto standard for Web documents.

This chapter reviews the history and current state of document and data interchange formats on the Web. This is an important aspect of interoperability on the Web.

3.1 Markup Languages

The idea of a markup “language” was introduced in 1967 by William Tunncliffe at a Canadian Government Printing Office meeting (SGML Users' Group, 1990). The language was called ‘generic coding’ to distinguish it from ‘specific coding’ which used control characters or a set of operations (procedural instructions) to instruct software on how to display documents. Generic coding introduced the idea of a declarative markup language. Rather than defining a set of operations, it used descriptive tags to instruct the display software how to format the document. For example, to format a heading of a document, a descriptive “HEAD” tag may be inserted around the heading text.

In 1969, Charles Goldfarb, together with Edward Mosher and Raymond Lorie, at IBM developed the Generalized Markup Language (GML) based on Tunncliffe’s generic coding. GML¹ introduced the idea of allowing users to design document structures using a formally-defined “document type definition”. GML played an important role in document publishing projects at IBM through the 1970s.

Further research on GML led, in 1978, to the initiation of a project, supported by the American National Standards Institute (ANSI), focused on the design of a text

¹ The name represents the initials of its three inventors.

description language standard. The language was later named the Standard Generalized Markup Language (SGML). The first working draft of the SGML standard was published in 1980. SGML was accepted by the International Organization for Standardization (ISO) as an ISO standard (ISO 8879) in 1986 (International Organization for Standardization, 1986). SGML is an international standard for the device-independent, system-independent representation of texts in electronic form.

Three major characteristics of SGML are descriptive markup, document type definition, data representation independence (Sperberg-McQueen & Burnard, 2001).

SGML does not specify procedural instructions, such as indenting text, stepping to the next line, etc. These procedural instructions are usually platform-dependent. Giving explicit procedural instructions would lead to a language that is not device-independent and not interoperable between systems. SGML only describes the logical structure of elements of document. It leaves the interpretation of document formatting to the processing software. Thus, SGML separates the content and structure of the document from its presentation.

The document type concept, which is originated with GML, allows document creators to create their own document type. The logical structure of a *document type*, i.e. book, recipe or brochure, can be defined in a formal definition called a Document Type Definition (DTD). Once a DTD has been defined, a document of this type can be instantiated from it. This document must follow the structural rules specified in its DTD. The ability to create DTDs makes the SGML markup language extensible.

SGML documents are designed to be transportable from one hardware/ software environment to another without loss of information. SGML provides encoding schemes that allow different character sets to be displayed correctly in different environments. This machine-independent encoding allows characters to be transformed or substituted to appropriate characters from system to system.

3.2 Hypertext Markup Language (HTML)

In 1989, Tim Berners-Lee and Robert Caillau at the European Organization for Nuclear Research (CERN - Conseil Européen pour la Recherche Nucléaire) collaborated on developing a universal linked information system for the CERN community. In October of 1990, the system was named the "World-Wide Web". One of the requirements of this system was a formatting language for the hypertext documents. Given the use of SGML by the CERN community, Berners-Lee developed an SGML DTD called the Hypertext Markup Language (HTML). In 1991, he put the code and specifications for HTML on the Internet.

As the World Wide Web became known among the Internet community, HTML was further extended from its original specification. In June 1993, Berners-Lee released an IETF draft version of the Hypertext Markup Language (Berners-Lee & Connolly, 1993). However, the implementation of HTML by many of the WWW browsers still extended beyond what had been defined in the draft. The first successful attempt at interoperability for HTML was HTML 2.0 (IETF RFC1866) (Berners-Lee & Connolly, 1995). HTML 2.0 attempted to capture the state of HTML as implemented in the WWW browsers in June 1994. HTML 2.0 was the de facto HTML standard until its replacement by HTML 3.2 (Raggett, 1997) in January 1997 and HTML 4.0 (Raggett, Hors, & Jacobs, 1998) in December 1997.

HTML consists of four major components: *tag*, *element*, *attribute* and *link*. The first three components are markup features of SGML. Links were added to enable hypertext. HTML required an addressing scheme for linking WWW resources. Berners-Lee designed a Uniform Resource Identifier (URI) addressing scheme¹ (Berners-Lee, Fielding, & Masinter, 1998) that allowed HTML documents to be linked to other resources on the Web. By combining the descriptive markup capability of SGML with the linking capability, HTML has served as a common format for publishing WWW documents.

¹ URI could be in the form that is location-dependent, i.e. Uniform Resource Locator (URL) or location-independent, i.e. Uniform Resource Name (URN).

Despite its success and popularity, HTML was criticized by the SGML community. HTML, as implemented, lacked *extensibility, structure and validation* (Bosak, 1997).

HTML is a simple set of markup tags. HTML does not allow document authors to extend the syntax. Extending HTML would lead to incompatibility. This makes the extensibility of HTML low compared to its meta-language, SGML. SGML does not define particular tags but the rules by which an author can create a unique DTD. SGML software would access the defined DTD and process documents that belong to the given document type.

HTML also lacks a clear separation between document structure and presentation. While some HTML tags are used to identify document structures, such as `<p>` for paragraph, `` for ordered list, other tags specify how the text should be displayed, such as `` for bold text, `` for properties of displayed font. This makes the document structure underlying in an HTML document obscure. The lack of structure is problematic when there is a need to represent structured or semistructured data on the Web (Abiteboul, Buneman, & Suci, 2000).

The last major limitation of HTML, based primarily on its loosely structure from, is the absence of any validation function. The HTML specification does not specify mechanisms for HTML applications to check the validity of HTML documents. While there are HTML validation services, such as W3C HTML validation service (<http://validator.w3.org>), the vast majority of HTML documents available on the Web are not valid HTML documents. One of the reasons is that most of the WWW browsers will display invalid HTML documents, even ones with incorrect syntax. In SGML, validity is important. SGML requires that SGML documents conform to the syntax, i.e., be well-formed, and the grammar, i.e. be valid by conforming to a DTD.

While SGML provides a broad solution, it has one major drawback -- it is too complex. Adding SGML processing software to existing WWW browser would require massive changes in browsers. This makes it politically and economically difficult to implement. This led the W3C to design a simplified version markup language of SGML to replace

HTML. As we will point out, the final form of XML *and* its companion standards may actually be more complex than SGML.

3.3 Extensible Markup Language (XML)

The eXtensible Markup Language (XML) was developed by the XML Working Group, originally known as the SGML Editorial Review Board, formed under the supervision of World Wide Web Consortium (W3C) in 1996. It was chaired by Jon Bosak of Sun Microsystems with the participation of an XML Special Interest Group, previously known as the SGML Working Group, which was also organized by the W3C. The XML specification was finally approved as W3C recommendation in February 1998.

W3C specified the original design goals for XML as follows (Bray, Paoli, Sperberg-McQueen, & Maler, 2000):

1. *XML shall be straightforwardly usable over the Internet.*
2. *XML shall support a wide variety of applications.*
3. *XML shall be compatible with SGML.*
4. *It shall be easy to write programs which process XML documents.*
5. *The number of optional features in XML is to be kept to the absolute minimum, ideally zero.*
6. *XML documents should be human-legible and reasonably clear.*
7. *The XML design should be prepared quickly.*
8. *The design of XML shall be formal and concise.*
9. *XML documents shall be easy to create.*
10. *Terseness in XML markup is of minimal importance.*

In some ways, XML is a subset of SGML. One of its goals is “to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML” (Bray et al., 2000). XML is designed to interoperate with SGML. An XML document is also a conforming SGML document. Detailed comparison of XML and SGML can be found in (Clark, 1997).

XML is currently supported in some ways by the major WWW browsers. XML documents can be parsed and displayed in the WWW browsers. This allows XML documents to be usable over the Internet. The APIs to process XML documents have become publicly available. These includes Document Object Model (DOM) (W3C DOM Working Group, 2001) and Simple API for XML (SAX) (Megginson, 2000). Using DOM, each component in XML document can be manipulated as an object. SAX is syntax-driven API to process XML. With the availability of public software modules to process XML documents, XML is being more frequently used in various Internet applications, particularly for business applications.

3.3.1 XML Document

A document is “well-formed” XML if it meets the syntax requirements related to tag formation and element nesting. A well-formed XML document is “valid” if it conforms to the structure specified in a DTD. In other word, a *well-formed* XML document is guaranteed to have the proper *syntax* and a *valid* XML document has a structure consistent with a specific DTD. Unlike SGML, a DTD is not mandatory for an XML document unless validity is of concern. This allows XML to be more flexible. An XML document is more restricted than an HTML document, in that well-formedness must be met¹. Thus, an XML document is syntactically more restricted than an HTML document but more structurally flexible than an SGML document.

While a DTD can define the structure of XML document, the need for better datatype control, along with other needs, led to a replacement for DTD’s called Schema (Berners-Lee, Connolly, & Swick, 1999). XML Schema² (Thompson, Maloney, & Mendelsohn, 2000; Biron & Malhotra, 2000) enhance the functionality of XML DTDs by addressing the following issues: *structured schema*, *data typing* and *conformance* (Malhotra & Maloney, 1999).

¹ An HTML document does not have to be well-formed

² XML Schema became a W3C Recommendation in May 2001

XML Schema extends DTD functionality by focusing on the reuse and interoperability. XML Schema allows the authors to extend (inherit) existing schemas. A document under SGML and XML may implement one and only one DTD. The XML namespace facility (Bray, Hollander, & Layman, 1999) supports integration and interoperability of schemas. Using namespaces, a document author can refer to multiple schemas that provide document models that can be mixed in the document. The use of namespace allows the XML schemas to be deployed on a large scale without collisions between elements from different schemas with the same name.

XML documents are intended for use on the WWW, which is a hypertext environment. This requires XML to have an ability to express linking information among XML documents and other Internet resources. To this end, the W3C has defined a specification for expressing links in an XML document. The language is known as XML Linking Language (XLink)¹ (DeRose, Maler, & Orchard, 2001). The design principles and requirements of XLink can be found in (DeRose, 2001). XLink is designed to be more expressive and more powerful than hypertext linking as defined under HTML. Some additional features of XLink include bidirectional links, multiple-destination links, and out-of-line links. The syntax of XLink is specified using XML. An XML link must use an URI (Berners-Lee et al., 1998) to address a resource. An XML link uses the XML Pointer Language (XPointer) specification (DeRose, Maler, & Daniel, 2001) to identify specific portions in an XML resource as link target.

Like SGML, XML separates document structure from presentation. W3C has been promoting the use of style sheets on the Web (Bos, 2001). By attaching style sheets to structured documents on the Web, authors and readers can influence the presentation of documents without interfering with document structure. The Extensible Stylesheet Language (XSL) (Adler et al., 2000), developed by the W3C XSL Working Group, is a language for expressing stylesheets for XML documents.

¹ XLink became a W3C recommendation in June 2001

The design of XSL was influenced by the Document Style Semantics and Specification Language (DSSSL) (International Organization for Standardization, 1996), DSSSL Online Application Profile (DSSSL-O) (Bosak, 1996) and Cascading Style Sheets (CSS) (Bos, Wium Lie, Lilley, & Jacobs, 1998; Wium Lie & Bos, 1999). DSSSL is an International Standard for style sheet language that is particularly used for formatting of print documents. It was designed as a companion to SGML. DSSSL-O is a profile of DSSSL, adapted for online documentation. XSL semantics is based on DSSSL and the DSSSL-O. Bosak, the editor of the DSSSL-O specification, considered XSL to serve the purposes of DSSSL-O in a form that is easier to use with clean separation between the declarative and programmatic parts of a style sheet (Jon.Bosak@eng.Sun.COM (Jon Bosak), 1997). The W3C experience of deploying CSS over the Web has influenced the design of XSL. Both XSL and CSS can be used to style XML documents, and they are intended to interoperate with each other (Bos, 2001). In CSS, the structure of the XML document cannot be changed; while XSL provides a mechanism for transforming the structure of an XML document. Another major difference between XSL and CSS is that XSL uses XML as its syntax¹.

XSL consists of two parts: XML Transformation language (XSLT) (Clark, 1999), and an XML vocabulary for specifying formatting semantics. XSL specifies the styling of an XML document by using XSLT to describe how the document is transformed into another XML document that uses the formatting vocabulary. XSLT language is a language for transforming XML documents into other XML documents. XSLT is also designed to be used independently of XSL formatting objects as a XML transformation language (Clark, 1999). XSLT makes use of the expression language defined by XPath (Clark & DeRose, 1999) for *addressing* and *matching*. XSLT uses XPath to select parts of an XML document for processing. XSLT also provides facilities for string and number manipulation. XSL uses the same underlying formatting model as CSS to ensure the interoperability of style sheets for the Web documents (Bos, 2001).

¹ All of the XML companion standards, using schema and namespaces, are defined in the form of XML documents. The recursive nature of these definitions makes many of these objects much more difficult for human to read, but greatly decreases the complexity of the parser engines that need to be written.

3.3.2 XML as a Data Interchange Format for the Web

W3C XML Activity Statement regards XML as “the universal format for *structured documents* and *data* on the Web”. Although XML was derived from SGML, which is designed for *document* publishing, XML is also considered a tool for represent *data* of any form. While XML can easily represent structured data, there were several limitations in the XML based on SGML that had to be overcome. Specifically SGML purposefully avoided any form of control, beyond that required to check syntax, on the “content” of a document.

Semistructured data is data without explicit description of the type or structure of the data. Semistructured data is sometimes described as “schemaless” or “self-describing” data (Abiteboul et al., 2000). Semistructured data does not need external schema to describe its data. This is in contrast to structured data such as a database record, where database schemas are available to define the structure of the data. The structure of semistructured data is explicitly annotated along with each data item. As an example of semistructured data, consider:

```
{name: “Alan”, tel: 2157786, email: agb@abc.com}
```

XML documents may be defined as semistructured. The content in a well-formed XML document can be described by the markup tags surrounding it without an external DTD. This makes XML a good model for semistructured data. For example, one way to represent the above example as an XML document is shown as follows:

```
<person>
  <name>Alan</name>
  <tel>2157786</tel>
  <email>agb@abc.com</email>
</person>
```

However, while a database might control what constitutes a “tel” to digits, XML, prior to schema, had no such mechanism.

3.3.3 XML Schema

W3C's XML Schema Working Group has listed the design principles of XML schema language as follows (Malhotra et al., 1999):

1. *more expressive than XML DTDs;*
2. *expressed in XML;*
3. *self-describing;*
4. *usable by a wide variety of applications that employ XML;*
5. *straightforwardly usable on the Internet;*
6. *optimized for interoperability;*
7. *simple enough to implement with modest design and runtime resources;*
8. *coordinated with relevant W3C specs (XML Information Set, Links, Namespaces, Pointers, Style and Syntax, as well as DOM, HTML, and RDF Schema).*

XML Schema enhances the functionality of XML DTD by addressing the following issues: *structured schema, data typing and conformance* (Malhotra et al., 1999).

XML Schema (Thompson et al., 2000; Biron et al., 2000) specifies the XML Schema definition language, which offers facilities for describing the structure and constraining the contents of XML documents. The schema language is represented in XML. The specification also addresses the issue of reuse and interoperation between schemas using the XML namespace facility (Bray et al., 1999).

The XML Schema supports data typing for specifying constraints, such as range, precision, etc, of the data. The XML Schema Specification Part 2 (Biron et al., 2000) has defined primitive data types, which are summarized in Figure 7 (Jelliffe, 2000). XML Schema also provides mechanisms for document authors to specify complex data types, which is similar to the composition of a data structure in programming languages.

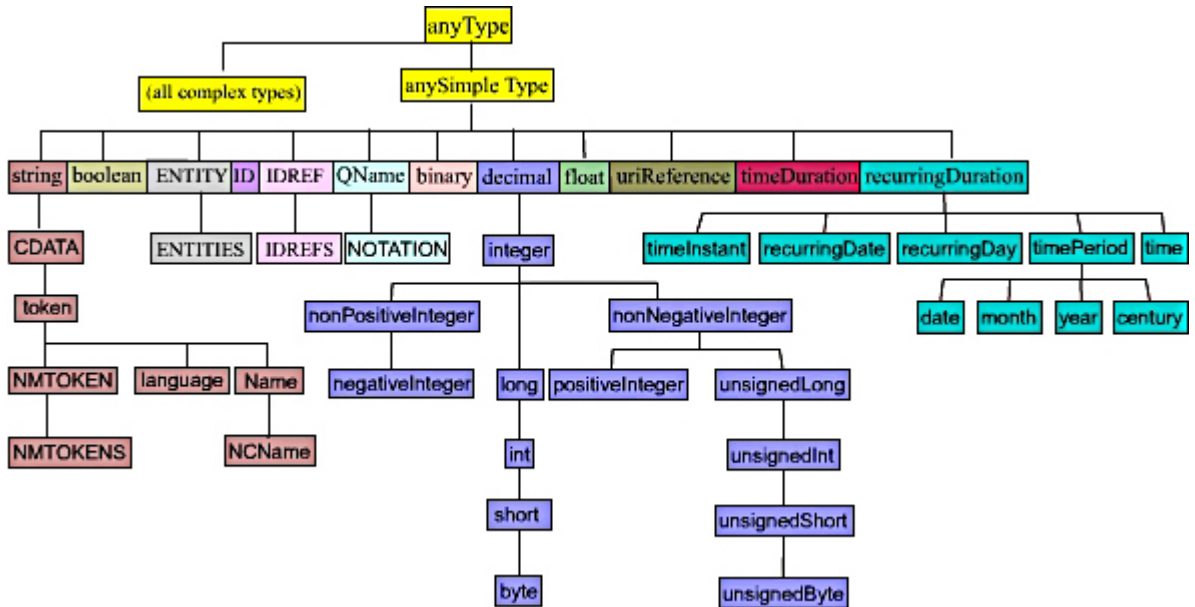


Figure 7: Map of built-in data type in XML Schema

Another requirement of XML Schema is *conformance specification*, which addresses the issue of checking validity of the schema. XML Schema defined three levels of conformance of a schema (Thompson et al., 2000).

- Level 1: abstract data model
- Level 2: schema representation (syntax)
- Level 3: Web interoperability

The first level requires that all the XML Schema processors need to understand the vocabulary and rules, given in the specification, in order to validate XML documents based on a schema. This is the only level that is required for all XML schema processors. It is independent of the syntax that is used to represent the schema. The second level, which is an optional requirement, specifies how the document authors can use XML syntax to describe the schema. Using XML syntax would allow the schema to receive the advantages of XML, such as availability of parsers and expressiveness, etc. The third level, which is also an optional requirement, specifies how a schema could be deployed over the WWW environment. The specification for this level specifies the addressing

scheme for a schema on the Web using Uniform Resource Identifiers (URI) and the MIME type of the schema as “text/xml”.

4. Web Semantics

4.1 Metadata

4.1.1 Definitions

Metadata is “data about data”. It is information about a document, such as author, publication date, etc. The International Federation of Library Associations (IFLA) (International Federation of Library Associations, 2000) defines metadata as follows:

Metadata is data about data. The term refers to any data used to aid the identification, description and location of networked electronic resources.

This definition has limited the scope of metadata usage to electronic resources. Another definition of metadata from Caplan (Caplan, 1995) is:

Metadata really is nothing more than data about data; a catalog record is metadata; so is a TEI header, or any other form of description.

According to Caplan, metadata can be used to describe any resources. Caplan does not limit the scope of metadata to electronic resources. Under this definition, a traditional library catalog card is also one kind of metadata. Wool (Wool, 1998) indicates that this definition is preferable as it suggests that metadata is not something new. It has been used for centuries by librarians and publishers. Metadata can be viewed as a kind of cataloging information. The definition of metadata and scope of its usage are still a debate in the library community (Gradmann, 1998; Milstead & Feldman, 1999).

4.1.2 Dublin Core

In October 1994, at the second International World Wide Web Conference, Stuart Weibel, Senior Research Scientist at the Online Computer Library Center (OCLC), pointed out the need for an agreement on semantics for Internet resources. From a librarian’s point of view, this semantics would be equivalent to creating simple catalogue

cards for the Internet resources that are not domain-specific but can work across disciplines (Chepesiuk, 1999). This information would help people to describe their materials in order to help Internet users find materials they are looking for.

This initiative led to the OCLC/NCSA Metadata Workshop in March 1995, in Dublin, Ohio, which is also known as the first Dublin Core workshop. The workshop included 52 librarians, archivists and scholars. They tried to reach an agreement on a set of simple *metadata elements* that could be used to describe networked digital resources, i.e. resources on the World Wide Web. The scope of the resources that were to be described by these metadata elements, were limited to *document-like objects*, or *DLOs*. DLOs are primarily text resources. By restricting the focus to DLOs, the design of metadata elements would resemble the cataloging information that are used to describe traditional print materials.

The result of the agreement was a set of thirteen metadata elements, known as the *Dublin Core Metadata Element Set* or *Dublin Core*. The original Dublin Core elements were:

1. **Subject:** The topic addressed by the work
2. **Title:** The name of the object
3. **Author:** The person(s) primarily responsible for the intellectual content of the object
4. **Publisher:** The agent or agency responsible for making the object available
5. **OtherAgent:** The person(s), such as editors and transcribers, who have made other significant intellectual contributions to the work
6. **Date:** The date of publication
7. **ObjectType:** The genre of the object, such as novel, poem, or dictionary
8. **Form:** The physical manifestation of the object, such as Postscript file or Windows executable file
9. **Identifier:** String or number used to uniquely identify the object
10. **Relation:** Relationship to other objects

11. **Source:** Objects, either print or electronic, from which this object is derived, if applicable
12. **Language:** Language of the intellectual content
13. **Coverage:** The spatial locations and temporal durations characteristic of the object (Weibel, 1995)

The underlying design principles of the Dublin Core are *intrinsicity*, *extensibility*, *syntax independence*, *optionality*, *repeatability*, and *modifiability* (Weibel, 1995). *Intrinsicity* is an ability to describe the resource from its content; no context of use is needed. *Extensibility* is an ability to add extra elements for domain specific information; the core elements must maintain backward-compatibility when they are updated. Dublin core defines the semantics of elements, but not syntax. This makes it usable in a wide range of applications. All the elements in Dublin Core are *optional*. All the elements in Dublin Core are *repeatable*, e.g. to identify multiple authors of the resources, the “author” field can be repeated. Dublin Core allows sophisticated users to use optional qualifiers (Dublin Core Metadata Initiative, 2000) to specify specific definition of the element (*modifiability*), i.e. “Subject (scheme=LCSH)” indicating that the subject terms are taken from the Library of Congress Subject Headings.

The number of core elements has been kept low in order to make the standard simple and applicable to a wide-range of resources. Dublin Core 1.1 (Dublin Core Metadata Initiative, 1999) has defined two more elements: **contributors** for person(s) who contribute to the content of the resource and **rights** for copyright information. Thus, there are currently 15 core elements in the Dublin Core.

4.1.3 Warwick Framework

A year after the Dublin Workshop, a follow-up workshop was held at the University of Warwick, U.K. The workshop addressed several issues, including the assessment of the one-year experiment with the Dublin Core. Another focus of the workshop was to promote interoperability among different metadata schemes. It sought to address how the Dublin Core could work with other metadata standards. The result of the workshop was an architecture, known as the Warwick Framework (Lagoze, Lynch, & Daniel, 1996).

The motivation for the development of the Warwick Framework was to allow interoperability among the existing metadata schemes. Some metadata schemes are general and not domain-specific, e.g. MARC (<http://www.loc.gov/marc/>), while some are domain-specific, e.g. SAE (<http://www.sae.org/>). Different metadata schemes are also created for different purposes. Some are created for descriptive cataloging purpose. Some are created for legal purposes, i.e. to describe the terms and conditions of use. Some are created for rating the suitability of the content for audiences, e.g. Platform for Internet Content Selection (PICS - <http://www.w3.org/PICS/>). The Dublin Core, which is a general descriptive metadata scheme, was not sufficient to capture all the requirements without incorporating other metadata schemes.

The Warwick Framework was designed to support modularity. For example, a legal organization may want to create a metadata set for describing terms and conditions of use, while librarians might just want to create descriptive cataloging metadata set. The Warwick Framework was designed to support the integration of these pieces. It is also designed to support user selectivity. The architecture allows the users to access to a specific set of metadata in a document, i.e. the parental control software can choose to look at the content rating of the document, while the search engine robots may choose to look at the descriptive cataloging information of the document. The need for selectivity is also necessary when the information that looks like metadata to one program becomes data for another application, e.g. a review of a document can be considered as metadata of the document or a part of the document itself. Under the Warwick Framework, users can choose the metadata elements that are appropriate for their needs.

The Warwick Framework allows different metadata sets in a single document to coexist in the document by separating each metadata set into a *package*. These packages are then grouped together in a *container*. This architecture is illustrated in Figure 8.

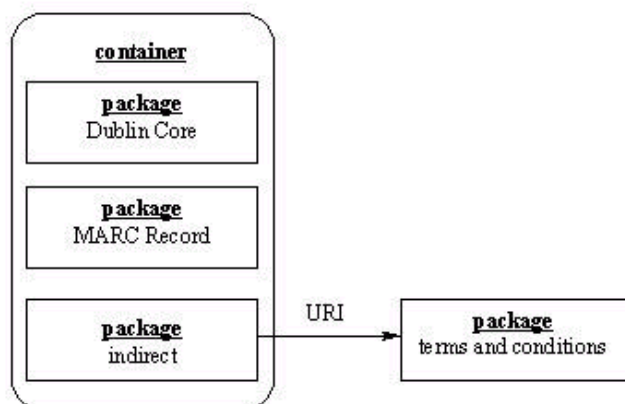


Figure 8: Warwick Framework architecture

Under this architecture, a package can contain records of a metadata set, such as MARC records, Dublin Core records. A package can also be a link to a package in an external document, i.e. via URL. Under the Warwick Framework, a package can also be a container, making the architecture recursive.

Even though the main architecture of the Warwick Framework has been defined, many problems and issues are still open and undefined. These include (Lagoze, 1996):

- **Semantic Overlap:** It is possible that two metadata sets have semantic overlap. The Warwick Framework has not defined how the applications should handle this interaction, i.e. how to interpret two metadata records that are conflicting with each other within a single document.
- **Package Type:** In order for a program to interpret the metadata inside a package correctly, it must understand the type of the package, e.g. Dublin Core Package Type or MARC Package Type. As new metadata standards emerge, the architecture needs to specify how the processing software can update its understanding of these new metadata standards.
- **Syntactic interoperability:** The Warwick Framework was syntax-independent. Although this increases flexibility, there needs to be an agreement on the syntax for the metadata sets to interoperate.

- **Efficiency:** The distributed nature of the Warwick Framework can lead to inefficiency, i.e. slow response time, failed connection, etc.
- **Querying:** The selectivity characteristic of the Warwick Framework requires the ability to query and retrieve packages at various levels. The Warwick Framework does not define the mechanism for this.

Although there are many issues still left open, some of which are difficult to overcome, the Warwick Framework has set out a framework for interoperability among metadata standards. Many ideas of the Warwick Framework have resulted in the design of the W3C Resource Description Framework (RDF).

4.1.4 Resource Description Framework (RDF)

The Resource Description Framework (RDF) is “*a foundation for processing metadata; it provides interoperability between applications that exchange machine-understandable information on the Web*” (Brickley & Guha, 2000). RDF provides a formal data model and syntax for encoding metadata for the purpose of machine processing (Lassila & Swick , 1999).

A. RDF Design Goals

RDF has been designed to provide the following main features (Berners-Lee & Swick , 1999):

- *Interoperability of metadata*
- *Machine understandable semantics for metadata*
- *Better precision in resource discovery than full text searching*
- *Future-proofing applications as schemas evolve*

RDF as a framework for interoperability of metadata

One of the major influences in the design of RDF comes from the digital library community. Research has focused on how different sets of metadata for electronic resources can interoperate (Lagoze et al., 1996). RDF may be viewed as an

implementation of the Warwick Framework (Brickley et al., 2000). RDF has proposed some solutions to the problems that were left unsolved in the Warwick Framework.

The Warwick Framework is syntax-independent. Each metadata set can be represented using its own syntax. Also, in the Warwick Framework, there is no unified data model for all metadata sets. Thus, when a new metadata scheme emerges, the processing software needs information about how to parse the new metadata scheme. The software needs to be updated to support the new metadata standards accordingly. In response to these issues, RDF extends the Warwick Framework by defining a unified data model and syntax that all metadata standards can share. Another problem in the Warwick Framework is *semantic overlap* -- two different metadata sets can use the same term with different meanings or different terms with the same meaning. This problem has been left open in the [Warwick Framework](#). RDF makes use of XML namespace (Bray et al., 1999) to avoid conflicting definitions of the same term. RDF supports interoperability among different metadata standards.

RDF as a framework to enable machine-understandable metadata

Another major influence in the design of RDF comes from the Knowledge Representation (KR) community. KR is a field that focuses on how information can be encoded and utilized in computational models of cognition (Hayes , 1999). KR research considers how to represent knowledge such that machines can make use of it. RDF is designed to represent metadata for Web resources that are not only readable to machines, but also capable of being automatically processed.

One of the design goals of the RDF is to make the semantics of the Web resources accessible to machines. Although the information on the Web can be read by programs, the semantics are undefined. RDF allows the semantics of the Web pages to be represented by metadata in a form that machines can make use of in an intelligent manner (Lassila , 1998). RDF aims to make the information on the Web a knowledge base for intelligent software agents. Lassila, the designer of RDF, discussed how the intelligent software agents can make use of these machine-understandable semantics (Lassila , 1997).

RDF for better Web resource discovery

RDF addresses some of the limitations of Web resource discovery (Swick , 1999). Much of the material on the Web is not in text format, e.g. images, music, video. Existing methods of Web page indexing are not applicable to these resources. RDF aims to provide a uniform metadata framework such that these resources can be discovered from this information. The use of RDF also aids in automated classification and organization of Web resources, which is traditionally performed manually, e.g. by Yahoo! (<http://www.yahoo.com/>). The automated classification of Web resources would make searching the Web more efficient (Bray, 2001).

RDF as an extensible language

Berners-Lee, the director of the W3C, stated the goal of the W3C as follows.

“... to lead **Evolution** of the Web while maintaining its **Interoperability** as a universal space. Interoperability and Evolvability were two goals for all W3C technology, ...”
(Berners-Lee, 1998a)

RDF is designed for evolvability. Berners-Lee and Connolly have described some required properties of a language for a rapid growth environment like the Web (Berners-Lee & Connolly, 1998). The language must be able to work with a *partial understanding* of the new features of the language. For example, in an HTML page, markup tags that are unknown to the browser are ignored by the browsers but the page is still displayed. The vocabulary and schema are important to the understanding of unknown features. The system can understand the new vocabulary by reading the definition specified in its schema. RDF defines the semantics of new vocabularies in *RDF Schema*. The language must also allow the mixing of vocabularies. RDF does this through the XML namespace facility (Bray et al., 1999).

The RDF specifications are released in two parts: the *Resource Description Framework (RDF) Model and Syntax Specification*¹ (Lassila et al., 1999) and the *Resource Description Framework (RDF) Schema Specification 1.0*² (Brickley et al., 2000)

B. RDF Data Model

The RDF data model can be introduced by considering a simple example sentence (from Lassila et al., 1999):

Ora Lassila is the creator of the resource
<http://www.w3.org/Home/Lassila>.

The sentence has the following parts:

Subject (Resource)	http://www.w3.org/Home/Lassila
Predicate (Property)	Creator
Object (literal)	"Ora Lassila"

In the RDF data model, the *Subject* and *Predicate* can be identified by URIs, while *Object* can be indentified by URIs or just strings.

This can be transformed into an RDF data model, represented as the following directed label graph (DLG).



Figure 9: RDF data model for a basic statement

¹ RDF Model and Syntax Specification became a W3C recommendation in February 1999.

² RDF Schema Specification became a W3C candidate recommendation in March 2000

In the graph, the oval represents a resource and the box represents an object that is identified by a string. A resource is anything that can be identified by URI.

The RDF data model can also be represented in the following triple “*Predicate (Subject, Object)*”

```
Creator ( 'http://www.w3.org/Home/Lassila', "Ora Lassila" )
```

The basic RDF statement provides an ability to describe Web resources as attribute-value pairs. The RDF data model also allows document authors to describe their resources using more expressive statements, such as a statement about a property that has multiple values, e.g. a page may be created by multiple authors, or a statement about list of resources that share the same value, i.e. a list of pages created by a person. RDF has defined the notion of *container* to allow the statement about a collection of resources or collection of strings. Container can be one of the following three types.

- *Bag*: used when the order of the items in the collection is not important and duplicates are allowed.
- *Sequence*: used when the order of the items in the collection is important and duplicates are allowed.
- *Alternative*: used when any one of the items in the collection can be picked, i.e. list of Internet mirror sites

RDF allows duplication of metadata statements; thus there is no definition of ‘*Set*’ (unordered list without duplication) in the RDF data model.

RDF also allows people to make statements about statements. This *higher-order* statement requires that a statement must also be able to be regarded as an object. This process is called *reification*, a term that has been widely used in Knowledge Representation as a mechanism of transforming a statement into an object. RDF implements this mechanism by explicitly describing the statement as an object with four

properties: *rdf: subject*, *rdf: predicate*, *rdf:object* and *rdf:type*. Thus, the statement being described with these properties becomes a first-class object where people can make a statement about it. An example of higher-order statement can be shown as follows:

An example sentence:

*Ralph Swick says that Ora Lassila is the creator of the resource
http://www.w3.org/Home/Lassila. (from Lassila et al., 1999)*

This can be represented in RDF as:

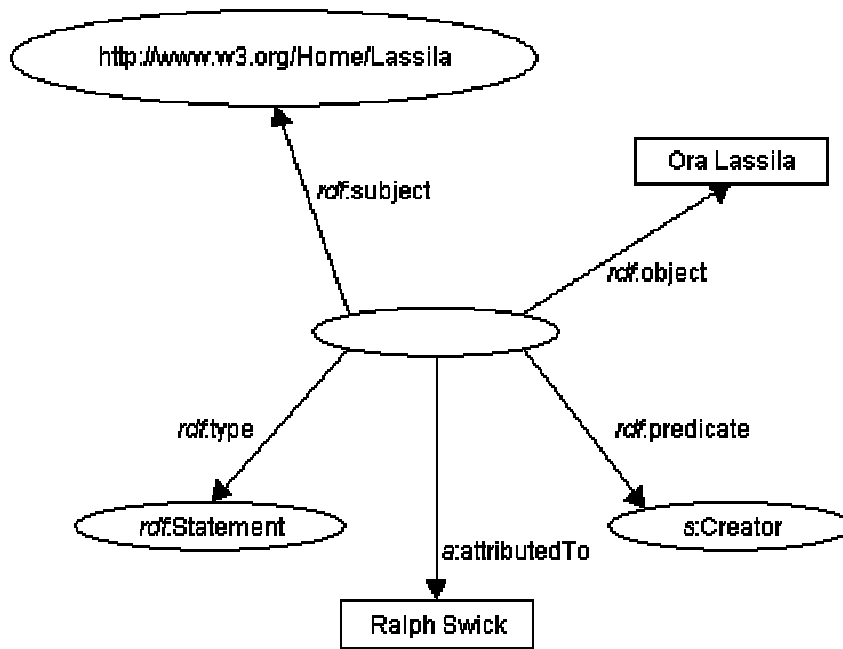


Figure 10: RDF data model for a high-order statement

In Figure 10, the empty oval represents the statement reified to become a resource, that can be described by five properties: four of which are about the sentence structure of the original statement (subject, predicate, object, type) and another one states that the original

statement is cited by (attributedTo) the specified person. The prefix “a” of the property “attributedTo” is a namespace to a schema where the definition of the property is defined.

C. RDF Syntax

A serialization syntax allows machines to create and exchange metadata information. Although RDF is independent of syntax, the designers of RDF have chosen XML as the default syntax for RDF due to its strength as universal data interchange format. Another main reason is that it allows RDF to use the namespace facility of XML.

Namespaces allow a word to be defined in the context of dictionary. The need for namespaces arises when people want to use the same word to convey different meanings. For example, a word “creator” might mean “the author of resource” to one person but might mean “the publisher of the resource” to another person. Namespaces solve this problem by requiring that a word be defined with a reference to a dictionary. In XML namespaces, the dictionary is a schema, i.e. an XML Schema; a reference to it is given in the form of a URI (Berners-Lee et al., 1998). The use of the XML namespace facility in RDF helps to avoid the confusion and conflict between multiple definitions of the same term.

The following XML namespace declaration associates namespace prefix, “*rdf*”, with the URI of schema for RDF, <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

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

The RDF data can be represented in two kinds of XML syntax: *full serialization syntax* and *abbreviated syntax*.

Given the sentence,

Ora Lassila is the creator of the resource <http://www.w3.org/Home/Lassila>.

The full RDF’s XML syntax is:

```
<rdf:RDF>
```

```

<rdf:Description about="http://www.w3.org/Home/Lassila">
  <s:Creator>Ora Lassila</s:Creator>
</rdf:Description>
</rdf:RDF>

```

where the prefix ‘s’ refers to an XML namespace declaration, which defines the metadata terms, such as

```
xmlns:s="http://description.org/schema/"
```

For the purpose of compactness, RDF allows the syntax to be written in *abbreviated syntax* form. The example sentence can be written in the abbreviated form:

```

<rdf:RDF>
  <rdf:Description about="http://www.w3.org/Home/Lassila"
    s:Creator="Ora Lassila" />
</rdf:RDF>

```

One benefit of abbreviated syntax is that it hides RDF data in an XML or HTML document.

D. RDF Schema

Under RDF, a schema is used to define terms, as well as restrict terms usage. RDF Schema should not be confused with the XML Schema. RDF schema is a kind of machine-readable dictionary. For example, in a metadata application, an RDF schema declares the vocabulary of the metadata elements and their corresponding meanings. These meanings are described in term of the relationship between terms. The relationships allowed by the RDF Schema include *inheritance* and *instance-of* relationships.

In order to allow people to create RDF schema in a uniform way, W3C has released the RDF Schema Specification 1.0 (Brickley et al., 2000), as a separate specification from the RDF Model and Syntax specification (Lassila et al., 1999). The specification does

not specify vocabulary for metadata elements, i.e. “creator”, “subject”. It provides a language, *RDF schema specification language*, which allows people to define the meanings of metadata elements. The language itself is a meta-language in that it is a language used to create RDF Schema.

The following XML namespace declaration associates namespace prefix, “*rdfs*”, with the URI of RDF Schema for RDF Schema specification language, <http://www.w3.org/2000/01/rdf-schema#>.

```
xmlns:rdfs=" http://www.w3.org/2000/01/rdf-schema#"
```

Class and property

The RDF schema specification language allows the semantics of a new vocabulary to be expressed in terms of relationships to other vocabularies. The relationship can be expressed in terms of relation to an existing class, defined by “*rdfs:Class*”, or existing property, defined by “*rdf:Property*”. For example, “*rdfs:subClassof*” defines an *inheritance* relationship between two *classes*; “*rdfs:subPropertyof*” defines an *inheritance* relationship between two *properties*; “*rdf:type*” defines *instance-of* relationship between a *class* and a *resource*. RDF Schema also allow the definitions of vocabulary to be defined elsewhere. The references to external definitions can be defined using “*rdfs:seeAlso*” or “*rdfs:isDefinedBy*”.

Constraints

A vocabulary can also be defined in terms of constraints and restrictions. The RDF Schema specification language provides “*rdfs:range*” and “*rdfs:domain*” elements for expressing constraints. The “*rdfs:range*” element is used to specify the value restriction of a property. For example, the following RDF statement says that the value of “*author*” property must be a resource of “*Person*” class.

```
<rdf:Description ID="author">  
  <rdfs:range rdf:resource="#Person"/>
```

```
</rdf>
```

RDF Schema specification allows property constraints to be defined in terms of the class where the property can be used. For example, to state that the “author” property can be used with “Book” class, the “rdfs:domain” element would be used

```
<rdfs:Description ID="author">  
  <rdfs:domain rdf:resource="#Book"/>  
  <rdfs:range rdf:resource="#Person"/>  
</rdf>
```

To make an RDF Schema extensible, different versions of the same RDF schema are advised have different URI to avoid ambiguity.

4.1.5 Topic Maps

Topic Maps are an ISO standard for “describing knowledge structures and associating them with information resources” (Pepper, 2000). The origin of Topic Maps was SOFABED (Standard Open Formal Architecture for Browsable Electronic Documents) (Biezunski & Newcomb, 2001). SOFABED was created in 1991 by UNIX system vendors to create consistent terms for a master index of technical documentation. In 1993, the Conventions for the Application of HyTime (CApH) group elaborated the SOFABED model and renamed it as *Topic Maps*. The Topic Maps standard was finalized and published in January 2000 as ISO/IEC 13250:2000 (International Organization for Standardization, 1999). To enable the applications of Topic Maps on the WWW, the XML grammar for interchanging Web-based Topic Maps, called the *XTM Specification* (Topicmaps.Org Authoring Group, 2001), is being developed by TopicMaps.Org (<http://www.topicmaps.org>).

Pepper (Pepper, 2000) describes the basic concepts of topic maps as Topics, Associations and Occurrences (TAO). The examples provided by (Pepper, 2000) are given in the domain of opera.

Topics are *subjects*. A topic is an instance of zero or more *topic types*. A topic normally has *topic name*. A topic name consists of one or more of the following types: *base name* (required), *display name* (optional) and *sort name* (optional). However, no two subjects can have exactly the same name in the same scope (topic naming constraint). For example, “Tosca” is a topic of the topic type “opera” with a topic name of “Tosca”.

Occurrence or *topic occurrence* is the information that is specified as relevant to a given subject (International Organization for Standardization, 1999). Occurrence of a topic can be outside or inside the topic map. The addressing mechanism can be provided by HyTime (International Organization for Standardization, 1997) addressing or XPointers (DeRose et al., 2001). *Occurrence role* is the sense in which the occurrences are relevant to a topic (International Organization for Standardization, 1999). Examples of occurrence roles include “article”, “illustration”, etc.

Association is a link element that asserts a relationship between two or more subjects (Pepper, 2000). Subjects can be related by an *association type*, where each subject has its *association role*. For example, an association of “Tosca was written by Puccini” relates the topic of “Tosca” to the topic of “Puccini”. The association type of this association could be “written_by” and the association roles are “opera” and “composer”.

Other major components of topic maps include identity (and public subjects), facets and scope (Pepper, 2000). *Public subject* provides the mechanism to identify the *identity* attribute of a topic. *Facets* provide a mechanism to assign metadata to the information resources. Facet is considered redundant in topic maps and has not been adopted in the XTM specification (Topicmaps.Org Authoring Group, 2000). *Scope* avoids ambiguity by giving the context of the topic characteristics.

Topic Maps share similar goals to the RDF initiative as both aim to describe metadata and relationships between Internet resources. Both standards are also influenced by the KR community. The interoperability between Topic Maps and RDF is currently an open

issue. Lacher & Decker (Lacher & Decker, 2001) have shown a transformation between XTM Topic Maps and RDF data. The transformation generates an RDF graph representation from a Topic Maps graph representation. This allows Topic Maps data sources to be queried by an RDF-aware query processor. Freese (Freese, 2000) provides a comparison between Topic Maps and RDF.

4.2 Semantic Web

The vision of the *Semantic Web* was introduced in 1998 by Berners-Lee, director of the World Wide Web Consortium (W3C), in his personal notes on the high-level view of the architecture of the World Wide Web (Berners-Lee, 1998b; Berners-Lee, 1998c). The vision has been further clarified in “Weaving the Web” (Berners-Lee, 2000d). Berners-Lee views the Semantic Web as “*a Web of data that can be processed directly or indirectly by machines*” (Berners-Lee, 2000c). The Semantic Web is the proposed architecture for the future Web, where data on the Web is in a form that can be automatically processed by machines. The architecture of the Semantic Web as envisioned by Berners-Lee is shown in Figure 1.

4.2.1 Definition

W3C Semantic Web Activity Statement (W3C, 2001) includes the following explanation of the Semantic Web:

The Semantic Web is a vision: the idea of having data on the Web defined and linked in a way that it can be used by machines not just for display purposes, but for automation, integration and reuse of data across various applications.

4.2.2 The roles of XML and RDF to the Semantic Web

RDF uses XML as its composition layer. The rules for specifying RDF in XML can be found in XML Schema for RDF (<http://www.w3.org/2000/07/rdf.xsd>) (Connolly, 2001). XML and RDF both play an important role for the Semantic Web (Decker et al., 2000b).

XML allows different schema to be specified, which allows document instances (resources) to be created. However, different XML documents using different XML

schema could be used to represent the same meaning. This makes it difficult for the machine to interpret the semantic equivalence of the two XML documents. RDF provides a mechanism to define the semantics of Web documents in the form of metadata. RDF defines a uniform data model in the form of RDF triple: *Predicate*, *Subject* and *Object*. The RDF specification (Lassila et al., 1999) specifies how the RDF processing software should behave when it encounters RDF elements. The RDF specification requires that the RDF processing software translate any syntactic representations of RDF statements back to the RDF triples. This canonical form makes it possible for the machine to interpret the semantics of Web documents independently of the syntax. The definition and relationship of terms used in an RDF statement are defined in an RDF Schema. The RDF Schema specification (Brickley et al., 2000) specifies how RDF Schema could be created and processed.

To realize the full potential of the Semantic Web, ontology and logic layers are required on top of the RDF layer. This has been illustrated as the three layers on top of the RDF layer in Berners-Lee's Semantic Web architecture (Figure 1, page 8). The ontology layer is needed to provide common terminologies in a domain of knowledge. The logic layer provides formal semantics, which is important to the machine-automated process, i.e. a reasoning service.

5. Semantic Interoperability on the World Wide Web

The Web represents a large knowledge base for humans. The Semantic Web (Berners-Lee, 2000c) will allow machines to use the information on the WWW. However, in order for machine to understand what resides in the WWW knowledge base, there needs to be an agreement on the meaning of the information. Put another way, there must be semantic interoperability.

An ontology defines the relationships among a set of concepts. Ontology is created for the purpose of sharing common knowledge in a domain of interest. An ontology provides a common understanding of definitions of terms in the domain and specifies relationships between the terms. Ontologies are considered critical to the achievement of the semantic interoperability on the Web.

In order for ontologies to interoperate, there need to be a unified framework for creating and sharing ontology on the Web. RDF (Lassila et al., 1999) and RDF Schema (Brickley et al., 2000) efforts attempt to provide a unified framework for web semantics in the form of metadata. From an ontology viewpoint, RDF Schema could be considered a way to represent an ontology. RDF Schema specification language provides basic primitives for modeling ontology, such as “subClassOf” or “subPropertyOf” primitives. According to Heflin et al. (2000b) and Broekstra et al. (2000), the expressiveness of the RDF Schema language is insufficient for representing general ontologies. This has led to some efforts to design a knowledge representation language that has more expressive power for creating and sharing ontologies on the Web. This chapter reviews the current state of this effort.

5.1 Ontology on the Web

There are many ways to specify terms on the Web. Lehman, McGuinness, Ushold and Welty (AAAI-99) came up with the list of possible forms of term specifications. McGuinness illustrates the spectrum of these possible forms of ontology in Figure 11 (McGuinness, 2001). The description of the spectrum as described by McGuinness (2001) is summarized as follows.

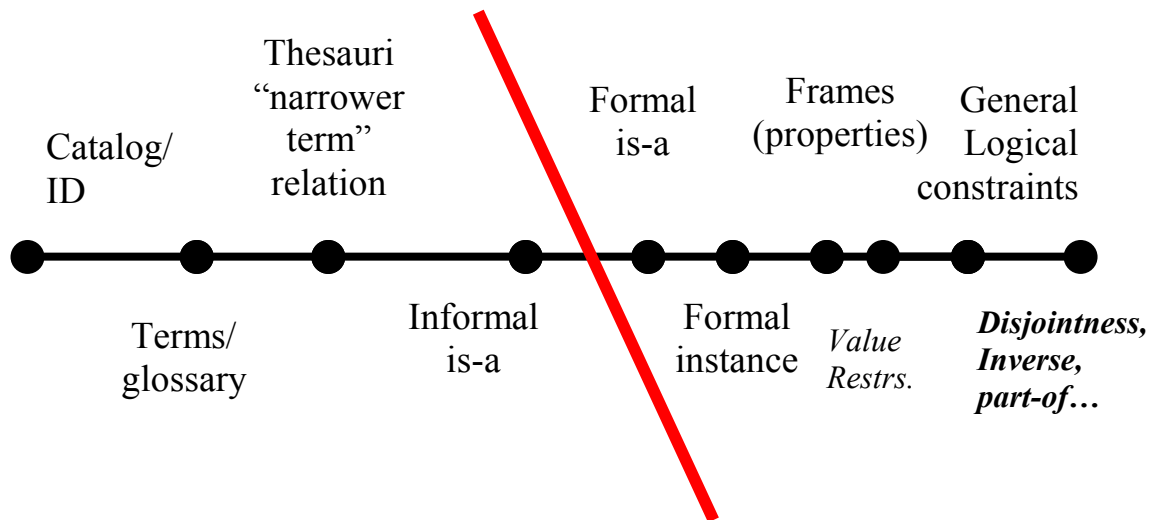


Figure 11: Ontology Spectrum

One of the simplest forms of term specification is **controlled vocabularies**, such as those used by the catalogers. Another simple form of term specification is a **glossary** where the meanings of terms or vocabulary are specified in natural language. These meanings are intended for human consumption, rather than for machines. **Thesauri** give another dimension of meaning, where the meaning of a term is defined in terms of its relationship to other terms. The relationships are usually defined in simple forms, such as narrower and broader terms, and usually are not hierarchically structured. The simplest form of hierarchically structured term relationships can be found in many WWW sites. Yahoo! (<http://www.yahoo.com/>) is a good example. The underlying meaning of the Yahoo! hierarchical scheme, however, is not explicitly defined. The links in the tree do not necessarily represent subclass (is-a) relationships. Thus it is difficult for the machines to automatically determine term relationships from this kind of hierarchical structure (**informal is-a**).

Due to the ambiguity and the lack of explicit semantics in these forms, the term specification schemes mentioned in the previous paragraph are not considered ontologies suitable for machine use. McGuinness specifies three essential properties of Web ontologies (McGuinness, 2001).

- Finite controlled (extensible) vocabulary
- Unambiguous interpretation of meanings and relationships
- Strict hierarchical subclass relationships

The forms of web ontology on the right side of the separation line in the ontology spectrum have these characteristics. A web ontology must have a strict subclass relationship. That is every instance of the child nodes must be an instance of the parent node as well (**formal is-a**). Some ontology schemes may also include instances of a class as nodes in the tree (**formal instance**). The properties of each node can be further defined using a **frames** (Minsky, 1975) approach. These properties may allow only certain values to be filled in (**value restrictions**). A more expressive ontology might allow the expression of other kinds of relationships, such as disjoint classes (i.e. male vs. female), inverse relationship (i.e. create vs. is-created-by) or part-whole relationship (i.e. has-a link).

According to McGuinness, **language** and **environment** are two major concerns in the deployment of ontology. Language is crucial to how the ontology is created and represented. Environment is crucial to how the ontology is maintained and used over time.

5.2 An ontology language for the Web

An ontology language allows an ontology to be created and represented such that machines can make use of it. An ontology language should:

- be expressive enough to capture many ontologies;
- have a common syntax and should be easy to integrate with ontologies that are created in other languages;

- have formal semantics such that machines can understand and reason on it. (van Harmelen, 2000)

RDF allows one to express the semantics of a web page in term of metadata. From a knowledge representation viewpoint, RDF can be considered as a simple frame-based system (Lassila & McGuinness, 2001). The mechanism of specifying relationships between concepts is influenced by the Semantic Network research (Quillian, 1967; Woods, 1975). Lassila and McGuinness compare RDF, Object-oriented programming (OOP), [frame-based systems](#), and [description logic](#), as shown in Table 2 (Lassila et al., 2001).

OOP Systems	Frame Systems	Description Logic	RDF
Object	Frame, instance, individual	Individual	Resource
Attribute	Slot	Role	Property
Value	Filler	Filler	Property value
Class	Frame, schema	Concept	Class

Table 2: An analogy of RDF terms to OOP, Frame systems and Description Logics

One might consider an RDF statement as the assertional part, known as the [ABox](#) in description logic. One might consider RDF Schema as the terminological part, known as [TBox](#) in description logic. From an ontology viewpoint, RDF Schema specification provides a set of primitives for modeling simple ontologies in RDF Schema. These primitives include “SubClassOf” and “SubPropertyOf”.

However, as an ontology language, the RDF Schema specification language has been criticized for its expressive inadequacy (Heflin et al., 2000b; Broekstra et al., 2000). Logical connectives such as conjunction, disjunction, and negation are not provided in RDF Schema. For example, one can not define disjointed classes using the negation operator. RDF Schema do not allow one to define a property (attribute) of a property. For example, one cannot define whether a property is transitive ($a(x,y), a(y,z) \rightarrow a(x,z)$) or

symmetric ($a(x,y) = a(y,x)$). These have limited the expressive power of the ontology. Another major criticism of RDF Schema specification language is its lack of formal semantics. The mapping of RDF Schema to some form of logic is not provided. The lack of an underlying formal logic eliminates its capability to provide reasoning support, which is crucial to automated processing.

The inadequacy of the RDF Schema specification language as an ontology language has motivated the design of full knowledge representation languages for representing and creating ontologies. These languages usually offer more expressive power for representing an ontology. The formal semantics of these languages are usually defined in logic, mainly in the description logic family, in order to gain its efficient reasoning support.

5.2.1 Simple HTML Ontology Extension (SHOE)

One attempt to define a more expressive ontology language is SHOE (Simple HTML Ontology Extension). SHOE is a knowledge representation language that allows web pages to be annotated with ontology-based semantics (Heflin, Hendler, & Luke, 1999b). SHOE has been proposed as an extension to HTML. SHOE was developed at the University of Maryland at College Park in 1996 (Luke, Specter, & Rager, 1996), prior to the development of XML and RDF. The syntax of SHOE is defined in a DTD (initially an SGML DTD and later an XML DTD (Luke & Heflin, 2000)).

SHOE separates the terminological descriptions, known as *ontology* part, from the assertions, known as *instance* part. The ontology part in SHOE allows one to define *Category* definitions. Category in SHOE is analogous to *Concept* in description logic or *Class* in the OO paradigm. The SHOE ontology also allows one to define *Relation* definitions. Relation in SHOE is analogous to *Role* in description logics or *Attribute* in the OO paradigm. A Relation in SHOE can be an n-ary predicate, while Role in description logics is a binary predicate. SHOE also allows inference rules to be defined in the ontology specification in the form of horn clause, i.e. $a \wedge b \wedge c \Rightarrow d$. An example of SHOE ontology definition is shown in the following example (Heflin et al., 1999b).

```

<HTML>
<HEAD>
  <TITLE>University Ontology</TITLE>
  Tell agents that we're using SHOE
  <META HTTP-EQUIV="SHOE" CONTENT="VERSION=1.0">
</HEAD>
<BODY>
  Declare an ontology called "university-ontology".
  <ONTOLOGY ID="university-ontology" VERSION="1.0">
  Borrow some elements from an existing ontology, prefixed with a "b."
  <USE-ONTOLOGY ID="base-ontology" VERSION="1.0" PREFIX="b"
    URL="http://www.cs.umd.edu/projects/plus/SHOE/base.html">
  Define some categories and subcategory relationships
  <DEF-CATEGORY NAME="Person" ISA="b.SHOEentity">
  <DEF-CATEGORY NAME="Organization" ISA="b.SHOEentity">
  <DEF-CATEGORY NAME="Worker" ISA="Person">
  <DEF-CATEGORY NAME="Advisor" ISA="Worker">
  <DEF-CATEGORY NAME="Student" ISA="Person">
  <DEF-CATEGORY NAME="GraduateStudent" ISA="Student Worker">
  Define some relations; these examples are binary, but relations can be n-ary
  <DEF-RELATION NAME="advises">
    <DEF-ARG POS=1 TYPE="Advisor">
    <DEF-ARG POS=2 TYPE="GraduateStudent"></DEF-RELATION>
  <DEF-RELATION "age">
    <DEF-ARG POS=1 TYPE="Person">
    <DEF-ARG POS=2 TYPE="b.NUMBER"></DEF-RELATION>
  <DEF-RELATION "suborganization">
    <DEF-ARG POS=1 TYPE="Organization">
    <DEF-ARG POS=2 TYPE="Organization"></DEF-RELATION>
  <DEF-RELATION "works-for">
    <DEF-ARG POS=1 TYPE="Person">
    <DEF-ARG POS=2 TYPE="Organization"></DEF-RELATION>
  Define a transfers-through inference over working for organizations
  <DEF-INFERENCE>
    <INF-IF>
      <RELATION NAME="works-for">
        <ARG POS=1 VALUE="x" VAR>
        <ARG POS=2 VALUE="y" VAR></RELATION>
      <RELATION NAME="suborganization">
        <ARG POS=1 VALUE="y" VAR>
        <ARG POS=1 VALUE="z" VAR></RELATION></INF-IF>
    <INF-THEN>
      <RELATION NAME="works-for">
        <ARG POS=1 VALUE="x" VAR>
        <ARG POS=2 VALUE="z" VAR></RELATION></INF-THEN>
    </DEF-INFERENCE>
  </ONTOLOGY>
</BODY>
</HTML>

```

Table 3: SHOE ontology example

In the university ontology example, five categories are defined: *Person*, *Organization*, *Worker*, *Advisor*, *Student* and *GraduateStudent*. These categories are defined in terms of their relationships to each other and to the base entities (defined in the SHOE base-ontology). The example ontology also defines four relations: *advises*, *age*,

suborganization and *works-for*. These relations are defined in terms of their value restrictions, which can be either in the form of allowed categories or allowed data types. SHOE currently supports four basic types: strings, numbers, dates and booleans. Inference rules can also be defined in the ontology. From the example ontology, the inference rule defines that every worker who works for an organization that is a suborganization of an organization would imply that the worker also works for the parent organization. This is equivalent to the following FOL sentence: $(\forall x \in \text{Worker}) (\forall y \in \text{Organization}) (\forall z \in \text{Organization}) \text{works-for}(x,y) \wedge \text{suborganization}(y,z) \Rightarrow \text{works-for}(x,z)$.

Once the term meanings and their relationships are defined in the ontology, one can make claims (assertions) using the terms previously defined in the ontology. The claims can be *category* claims, such as a claim that Mike is a graduate student. Where the definition of the category of graduate student (*GraduateStudent*) is defined in the *university* ontology, in which graduate student is defined as conjunction of *student* and *worker* categories, where *student* is a subclass of *person* category and *worker* is also a subclass of *person* category. The web page may also make a *relation* claim that Mike is advised by John, where *advise* is a relation of *Advisor* and *GraduateStudent*. This is shown in the following SHOE instance example (Heflin et al., 1999b).

```

<HTML>
<HEAD>
  <TITLE>John's Web Page</TITLE>
Tell agents that we're using SHOE
  <META HTTP-EQUIV="SHOE" CONTENT="VERSION=1.0">
</HEAD>
<BODY>
  <P>This is my home page, and I've got some SHOE data on it about me and
my advisor. Hi, Mom!</P>
Create an Instance. There's only one instance on this web page, so we might as
well use the web page's URL as its key. If there were more than one instance,
perhaps the instances might have keys of the form http://univ.edu/john#FOO
  <INSTANCE KEY="http://univ.edu/john">
Use the semantics from the ontology \university-ontology", prexed with a \u."
  <USE-ONTOLOGY ID="university-ontology" VERSION="1.0" PREFIX="u"
URL="http://univ.edu/ontology">
Claim some categories for me and others.
  <CATEGORY NAME="u.GraduateStudent">
  <CATEGORY NAME="u.Advisor" FOR="http://univ.edu/mike">
Claim some relationships about me and others. \me" is a keyword for the
enclosing instance.
  <RELATION NAME="u.advises">
    <ARG POS=1 VALUE="http://univ.edu/mike">
    <ARG POS=2 VALUE=me> </RELATION>
  <RELATION NAME="u.age">
    <ARG POS=1 VALUE=me>
    <ARG POS=2 VALUE="32"> </RELATION>
  </INSTANCE>
</BODY>
</HTML>

```

Table 4: SHOE instance example

Compared to RDF, the ontology part in SHOE is analogous to an RDF Schema, while a claim in the instance part of SHOE is analogous to an RDF statement. More specifically, the *<CATEGORY>* tag, used in an instance claim in SHOE, is analogous to *rdf:type*. The *<DEF-CATEGORY>* tag used in ontology definition is analogous to *rdfs:subclassOf*. The *<DEF-RELATION>* in SHOE is analogous to *rdfs:range*. However, some expressive power in RDF Schema is not provided in SHOE. For example, RDF Schema allows the expression of subclass relationship of properties with *rdfs:subPropertyOf*, which is not

provided in SHOE. RDF Schema also provides the *rdfs:domain* primitive to specify the class name that the property can be applied to. This is not provided in SHOE either. However, SHOE provides expressions of inference rules in the form of horn clauses, e.g. $a \wedge b \wedge c \Rightarrow d$. The expression of inference rules enables reasoning support in SHOE. This is its major advantage over RDF.

SHOE is one of a few ontology languages that focuses on the consistency of assertions (Heflin, Hendler, & Luke, 1999a; Heflin & Hendler, 2000a). Consistency is a major issue in a distributed knowledge base environment like the Web. SHOE prevents contradictions by allowing no retractions of knowledge from the knowledge base. SHOE does not allow negation in the claim statement. SHOE also includes the claimer along with the claim statements such that one can identify who has made the false claims. The consistency of the ontology is maintained by a versioning mechanism. Each ontology must have the version number associated with it. Different versions of the same ontology must be in separate files.

Heflin and Hendler describe how SHOE and its tools can be used in action in (Heflin & Hendler, 2001).

5.2.2 Ontology Inference Layer (OIL)

The Ontology Inference Layer¹ (OIL) is a proposed representation and inference language for ontology on the Web. The development of OIL is sponsored by the European Community via the On-to-knowledge project (<http://www.ontoknowledge.org/>). The core partners involved in OIL include University of Manchester (UK), Vrije Universiteit Amsterdam (NL), Stanford University (USA), University of Karlsruhe (Germany), Administrator Nederland (NL), Research Bell Labs (USA), and MIT (USA).

One of the motivations for OIL is the need for an expressive language for creating ontologies on the Web. RDF has inadequate of expressiveness and lacks formal

¹ Also known as the Ontology Inference Language

semantics and reasoning support. OIL is designed to overcome these limitations by extending the RDF Schema standard. OIL proposes to define an additional layer that provides formal semantic and reasoning capability on top of the RDF Schema layer (Decker et al., 2000a; Broekstra et al., 2000).

OIL is designed to be an extensible standard. To achieve this, OIL uses the layered approach. The lowest level of the OIL standard, known as Core OIL, is compatible with RDF Schema specification, except for the RDF reification mechanism, which is not supported in OIL. Ontologies defined by the Core OIL language are interpretable by an RDF Schema aware application. The next layer, Standard OIL, adds more features to OIL which makes it only partially understood by an RDF Schema aware application. Standard OIL is designed such that it can provide adequate expressive power as well as reasoning support. Instance OIL includes full integration of individuals (instances) into the language. The layers are illustrated in the following figure (Fensel, van Harmelen, Horrocks, McGuinness, & Patel-Schneider, 2001).

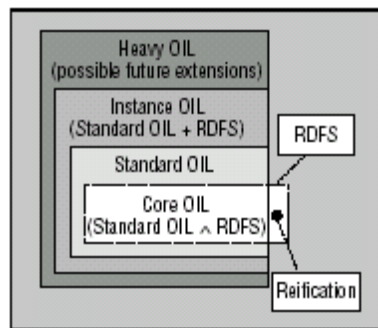


Figure 12: OIL's layered language model

An example OIL ontology is provided in Table 5 (Horrocks et al., 2000).

```

1  ontology-container
2      title "African animals"
3      creator "Ian Horrocks"
4      subject "animal, food, vegetarians"
5      description "A didactic example ontology describing African
6      animals"
7      description.release "1.01"
8      publisher "I. Horrocks"
9      type "ontology"
10     format "pseudo-xml"
11     format "pdf"
12     identifier "http://www.cs.vu.nl/~dieter/oil/TR/oil.pdf"
13     source "http://www.africa.com/nature/animals.html"
14     language "OIL"
15     language "en-uk"
16     relation.hasPart
17     "http://www.ontosRus.com/animals/jungle.onto"
18 ontology-definitions
19     slot-def eats
20     inverse is-eaten-by
21     slot-def has-part
22     inverse is-part-of
23     properties transitive
24     class-def animal
25     class-def plant
26     subclass-of NOT animal
27     class-def tree
28     subclass-of plant
29     class-def branch
30     slot-constraint is-part-of
31     has-value tree
32     class-def leaf
33     slot-constraint is-part-of
34     has-value branch
35     class-def defined carnivore
36     subclass-of animal
37     slot-constraint eats
38     value-type animal
39     class-def defined herbivore
40     subclass-of animal
41     slot-constraint eats
42     value-type plant OR (slot-constraint is-part-of has-value
43     plant)
44     class-def herbivore
45     subclass-of NOT carnivore
46     class-def giraffe
47     subclass-of animal
48     slot-constraint eats
49     value-type leaf
50     class-def lion
51     subclass-of animal
52     slot-constraint eats
53     value-type herbivore
54     class-def tasty-plant
55     subclass-of plant
56     slot-constraint eaten-by
57     has-value herbivore, carnivore

```

Table 5: An OIL ontology example

An OIL ontology consists of two major parts: the *ontology container* and the *ontology definition*. The ontology container provides metadata information for the ontology, such as the title of the ontology, the author of the ontology, etc. OIL uses the Dublin Core Metadata Element Set, Version 1.1 standard (Dublin Core Metadata Initiative, 1999) for describing this metadata information. The ontology definition consists of a set of expressions that describe classes and slots. The ontology definition provides the following kinds of expressions: *class definition*, *slot constraints* and *slot definition*.

Class definition (class-def) associates a class name with its description. Class definitions are *primitive* or *defined*. A class is primitive type if it can be necessarily defined in terms of other classes but not vice versa. From the animal ontology example, lion is a primitive class because it can be necessarily defined as animal but not vice versa ($\text{lion} \Rightarrow \text{animal}$). Herbivore, however, is a defined class because it can be necessarily and sufficiently described in term of animal, whose “eat” slots must all be filled in by one of plant type ($\text{herbivore} \Leftrightarrow \text{animal} \wedge \forall \text{eats.plant}$). In OIL, a class has primitive type by default. A class definition can be defined in terms of a set of class expressions. A class expression can be expressed as its subclass relationship or its slot constraint. For example, the definition of Herbivore class is shown in lines 37-42 of Table 5.

Slot constraint (slot-constraint) can be defined by one of the following: *has-value*, *value-type*, *max-cardinality*, *min-cardinality*, and *cardinality*. The *has-value* constraint is equivalent to the existential quantifier of role in description logic ($\exists R.C$). The *value-type* constraint is analogous to the universal quantifier of role in description logic ($\forall R.C$). The *max-cardinality* and *min-cardinality* give the specific number of instances that are allowed for the slot. The *cardinality* is used when the *min-cardinality* and *max-cardinality* are the same. The cardinality expressions in OIL are similar to number restrictions in description logic. The current version of OIL allows two basic data types: integer and string. The data types allowed in OIL are limited to improve computability.

Slot definition (slot-def) associates a slot name with its description. A slot definition can include the following components: *subslot-of*, *domain*, *range*, *inverse* and *properties* (transitive or symmetric). Compared to RDF Schema specification language, *oil:subslot-of* is equivalent to *rdfs:subPropertyOf*. The *domain* and *range* elements in OIL have the same meaning as *rdfs:domain* and *rdfs:range* respectively. The *inverse* and *properties* elements have no equivalent in RDF Schema specification language. *Inverse* allows definition of the slot as having an inverse relationship with other slots. In the animal ontology example, the slot “is-eaten-by” is defined as the inverse of the slot “eats”. An inverse is shown in lines 17-18 of Table 5. Properties of slots in OIL can be defined. A slot can be transitive ($a(x,y) \wedge a(y,z) \Rightarrow a(x,z)$) or symmetric ($a(x,y) \Rightarrow a(y,x)$). For example, one might define the slot “longer-than” as transitive ($\text{longer-than}(x,y) \wedge \text{longer-than}(y,z) \Rightarrow \text{longer-than}(x,z)$) while defining the slot “live-with” as symmetric ($\text{live-with}(x,y) \Rightarrow \text{live-with}(y,x)$).

OIL provides formal and clear semantics for the ontology language by mapping the OIL expressions to description logic.

The formal logic used in OIL is an extension of the \mathcal{ALC} language, known as \mathcal{SHIQ} . The letter \mathcal{S} in \mathcal{SHIQ} is shorthand for the $\mathcal{ALC}_{\mathcal{R}}$ language, which is an extension of the \mathcal{ALC} language that includes Role transitivity. \mathcal{SHIQ} extends the $\mathcal{ALC}_{\mathcal{R}}$ language by adding a hierarchy of roles (\mathcal{H}), inverse roles (\mathcal{I}) and fully qualified number restrictions (\mathcal{Q}) (Horrocks et al., 2000). In order to enable support for concrete data types such as integer and string, \mathcal{SHIQ} has been extended to $\mathcal{SHIQ}(d)$. The $\mathcal{SHIQ}(d)$ logic can capture the semantics of both Standard OIL and Instance OIL. A complete mapping of OIL language to $\mathcal{SHIQ}(d)$ description logic can be found in (Horrocks, 2000).

The reasoning support for \mathcal{SHIQ} is implemented in FaCT (Fast Classification of Terminologies - <http://www.cs.man.ac.uk/~horrocks/FaCT>). FaCT uses the optimized tableaux implementation. The reasoning services provided in FaCT include finding concept inconsistency and determining subsumption relationships. Currently, only TBox

reasoning is implemented in FaCT. The inclusion of ABox reasoning in FaCT is currently underway (Horrocks, 2000). FaCT is implemented in Common Lisp. In order to allow the use of FaCT in non-Lisp client applications, FaCT provides a CORBA (<http://www.omg.org/corba/>) interface.

Even though *SHIQ* can provide an efficient reasoning service, its lack of support for expressing instance (individual) in a class expression has been a major limitation of the expressiveness of the ontology language. There are many cases where expressing class definition in terms of instance is useful. For example, one might want to define the class of “Italian” as “person” who was born in “Italy” (Horrocks & Sattler, 2001). In this case, Italy is an instance of “Country” class. This cannot be expressed in the ontology language that has no support for instance in a class expression. *SHIQ* logic does not support this kind of expression, thus adding this form of expression to OIL ontology language would result in no mapping to description logic. As a result, there would be no reasoning support from the description logic. To overcome this limitation, *SHOQ(D)* has been proposed recently (Horrocks et al., 2001) as a formal logic for OIL in place of *SHIQ*.

SHOQ(D) is an extension of *SHQ* (*SHIQ* without support for inverse roles). *SHOQ(D)* extends *SHQ* by allowing instance in class definition or named individual (*O*) and has support for concrete data types (*D*). The reason that *SHOQ(D)* extends *SHQ*, rather than *SHIQ*, is that reasoning with inverse roles is known to be difficult or even intractable when combined with either concrete data types or named individuals (Horrocks et al., 2001). It is considered that the expression of inverse roles in ontology is less useful than the expressions of data types and named individuals, thus the support for inverse roles has been dropped in exchange for computability. The future work will investigate the restricted support for inverse roles without triggering computational complexity (Horrocks et al., 2001).

OIL has tools that support *ontology editing* and *ontology reasoning*. Ontology editors help in the manual creation of ontologies. There are currently three ontology editors for

OIL that are available (Fensel et al., 2001): OntoEdit, OILed, and Protégé. OntoEdit is an ontology development environment developed at the Knowledge Management Group of the University of Karlsruhe, Institute AIFB (<http://ontoserver.aifb.uni-karlsruhe.de/ontoedit>). The tool also has support for RDF Schema. OILed is a simple freeware ontology editor, developed at the University of Manchester (<http://img.cs.man.ac.uk/oil/>). Protégé (<http://www.smi.stanford.edu/projects/protege/>) was recently released as Protégé-2000 (Noy et al., 2001) and aims to support ontology development in different Semantic Web languages. For the reasoning support for ontology, OIL uses the FaCT as its inference engine.

OIL has some advantages over other Web ontology languages. These include its tight coupling to RDF and XML. Another major advantage of OIL is that the formal semantics are defined in description logic. This allows efficient reasoning support to be available in OIL. The approaches used by OIL have become a foundation for the DARPA Agent Markup Language (DAML).

5.2.3 The DARPA Agent Markup Language (DAML)

The DARPA Agent Markup Language (DAML – <http://www.daml.org/>) is built on W3C XML and RDF, OIL, SHOE, and related efforts (Dean, 2001). DARPA and the European Union Information Society Technologies Programme (IST) have formed the joint US/EU ad hoc Agent Markup Language Committee (<http://www.daml.org/committee/>), which includes the key persons from these related projects. The purpose is to define a unified framework for a Web ontology language by working closely with the existing Web ontology language efforts.

DAML released its first ontology language specification, DAML-ONT, in October, 2000 (Stein, Connolly, & McGuinness, 2000). In December, 2000, DAML+OIL (Horrocks & van Harmelen, 2000) had been released to replace DAML-ONT. DAML+OIL provides clearer semantics while making the language more consistent with the OIL project. This specification was later replaced by the latest version of DAML+OIL (Horrocks, van Harmelen, & Patel-Schneider, 2001a), released in March, 2000. One of the major

additions in this release is the support for XML Schema data types (Biron et al., 2000). The support for arbitrary data types, from the XML Schema type system, is a feature that has not yet been achieved in any other Web ontology language.

The formal semantics of DAML+OIL (March 2001) is provided as a mapping to first-order predicate logic¹ (axiomatic form) (Fikes & McGuinness, 2001) and in model-theoretic form (van Harmelen, Patel-Schneider, & Horrocks, 2001a). An example of DAML+OIL ontology (<http://www.daml.org/2001/03/daml+oil-ex.daml>), released with the specification, demonstrates the expressive power of the DAML+OIL language. It is also provided in Appendix C. The detailed explanation of the example can be found in (van Harmelen, Patel-Schneider, & Horrocks, 2001b).

DAML+OIL (March 2001) is divided into two parts. The first part, called the *object domain*, consists of objects that are members of classes defined in the DAML ontology. The latter part is called the *datatype domain*, which consists of values that belong to XML Schema data types. For example, in DAML+OIL, instances of class, e.g. the person “John Smith”, would be interpreted separately from instances of data types, e.g. the integer 5. Horrocks et al. (Horrocks, van Harmelen, & Patel-Schneider, 2001b) suggested that, by separating data types from classes, the data types would be modeled outside the ontology language. This helps in maintaining the simplicity and compactness of the ontology language. Separation of data types outside the ontology language also keeps reasoning support for the ontology language implementable.

A brief description of all major DAML elements can be described as follows. For brevity, the explanation focuses on the meaning and expressiveness of each DAML+OIL element rather than its syntax. A complete reference of all DAML+OIL elements can be found in the DAML+OIL reference description (van Harmelen, Patel-Schneider, & Horrocks, 2001c). The following XML namespace declaration associates namespace prefix,

¹ The axiomatic semantics of DAML+OIL (March 2001) are written in ANSI Knowledge Interchange Format (KIF – <http://logic.stanford.edu/kif/kif.html>)

“*daml*”, with the URI of RDF Schema for DAML+OIL (March 2001),
“<http://www.daml.org/2001/03/daml+oil.daml>”.

```
xmlns:daml=" http://www.daml.org/2001/03/daml+oil.daml "
```

DAML allows the following forms of expressions: *class element*, *class expression* and *property element*.

Class element associates a class name with its definition. Class definition is defined in terms of the following five optional elements: *rdfs:subClassOf*, *daml:disjointWith*, *daml:disjointUnionOf*, *daml:sameClassAs* and *daml:equivalentTo*. The *rdfs:subClassOf* allows the definition of a class to be defined in term of its subclass relationship to other classes. The *daml:disjointWith* allows class definition to be defined in term of its complement relationship to other classes. For example, “male” could be defined as a disjoint class of “female”. The *daml:disjointUnionOf* allows class definition to be defined in term of the union of disjoint classes. For example, human is a union of male and female, where male and female are disjointed classes. The *daml:sameClassAs* and *daml:equivalentTo*, in the context of class definition, share the same meaning of defining equivalence of classes. However, it is suggested that the use of *daml:sameClassAs* is preferred because the element is a subclass of *rdfs:subClassOf*, which make it partially understood by the RDF Schema parser.

Class expressions are possible forms that a class could be referred to. Class expression can be expressed in the form of one of the following: *a class name*, *an enumeration*, *property-restriction* or *their boolean combination*.

A class name is the name of the class whose definition is defined elsewhere. There are two predefined class names: *daml:Thing* and *daml:Nothing*. Every class is a subclass of *daml:Thing*, while *daml:Nothing* is a subclass of every class. From the instance viewpoint, every object is a member of *daml:Thing* and no object is a member of *daml:Nothing*. Enumeration is expressed by a *daml:oneOf* element followed by a list of

enumerated instances. For example, the “Continent” class can be expressed as one of the enumerated list: Eurasia, Africa, NorthAmerica, SouthAmerica, and Australia.

Class expression can also be expressed in term of property restrictions, using *daml:Restriction* and *daml:onProperty*. There are two kinds of property restrictions: *ObjectRestriction*, where the property must be an instance from the specified class, and *DatatypeRestriction*, where the property must have its value in the specified data type. The restriction can be expressed by one of the following elements: *daml:toClass*, *daml:hasClass* and *daml:hasValue*. The *daml:toClass* is analogous to the universal quantifier in predicate logic. It specifies the classes whose instances are allowed to be filled in the property. For example, “herbivore” may allow *only* instances of “plant” to be filled in its “eat” property (herbivore only eats plant). The *daml:hasClass* is analogous to the existential quantifier in predicate logic. It specifies that there exists at least one of the property that is filled in by the instance of the specified class or data type. For example, “human” may allow, but not be limited to, instances of “plant” to be filled in its “eat” property (human eats some plants). The *daml:hasValue* specified the specific instance of class or data type value that is allowed to fill in the property. For example, “Italian” is a class of person whose “nation” property must have the value of “Italy”, where “Italy” is an instance of “Country” class.

The property restriction can also be expressed in term of its cardinality restrictions: *daml:maxCardinality*, *daml:minCardinality* and *daml:cardinality*. The *daml:maxCardinality* and *daml:minCardinality* elements specify the maximum and minimum number of instances to be filled in the specified property. The *daml:cardinality* is the shorthand that is used when the number specified in *daml:maxCardinality* equals *daml:minCardinality*. For example, the “father” property allows only one instance to fill in. In this case, the *daml:maxCardinality*, *daml:minCardinality* and *daml:cardinality* of “father” property would all be equal to one. The expression of cardinality restriction could be expressed more specifically using the qualified cardinality restriction. For example, one might want to restrict the “father” property to allow only one instance of

“doctor” class to fill in. This can be expressed in DAML using *daml:maxCardinalityQ*, *daml:minCardinalityQ* or *daml:cardinalityQ* respectively.

DAML also allows the combination of class expressions to form a new class expression using the logical connectives. The connectives are expressed by one of the following elements: *daml:intersectionOf*, *daml:unionOf*, and *daml:complementOf*. The *daml:intersectionOf* is analogous to the logical conjunctive operator (AND). The *daml:unionOf* is analogous to the logical disjunctive operator (OR). The *daml:complementOf* is analogous to the logical negation operator (NOT).

Property element associates a property name with its definition. Property definition can be defined in terms of the following elements: *rdfs:subPropertyOf*, *rdfs:domain*, *rdfs:range*, *daml:samePropertyAs*, *daml:equivalentTo* and *daml:inverseOf*. The *rdfs:subPropertyOf* element defines the property definition in term of its subclass relationship to other properties. The *rdfs:domain* defines the property in term of what classes it can be applied to. For example, the domain of “shoe size” property could be specified as “human”. The *rdfs:range* defines the property that its allowable value must be an instance of the specified class. For example, the range of “hair” property could be specified as “color”. The *daml:samePropertyAs* and *daml:equivalentTo*, in the context of property definition, share the same meaning of stating the equivalence of one property to another property. However, it is suggested that the use of *daml:samePropertyAs* is preferred because the element is a subclass of *rdfs:subPropertyOf*, which make it partially be understood by the RDF Schema parser. The *daml:inverseOf* element defines the property in term of its inverse relationship with another property. For example, “is-eaten-by” property is an inverse of “eat” property.

There are two major types of properties: *daml:ObjectProperty*, and *daml:DatatypeProperty*. The *ObjectProperty* defines the property in term of its relationship to objects, while the *DatatypeProperty* defines the property in terms of its relationship to a data type value. There are three other kinds of properties which represent their special characteristics: *daml:TransitiveProperty*, *daml:UniqueProperty* and

daml:UnambiguousProperty. A property “P” is defined as *daml:TransitiveProperty* if $P(x,y)$ and $P(y,z)$ imply $P(x,z)$. The *daml:UniqueProperty* is a shorthand notation for the property that has its *maxCardinality* restriction of one. The property that is defined as *daml:UnambiguousProperty* is an inverse property of the property that is defined as *daml:UniqueProperty*.

DAML uses RDF syntax for making assertions about instances. For example, the statement “Adam is a person whose age is 13 and whose shoe size is 9.5” could be expressed as follows.

```
<Person rdf:ID="Adam">
  <rdfs:label>Adam</rdfs:label>
  <age><xsd:integer rdf:value="13"/></age>
  <shoesize><xsd:decimal rdf:value="9.5"/></shoesize>
</Person>
```

where the definition of the “Person” class and the “age” and “shoesize” properties are defined elsewhere. The *xsd* namespace states that the data type is defined by the XML Schema data type. DAML+OIL also provides *daml:sameIndividualAs* and *daml:differentIndividualFrom* elements for stating that two individuals are the same or different respectively.

In order to allow the expression of a collection of items, DAML+OIL provides the *daml:collection* element as an extension of *rdf:parseType*. The *daml:collection* is meant to be interpreted as an unordered list, aka a bag.

As of July, 2001, there are 160 ontologies written in DAML that are being submitted to the DAML ontology library (<http://www.daml.org/ontologies/>). The tools for DAML are also being created and provided. A complete list can be found in DAML Tools web page (<http://www.daml.org/tools/>). The tools are being developed in the following categories: DAML API, DAML Browser, DAML Validator, DAML Crawler, DAML Transformation, Inference Engine, DAML ontology editors, etc. The wide adoption of

DAML ontologies and DAML tools makes DAML an early favorite to provide semantic interoperability on the Web.

5.3 Ontology in distributed environment

Once an ontology is created, it must be shared between various parties for communicating domain knowledge. As the domain knowledge grows, the ontology must be able to evolve and be extended accordingly. However, due to the distributed and dynamic nature of the Web environment, any change in the ontology could cause inconsistency of domain knowledge between parties. The ontology revision and ontology integration schemes are crucial to the maintenance of Web ontology over time. Among the Web ontology language efforts, SHOE has explored this issue in the greatest detail (Heflin et al., 1999a; Heflin et al., 2000a).

SHOE considers ontology revision as a form of either *addition* or *removal* of SHOE's categories (classes), relations (attributes) and/or axioms (inference rules). The *modification* of an ontology element can be thought of a removal followed by an addition operation. SHOE has investigated how each type of revision will affect the HTML pages that are using the revised ontology and how it will affect the results of user queries. By mapping SHOE expressions into first-order logic, the effect of each type of revision can be analyzed.

The studies (Heflin et al., 1999a; Heflin et al., 2000a) show that revisions that add categories or relations will have no effect to the data sources that use the ontology since the revised ontology will subsume the original ontology. The vocabulary of the original ontology will become a part of the vocabulary of the revised ontology. The addition of categories or relations will not change the results of the user queries due to the monotonicity of the first-order logic. The addition of rules also has no effect to the existing data sources due to the monotonicity in first-order logic where new rules can not change the inferences drawn from the rules in the original ontology. The result indicates that one can safely add the categories (classes), relations (attributes) and rules to the ontology with no effect to existing data sources that use the ontology. This result,

however, does not hold for the removal of elements from the ontology. The same studies (Heflin et al., 2000a) show that removal of categories, relations and/or rules from the ontology could change the results of user queries, by giving fewer answers. This could result in the incompatibility between the revised ontology and the data sources that currently use the ontology.

In sharing domain knowledge, it is usually difficult to find a single ontology representing the domain knowledge that everyone can agree on. There could be many ontologies for the same domain. In order to achieve semantic interoperability, these different ontologies must be able to interoperate or integrate with each other. This requires one ontology to map its vocabulary to the vocabulary in the other ontologies. Wiederhold (Wiederhold, 1994) describes four types of differences that usually be found in the integration of ontologies:

- **Naming difference.** The situation where two different terms are used to describe the same concept.
- **Scope difference.** The situation where two ontologies share many things in common but one ontology is more specific than the other.
- **Encoding difference.** The situation where two ontologies use different measurement scales for the same value.
- **Context difference.** The situation where the same terms are used to describe different concepts.

Heflin et al. (Heflin et al., 2000a) gives the following examples of how these differences could be overcome using the mapping rules. The naming difference could be resolved using the two-way mapping rule. For example, in order to map two different names of the same concept, “Employee” vs. “StaffMember”, one could define the rule “BusOnt1.Employee(x) \Leftrightarrow BusOnt2.StaffMember(x)” to map the difference. The scope difference could be resolved using the one-way mapping rule. For example, if the concept of the term “FighterPilot” in one ontology is subsumed by the concept of the term “JetPilot” in a more general ontology, one could define the rule “AF_Ont.FighterPilot(x) \Rightarrow AF_Ont.JetPilot(x)” to map the difference. The encoding difference could be resolved using the two-way mapping rule, For example, the rule “CriticOnt1.Rating(x.”Good”) \Leftrightarrow

CriticOnt2.Rating(x,"3")" could be used to map two different scalings in two different ontologies. To resolve the context difference, it has been suggested that it be assumed that two terms from two different ontologies never represent the same meaning unless explicitly instructed (Heflin et al., 2000a). This assumption is also suggested by (Wiederhold, 1994).

Heflin et al. (Heflin et al., 2000a) suggested three approaches on how the ontology mapping could be implemented. This is illustrated in the following figure (reproduced from (Heflin et al., 2000a)).

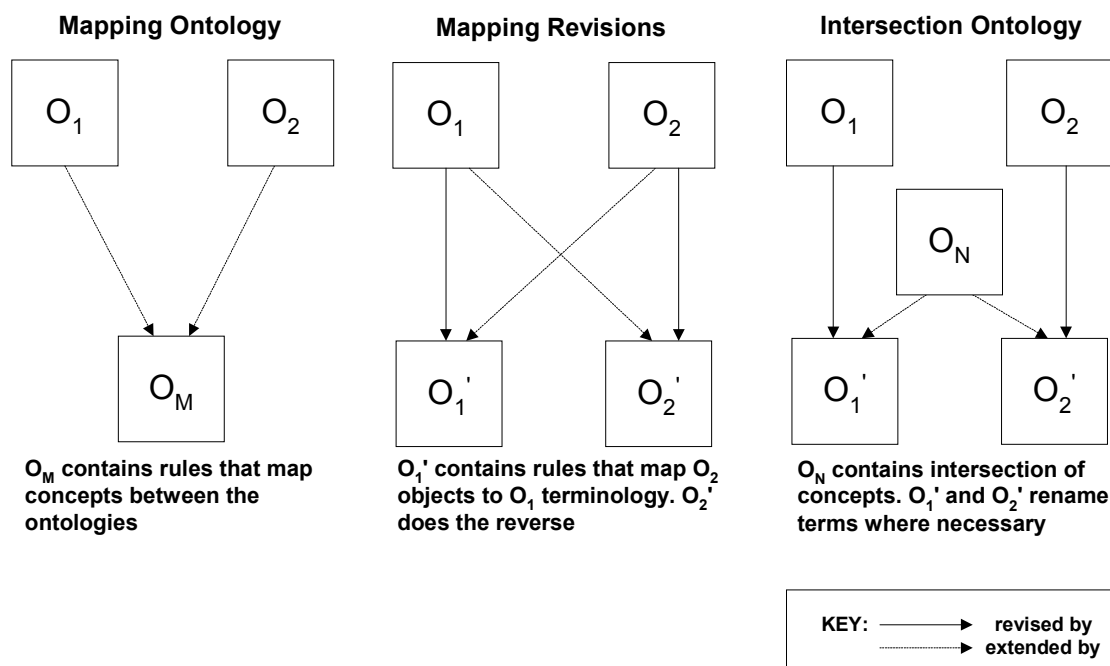


Figure 13: Ontology Integration methods

In the first approach, a new ontology is extended from the two ontologies that are being mapped to each other. The extended ontology includes all the rules defined for the mapping between the two ontologies. In the second approach, the original ontologies are revised to include the rules for mapping to other ontologies. These two approaches, however, both suffer from the complexity of mapping rules when the number of ontologies involved increases, i.e. $O(n^2)$. In the third approach, a new ontology is created from the intersection of the involved ontologies. The intersection ontology contains the general components that all ontologies share in common. This allows the revised

ontology to contain only the mapping rules to the intersection ontology. The number of necessary mapping will be reduced, i.e. $O(n)$.

5.4 Ontology in Electronic Commerce

Ontology is considered to play an important role in Electronic Commerce (EC), especially in the Business-to-Business (B2B) area. Fensel (Fensel, 2001b) suggests the three exchange models for B2B: $1:1$, $1:N$ and $N:M$. The $1:1$ model is where two companies exchange business transactions electronically. The $1:N$ model is where one large company exchanges business transactions with a number of other small companies. The $N:M$ model is where M companies exchange business transactions with N companies. The $N:M$ model needs an Internet-based marketplace to help in bringing both sides to meet. Fensel (Fensel, 2001b) suggests that the establishment of such marketplaces will require a means for representation and translation as well as a means for content descriptions or ontologies.

One attempt to provide a standard for electronic data interchange between different parties in business is the Electronic Data Interchange for Administration, Commerce and Transport (EDIFACT). However, the EDIFACT is a procedural and cumbersome standard, this makes its application difficult to implement or maintain. With the emergence of XML, the disadvantages in EDIFACT could be potentially overcome using the declarative markup language approach. As a result, the XML-based EDI standards have been set up to gain this benefit (e.g. XML/EDI – <http://www.xmledi.com/>).

Even though, XML provides common data interchange format, there is a need to define common terminologies for different business areas. This has resulted in the initiatives that define ontologies as a means for mediating e-commerce. This is to facilitate the integration of heterogeneous and distributed product descriptions (Fensel, 2001a). There are two kinds of such ontology approaches: horizontal-based ontology and vertical-based ontology. The horizontal-based standards try to cover all product areas. An example of such a standard is the Universal Standard Products and Services Classification (UN/SPSC - <http://www.unspsc.org/>). The UN/SPSC standard defines a concept hierarchy to

classify all product categories. The Universal Content Extended Classification (UCEC – <http://www.ucec.org/>) is an open standard for e-commerce catalogues classification and business exchanges that define the attributes of each category defined in UN/SPSC. The vertical standards try to focus within a domain of products. For example, RosettaNet (<http://www.rosettanet.org/>) provides a vertical-based ontology standard for electronic and computer hardware and software products. Other initiatives that define standards for terminologies in B2B data interchange include Microsoft' s BizTalk Framework (<http://www.biztalk.org/>), ebXML (<http://www.ebxml.org/>) and XML Common Business Library (xCBL -<http://www.xcbl.org/>). A review of some of these standards can be found in (Keong Ng, Peng Lim, & Yan, 2001).

Appendix

A. Description Logic

Motivation

The characteristics of computational logic include expressiveness, decidable and efficient reasoning and sound and complete inference procedures. There is a fundamental tradeoff between expressiveness and the computability of reasoning procedures (Levesque et al., 1985). First-order logic (FOL) has a high degree of expressiveness, which makes its inferences undecidable and inefficient. Description Logic (DL), aka Terminological Logic or Concept language focuses on computability while maintaining a considerable degree of expressiveness. The semantics of Description Logic is based on structured representation of KL-ONE, which is based on frames. Description logic can be considered as a unifying formalism for structured representation, such as frames, or can be considered as a *structured* fragment of FOL.

TBox and ABox

Description logic theory is divided in two parts: TBox and ABox. The “T” in TBox represents “terminological” and “A” in ABox represents “assertional”. TBox deals with the definitions of concepts, while ABox deals with assertions over facts. The clear separation between definition and assertion was introduced in KL-ONE and later is emphasized and formalized into two separate boxes in KRYPTON (Brachman et al., 1983), one of the KL-ONE successors.

TBox allows the establishment of taxonomies of structured terms. This allows the TBox to answer questions about relationships among terms. A definition of a concept in TBox is equivalent to the level of noun phrases in natural language, such as “a person with at least 3 children”. The reasoning service in TBox would allow it to infer that the concept is subsumed by the concept “a person with at least 1 child”. ABox allows the establishment of descriptive facts about the domains of interest. The reasoning service in ABox is able to answer questions related to these facts. Expression in ABox is equivalent

to the level of sentences in natural language, such as “Every person with at least 3 children owns a car”.

Brachman et al. illustrated the relationship of TBox and ABox as shown in Figure A1 (Brachman et al., 1983).

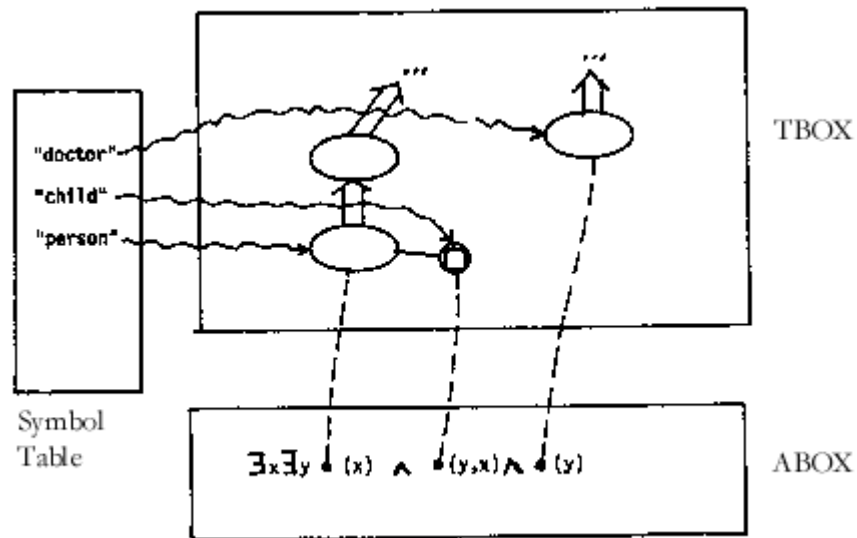


Figure 14: TBox, ABox and its relationship

Description Logic and computation

In order to achieve high computability, Brachman & Levesque reduced the degree of expressiveness in the FOL. The reduced version of FOL limits the form of expression to a frame description form. The logic of this language is known as \mathcal{FL}^- (Brachman et al., 1984). In \mathcal{FL}^- , *concepts*, denoted by the letters C and D, are built upon *primitive concepts*, denoted by the letter A, and *primitive roles*, denoted by the letter R. The grammar of \mathcal{FL}^- is shown in Table A1.

$C, D \rightarrow$	$A \mid$	(primitive concept)
	$C \sqcap D \mid$	(conjunction)
	$\forall R.C \mid$	(universal quantification)
	$\exists R$	(existential quantification)

Table 6: Syntax rule of \mathcal{FL} language

The only logical connective that \mathcal{FL} provides is conjunction of concepts. The value restriction of role is only allowed in universal quantification, not in existential quantification. The reasoning service for \mathcal{FL} determines subsumption relationships between concepts, and is decidable in polynomial time (Brachman et al., 1984). \mathcal{FL} is known as the simplest structural description logic.

While \mathcal{FL} provides good computability, its expressiveness is limited. This has led to many variations that add more expressiveness while maintaining computability. The \mathcal{AL} language (Schmidt-Schauß et al., 1991) adds more expressiveness to the \mathcal{FL} language. \mathcal{ALC} , a version of \mathcal{AL} language, is one of the simplest propositional DLs. The syntax rule of the \mathcal{ALC} language is shown in Table A2.

$C, D \rightarrow$	$A \mid$	(primitive concept)
	$\top \mid$	(top concept)
	$\perp \mid$	(bottom concept)
	$C \sqcap D \mid$	(conjunction)
	$C \sqcup D$	(disjunction)
	$\neg C$	(complement)
	$\forall R.C \mid$	(universal quantification)
	$\exists R.C$	(existential quantification)

Table 7: Syntax rule of \mathcal{ALC} language

\mathcal{ALC} added the expression of *negation* (\neg), *disjunction* (\sqcup), *value restriction in existential role quantifier* ($\exists R.C$), *empty set* (\perp) and *non-empty set* (\top) to the \mathcal{FL} -language. Schmidt-Schauß et al. (Schmidt-Schauß et al., 1991) shows that the calculation of satisfiability checking (see *Reasoning in DL*) in \mathcal{ALC} is possible in polynomial time. Hollunder et al. (Hollunder & Nutt, 1990) further shows that adding number restriction ($(\leq n R) \mid (\geq n R)$) and conjunction of roles to the \mathcal{AL} language (\mathcal{ALNR}) also gives a similar computability result.

Hollunder et al. (Hollunder et al., 1990) suggested that the problem of determining subsumption, in fact, could be reduced to the problem of checking satisfiability. The transformation could be shown as follows.

- C is subsumed by D if and only if $C \sqcap \neg D$ is not satisfiable
- C is satisfiable if and only if C is not subsumed by \perp

Thus determining subsumption gives the same complexity as computing satisfiability checking, which is possible in polynomial time.

Reasoning services in DL

Reasoning services in DL are clearly separated between TBox and ABox. Donini et al. (Donini, Lenzerini, Nardi, & Schaerf, 1996) provide a good summary of basic reasoning

techniques with ABox and TBox. There are several basic reasoning services provided by DL-based KR systems. These reasoning services can be classified into the following major categories (Schaerf, 1994):

- Concept satisfiability,
- Terminological subsumption,
- Knowledge base satisfiability,
- Instance Checking
- Hybrid subsumption.

The relationship between these reasoning services can be illustrated in Figure A2 (Schaerf, 1994). The line and dashed line means that one reasoning service can be reduced to another reasoning service. The crossing line represents complement relationship.

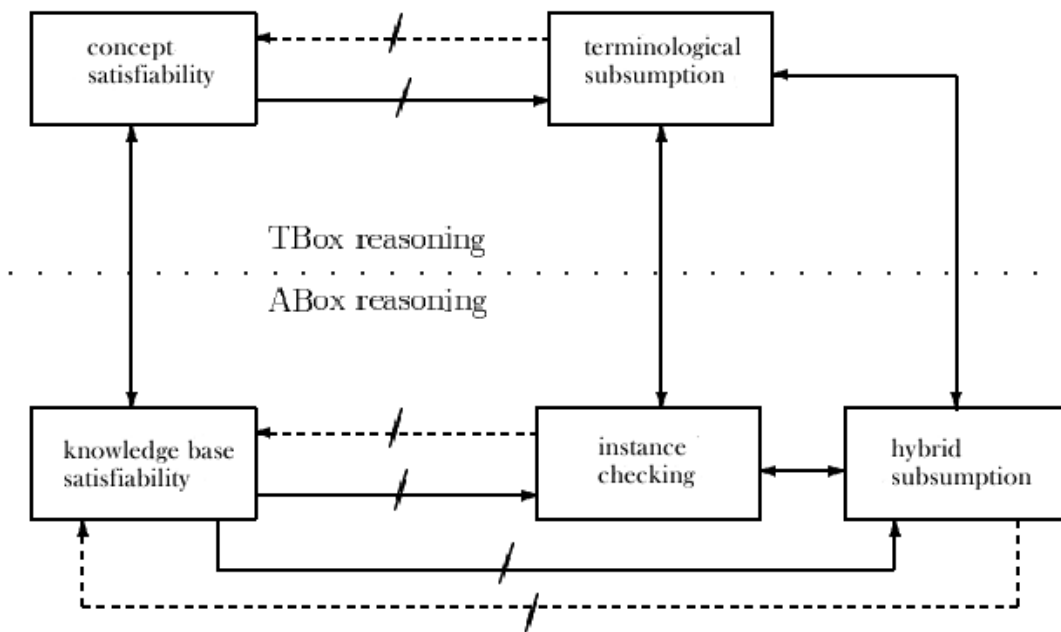


Figure 15: Reasoning services in Description Logics

Concept satisfiability is the problem of checking whether a concept is satisfiable. For example, if the *Student* concept is necessary to be *Person* concept, as denoted by “ $Student \leq Person$ ”, a concept of *Student* and *not Person*, as denoted by “ $Student \sqcap \neg Person$ ”, would not be satisfiable because there would be no member of *Student* concept that is not

a member of *Person* concept. **Terminological subsumption** is the problem of checking whether a concept is subsumed by another concept. For example, *Student* concept is subsumed by *Person* concept, as denoted by “ $Student \sqsubseteq Person$ ”, if every member of *Student* concept is a member of *Person* concept. **KB satisfiability** is the problem of checking whether the KB is satisfiable. For example, the axiom in the knowledge base stating that *Student* concept can be necessarily and sufficiently described by *not Person* concept, as denoted by “ $Student \doteq \neg Person$ ”, is not satisfiable because there will be no member of *Student* concept that is not belong to *Person* concept. **Instance Checking** is the problem of checking whether the assertion $C(a)$ is satisfiable according to the KB, where C is a concept and a is an individual. For example, reasoning for the assertion “ $Professor(John)$ ” assertion would return true if there exists an axiom in the KB (in the ABox) that *John* is an instance of *Professor* concept. The **hybrid subsumption** is the problem of checking whether the assertion about subsumption relationship between two concepts is satisfiable in the KB.

The computation of reasoning procedures in DL is mostly based on tableaux calculus. Tableaux calculus is a decision procedure solving the problem of satisfiability. It has been shown that other reasoning services in description logic can be reduced to the form of satisfiability checking, thus can be computed based-on tableaux calculus. Hollunder et al. (Hollunder et al., 1990) provide a good explanation of using tableaux calculus for checking the satisfiability of concepts. The paper also shows the complexity of the reasoning procedures in the \mathcal{AL} -family languages.

B. Nonmonotonic Logic and Reasoning

There are always cases where the expressions of exceptions and uncertainty are necessary. Classical logic is not expressive enough to describe exceptions or uncertain things. Classical logic does not allow one to draw a new conclusion that conflicts with existing axioms in the knowledge base. This makes the classical logic *monotonic*. The monotonic feature presents problems as few things in the real world are certain. As exceptions or changes are discovered, one would want to retract previously drawn conclusions from the knowledge base. In a classic example, if we know that all birds can fly ($\forall x. Bird(x) \Rightarrow Fly(x)$) and we know that Tweety is a bird ($Bird(Tweety)$). FOL would draw the conclusion that Tweety can fly ($Fly(Tweety)$). However, if it is later learned that Tweety is an ostrich, thus Tweety can not fly. FOL would not allow one to draw the conclusion that Tweety can not fly as it would conflict with the first conclusion in the KB. Nonmonotonic logics and reasoning are introduced to overcome this limitation. Nonmonotonic logics allow one to retract the previous conclusion if it is later known to be untrue.

One of the well-known uses of nonmonotonicity in KR is the closed-world assumption (CWA), introduced by Reiter (Reiter, 1978). The closed-world assumption, in short, states that anything that is unknown to the knowledge base is assumed to be untrue. The closed-world assumption eliminates the need to explicitly make statements about things that are believed to be false. This leads to nonmonotonicity in the knowledge base because once a new fact is obtained, the previous belief that it is false is no longer true. CWA is widely used in database application and logic programming.

There have been three major approaches to formalize nonmonotonic reasoning using logics. These approaches are **Circumscription**, **Default logic** and **Modal logic**. An overview of these approaches can be summarized as follows.

Circumscription

McCarthy introduced Circumscription as a form of nonmonotonic reasoning in 1980 (McCarthy, 1980). Circumscription is the only approach among the three that does not

extend the classical logic. This has an advantage that it does not introduce incompatibility to the classical logic. One of the motivations for circumscription is that humans usually jump to the conclusion that a thing works unless “something” prevents it (abnormality). This reasoning process minimizes the need for a large number of axioms that need to be stated about exceptions. Circumscription is a formalism of this human informal reasoning processes. It gives the minimal model of what we can infer from the given facts. Thus, when a new fact is introduced, the new circumscription would invalidate the previous minimal model. This characteristic is a form of nonmonotonic reasoning. McCarthy (McCarthy, 1986) demonstrates an example of circumscription in the following examples.

The bird flying example could be represented by the following set of axioms:

$$\begin{aligned} \forall x. \neg Ab(Aspect1(x)) \supset \neg Flies(x). \\ \forall x. Bird(x) \supset Ab(Aspect1(x)). \\ \forall x. Bird(x) \wedge \neg Ab(Aspect2(x)) \supset Flies(x). \\ \forall x. Ostrich(x) \supset Ab(Aspect2(x)). \\ \forall x. Ostrich(x) \wedge \neg Ab(Aspect3(x)) \supset \neg Flies(x). \end{aligned}$$

where *Ab* stands for abnormal.

The given axioms can be explained as follows. Unless *x* is abnormal under aspect 1, it can not fly. If *x* is a bird, it is abnormal under aspect 1. If *x* is a bird and not abnormal under aspect 2, it can fly. If *x* is an ostrich, it is abnormal under aspect 2. If *x* is an ostrich and not abnormal under aspect 3, it can not fly.

The circumscription over *Ab* and *Flies* predicates, represented as $A'(ab, flies)$, will return the minimal model. Thus the result would show that the minimal model of things that fly, represented as $flies'(x)$, would be equivalent to $bird(x) \wedge \neg ostrich(x)$. The minimal model of things that are abnormal, represented as $ab'(z)$, would be equivalent to $[\exists x. bird(x) \wedge z = aspect1(x)] \vee [\exists x. ostrich(x) \wedge z = aspect2(x)]$.

One of the difficulties with circumscription is that it makes use of predicates over predicates, which means the formula is now second-order logic. Little is known about inferences in second-order logic. This makes the calculation of circumscription hard to automate.

Default Logic

Another approach in nonmonotonic reasoning is Default Logic or Default Reasoning, introduced by Reiter (Reiter, 1980). Default Logic provides an inference rule for nonmonotonic reasoning. It could be considered as a variation of modus ponens. Default logic makes default assumptions when dealing with incomplete knowledge. These default assumptions can later be modified by subsequent evidence. This makes the default logic nonmonotonic.

The inference rule for default logic is shown by the following expression.

$$\frac{\alpha : \beta}{\gamma}$$

which reads “If α is true and it is consistent that β is true then assume γ is true”. For example, the bird flying example could be written as:

$$\frac{Bird(x) : Fly(x)}{Fly(x)}$$

Reiter gives the intuition of the above formula as follows: “If x is a bird, then in the absence of any information to the contrary, infer that x can fly”. Thus if Tweety is a bird and there is no evidence that it can not fly then we can infer that Tweety can fly by default. However if later it is discovered that Tweety is an ostrich and ostriches can not fly, default logic allows the retraction of the previous assumption that it can fly from the knowledge base.

Although the theory of Default logic is quite intuitive and stable, maintaining consistency in the systems that implement Default logic is a difficult task. Default logic can generate two default assumptions that conflict with each other and cause inconsistency in the knowledge base. This is known as the *multiple extensions* problem.

Modal Logic

Modal logic extends classical logic by introducing a new symbol to express uncertainty. McDermott and Doyle (McDermott et al., 1980) introduce the M operator to mean “possibly”. From the bird flying example, the modal operator can be used to represent Default reasoning as shown in the following logical sentence.

$$Bird(x) \wedge M Fly(x) \rightarrow Fly(x)$$

which reads “If x is a bird and it is consistent that x can fly, then assume that x can fly”.

The modal operator allows Default reasoning to be represented in logical sentences. Moore (Moore, 1985) argued that reasoning using modal operator is actually a form of *Autoepistemic reasoning* rather than *Default reasoning* as McDermott and Doyle has suggested. To make the semantics of the modal operator clearer, Moore introduced a new operator, L , which means “necessarily”. The semantic of the L operator can be translated to the M operator as $\neg L \neg p$ is equivalent to $M p$, where p is a logical sentence. Thus the above example can be represented in autoepistemic reasoning as the following logical sentence (Morgenstern, 1999).

$$L (Bird(x)) \wedge \neg L (\neg Fly(x)) \rightarrow Fly (x)$$

which reads “If I believe that x is a bird and I don’t believe that x can not fly, then (I will conclude that) x flies”. Thus rather than assuming that “it can fly because most of their type (bird) can” like in default reasoning, autoepistemic reasoning would say “I believe that it can fly because I don’t believe that it can not”.

C. An DAML+OIL (March 2001) Ontology Example¹

```
<!-- $Revision: 1.9 $ of $Date: 2001/05/03 16:38:38 $ -->

<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
  xmlns:xsd="http://www.w3.org/2000/10/XMLSchema#"
  xmlns:dex="http://www.daml.org/2001/03/daml+oil-ex#"
  xmlns:exd="http://www.daml.org/2001/03/daml+oil-ex-dt#"
  xmlns      ="http://www.daml.org/2001/03/daml+oil-ex#"
>

<daml:Ontology rdf:about="">
  <daml:versionInfo>$Id: daml+oil-ex.daml,v 1.9 2001/05/03 16:38:38
mdean Exp $</daml:versionInfo>
  <rdfs:comment>
    An example ontology, with data types taken from XML Schema
  </rdfs:comment>
  <daml:imports rdf:resource="http://www.daml.org/2001/03/daml+oil"/>
</daml:Ontology>

<daml:Class rdf:ID="Animal">
  <rdfs:label>Animal</rdfs:label>
  <rdfs:comment>
    This class of animals is illustrative of a number of ontological
idioms.
  </rdfs:comment>
</daml:Class>

<daml:Class rdf:ID="Male">
  <rdfs:subClassOf rdf:resource="#Animal"/>
</daml:Class>

<daml:Class rdf:ID="Female">
  <rdfs:subClassOf rdf:resource="#Animal"/>
  <daml:disjointWith rdf:resource="#Male"/>
</daml:Class>

<daml:Class rdf:ID="Man">
  <rdfs:subClassOf rdf:resource="#Person"/>
  <rdfs:subClassOf rdf:resource="#Male"/>
</daml:Class>

<daml:Class rdf:ID="Woman">
  <rdfs:subClassOf rdf:resource="#Person"/>
  <rdfs:subClassOf rdf:resource="#Female"/>
</daml:Class>

<daml:ObjectProperty rdf:ID="hasParent">
  <rdfs:domain rdf:resource="#Animal"/>
  <rdfs:range rdf:resource="#Animal"/>
```

¹ <http://www.daml.org/2001/03/daml+oil-ex.daml>

```

</daml:ObjectProperty>

<daml:ObjectProperty rdf:ID="hasFather">
  <rdfs:subPropertyOf rdf:resource="#hasParent"/>
  <rdfs:range rdf:resource="#Male"/>
</daml:ObjectProperty>

<daml:DatatypeProperty rdf:ID="shoesize">
  <rdfs:comment>
    shoesize is a DatatypeProperty whose range is xsd:decimal.
    shoesize is also a UniqueProperty (can only have one shoesize)
  </rdfs:comment>
  <rdfs:type
rdf:resource="http://www.daml.org/2001/03/daml+oil#UniqueProperty"/>
  <rdfs:range
rdf:resource="http://www.w3.org/2000/10/XMLSchema#decimal"/>
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="age">
  <rdfs:comment>
    age is a DatatypeProperty whose range is xsd:decimal.
    age is also a UniqueProperty (can only have one age)
  </rdfs:comment>
  <rdfs:type
rdf:resource="http://www.daml.org/2001/03/daml+oil#UniqueProperty"/>
  <rdfs:range
rdf:resource="http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger"/>
</daml:DatatypeProperty>

<daml:Class rdf:ID="Person">
  <rdfs:subClassOf rdf:resource="#Animal"/>
  <rdfs:subClassOf>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#hasParent"/>
      <daml:toClass rdf:resource="#Person"/>
    </daml:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <daml:Restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#hasFather"/>
    </daml:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#shoesize"/>
      <daml:minCardinality>1</daml:minCardinality>
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>

<daml:Class rdf:about="#Animal">
  <rdfs:comment>
    Animals have exactly two parents, ie:
    If x is an animal, then it has exactly 2 parents
    (but it is NOT the case that anything that has 2 parents is an
animal).
  </rdfs:comment>

```

```

    <rdfs:subClassOf>
      <daml:Restriction daml:cardinality="2">
        <daml:onProperty rdf:resource="#hasParent"/>
      </daml:Restriction>
    </rdfs:subClassOf>
  </daml:Class>

<daml:Class rdf:about="#Person">
  <rdfs:subClassOf>
    <daml:Restriction daml:maxCardinality="1">
      <daml:onProperty rdf:resource="#hasSpouse"/>
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>

<daml:Class rdf:about="#Person">
  <rdfs:subClassOf>
    <daml:Restriction daml:maxCardinalityQ="1">
      <daml:onProperty rdf:resource="#hasOccupation"/>
      <daml:hasClassQ rdf:resource="#FullTimeOccupation"/>
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>

<daml:UniqueProperty rdf:ID="hasMother">
  <rdfs:subPropertyOf rdf:resource="#hasParent"/>
  <rdfs:range rdf:resource="#Female"/>
</daml:UniqueProperty>

<daml:ObjectProperty rdf:ID="hasChild">
  <daml:inverseOf rdf:resource="#hasParent"/>
</daml:ObjectProperty>

<daml:TransitiveProperty rdf:ID="hasAncestor">
  <rdfs:label>hasAncestor</rdfs:label>
</daml:TransitiveProperty>

<daml:TransitiveProperty rdf:ID="descendant"/>

<daml:ObjectProperty rdf:ID="hasMom">
  <daml:samePropertyAs rdf:resource="#hasMother"/>
</daml:ObjectProperty>

<daml:Class rdf:ID="Car">
  <rdfs:comment>no car is a person</rdfs:comment>
  <rdfs:subClassOf>
    <daml:Class>
      <daml:complementOf rdf:resource="#Person"/>
    </daml:Class>
  </rdfs:subClassOf>
</daml:Class>

<!-- @@CAVEAT: daml:collection is an extension of RDF 1.0 syntax;
      don't expect existing tools to support it.
      See http://www.daml.org/2001/03/reference.html#collection for
      details.
-->

```

```

<daml:Class rdf:about="#Person">
  <rdfs:comment>every person is a man or a woman</rdfs:comment>
  <daml:disjointUnionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="#Man"/>
    <daml:Class rdf:about="#Woman"/>
  </daml:disjointUnionOf>
</daml:Class>

<daml:Class rdf:ID="TallMan">
  <daml:intersectionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="#TallThing"/>
    <daml:Class rdf:about="#Man"/>
  </daml:intersectionOf>
</daml:Class>

<daml:Class rdf:ID="MarriedPerson">
  <daml:intersectionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="#Person"/>
    <daml:Restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#hasSpouse"/>
    </daml:Restriction>
  </daml:intersectionOf>
</daml:Class>

<daml:Class rdf:ID="HumanBeing">
  <daml:sameClassAs rdf:resource="#Person"/>
</daml:Class>

<daml:Class rdf:ID="Adult">
  <daml:intersectionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="#Person"/>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#age"/>
      <daml:hasClass
rdf:resource="http://www.daml.org/2001/03/daml+oil-ex-dt#over17"/>
    </daml:Restriction>
  </daml:intersectionOf>
</daml:Class>

<daml:Class rdf:ID="Senior">
  <daml:intersectionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="#Person"/>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#age"/>
      <daml:hasClass
rdf:resource="http://www.daml.org/2001/03/daml+oil-ex-dt#over59"/>
    </daml:Restriction>
  </daml:intersectionOf>
</daml:Class>

<Person rdf:ID="Adam">
  <rdfs:label>Adam</rdfs:label>
  <rdfs:comment>Adam is a person.</rdfs:comment>
  <age><xsd:integer rdf:value="13"/></age>
  <shoesize><xsd:decimal rdf:value="9.5"/></shoesize>
</Person>

```

```

<daml:ObjectProperty rdf:ID="hasHeight">
  <rdfs:range rdf:resource="#Height"/>
</daml:ObjectProperty>

<daml:Class rdf:ID="Height">
  <daml:oneOf rdf:parseType="daml:collection">
    <Height rdf:ID="short"/>
    <Height rdf:ID="medium"/>
    <Height rdf:ID="tall"/>
  </daml:oneOf>
</daml:Class>

<!-- TallThing is EXACTLY the class of things whose hasHeight is tall -
->

<daml:Class rdf:ID="TallThing">
  <daml:sameClassAs>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#hasHeight"/>
      <daml:hasValue rdf:resource="#tall"/>
    </daml:Restriction>
  </daml:sameClassAs>
</daml:Class>

<daml:DatatypeProperty rdf:ID="shirtsize">
  <rdfs:comment>
    shirtsize is a DatatypeProperty whose range is clothingsize.
  </rdfs:comment>
  <rdf:type
rdf:resource="http://www.daml.org/2001/03/daml+oil#UniqueProperty"/>
  <rdfs:range rdf:resource="http://www.daml.org/2001/03/daml+oil-ex-
dt#clothingsize"/>
</daml:DatatypeProperty>

<rdfs:Class rdf:ID="BigFoot">
  <rdfs:comment>
    BigFoots (BigFeet?) are exactly those persons whose shosize is
over12.
  </rdfs:comment>
  <daml:intersectionOf rdf:parseType="daml:collection">
    <rdfs:Class rdf:about="#Person"/>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#shoesize"/>
      <daml:hasClass
rdf:resource="http://www.daml.org/2001/03/daml+oil-ex-dt#over12"/>
    </daml:Restriction>
  </daml:intersectionOf>
</rdfs:Class>

<Person rdf:ID="Ian">
  <rdfs:comment>
    Ian is an instance of Person. Ian has shoesize 14 and age 37. From
the range restrictions we know that these are of type xsd:decimal
and xsd:nonNegativeInteger respectively. Ian also has shirtsize 12,
the type of which is the union type clothingsize; the discriminating
type "string" has been specified, so the value is to be taken as the

```

```

    string "12" rather than the integer 12. We may be able to infer
    that Ian is an instance of BigFoot (because 14 is a valid value for
    xsd:over12) .
    </rdfs:comment>
    <shoesize>14</shoesize>
    <age>37</age>
    <shirtsize><xsd:string rdf:value="12"/></shirtsize>
</Person>

<Person rdf:ID="Peter">
  <rdfs:comment>
    Peter is an instance of Person. Peter has shoesize 9.5 and age 46.
    From the
    range restrictions we know that these are of type xsd:decimal and
    xsd:nonNegativeInteger respectively. Peter also has shirtsize 15, the
    type
    of which is the union type clothingsize; no discriminating type
    has been specified, so the value may be either a string or an
    integer.
    </rdfs:comment>
    <shoesize>9.5</shoesize>
    <age>46</age>
    <shirtsize>15</shirtsize>
</Person>

<daml:DatatypeProperty rdf:ID="associatedData">
  <rdfs:comment>
    associatedData is a DatatypeProperty without a range restriction.
  </rdfs:comment>
</daml:DatatypeProperty>

<daml:Class rdf:about="#Person">
  <rdfs:comment>
    Persons have at most 1 item of associatedData
  </rdfs:comment>
  <rdfs:subClassOf>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#associatedData"/>
      <daml:maxCardinality>1</daml:maxCardinality>
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>

<Person rdf:ID="Santa">
  <rdfs:comment>
    Santa is an instance of Person. Santa has two pieces of
    associatedData, one of which is the real number 3.14159 and the
    other of which is the string "3.14159". We may be able to infer a
    logical inconsistency (because Persons can have at most 1 item of
    associatedData, and a value cannot be both a string and a real
    number) .
  </rdfs:comment>
  <associatedData><xsd:real rdf:value="3.14159"/></associatedData>
  <associatedData><xsd:string rdf:value="3.14159"/></associatedData>
</Person>

</rdf:RDF>

```

D. World Wide Web Consortium (W3C) Recommendation Track¹

The W3C "Recommendation track" is the process that W3C follows to build consensus around a Web technology, both within W3C and in the Web community as a whole. W3C turns a technical report into a Recommendation by following this process. The labels that describe increasing levels of maturity and consensus along the Recommendation track are shown in the following diagram:

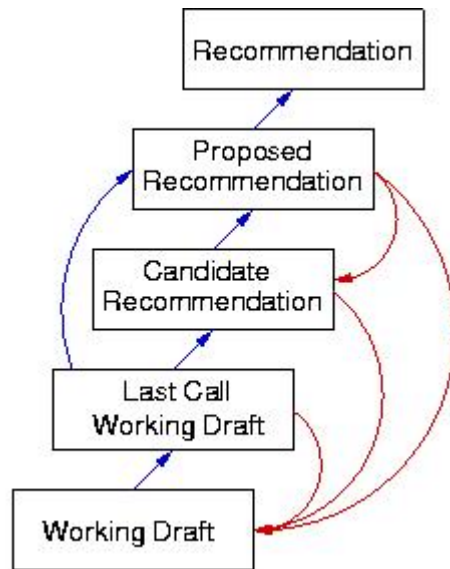


Figure 16: Possible transitions of the Recommendation track

¹ Excerpt from <http://www.w3.org/Consortium/Process-20010719/tr>

Reference List

Abiteboul, S., Buneman, P., & Suci, D. (2000). Data on the Web: From relations to semistructured data and XML. San Francisco, CA: Morgan Kaufmann.

Adler, S., Berglund, A., Caruso, J., Deach, S., Grosso, P., Gutentag, E., Milowski, A., Parnell, S., Richman, J., & Zilles, S. (2000). Extensible Stylesheet Language (XSL) Version 1.0. W3C Candidate Recommendation 21 November 2000 [On-line]. Available: <http://www.w3.org/TR/2000/CR-xsl-20001121/>

Berners-Lee, T. (2000a). Rules and Facts: Inference engines vs Web. Personal Note [On-line]. Available: <http://www.w3.org/DesignIssues/Rules.html>

Berners-Lee, T. (2000b). XML and the Web. Presentation at XML World 2000, Sep 6, 2000 [On-line]. Available: <http://www.w3.org/2000/Talks/0906-xmlweb-tbl/slide1-6.html>

Berners-Lee, T., Fielding, R., & Masinter, L. (1998). Uniform Resource Identifiers (URI): Generic Syntax. RFC 2396 [On-line]. Available: <http://www.ietf.org/rfc/rfc2396.txt?number=2396>

Berners-Lee, T. & Connolly, D. (1995). HyperText Markup Language Specification -- 2.0. Internet [On-line]. Available: <http://www.ics.uci.edu/pub/ietf/html/rfc1866.txt>

Berners-Lee, T., Fielding, R., & Frystyk, H. (1996). Hypertext Transfer Protocol - HTTP/1.0. RFC 1945 [On-line]. Available: <http://www.ietf.org/rfc/rfc1945.txt?number=1945>

Berners-Lee, T. (1998a). Evolvability. W3C [On-line]. Available: <http://www.w3.org/DesignIssues/Evolution.html>

Berners-Lee, T. (1998b). Semantic Web road map. Personal Note [On-line]. Available: <http://www.w3.org/DesignIssues/Semantic.html>

Berners-Lee, T. (1998c). Web architecture from 50,000 feet. Personal Note [On-line]. Available: <http://www.w3.org/DesignIssues/Architecture.html>

Berners-Lee, T. (2000c). Machines and the Web. In Weaving the Web (pp. 177-198). New York, NY: HarperCollins.

Berners-Lee, T. (2000d). Weaving the Web: The original design and ultimate destiny of the World Wide Web. New York, NY: HarperCollins.

Berners-Lee, T., Connolly, D., & Swick, R. R. (1999). Web architecture: Describing and exchanging data. W3C [On-line]. Available: <http://www.w3.org/1999/06/07-WebData.html>

Berners-Lee, T. & Connolly, D. (1993). Hypertext Markup Language (HTML): A representation of textual information and metaInformation for retrieval and interchange. W3C [On-line]. Available: <http://www.w3.org/MarkUp/draft-ietf-iiir-html-01.txt>

Berners-Lee, T. & Connolly, D. (1998). Web architecture: Extensible languages. W3C Note 10 Feb 1998 [On-line]. Available: <http://www.w3.org/TR/1998/NOTE-webarch-extlang-19980210>

Berners-Lee, T. & Swick, R. R. (1999). Frequently Asked Questions about RDF. W3C Technology and Society Domain [On-line]. Available: <http://www.w3.org/RDF/FAQ>

Biezunski, M. & Newcomb, S. R. (2001). XML Topic Maps: Finding aids for the Web. IEEE MultiMedia, 8, 104-108.

Biron, P. V. & Malhotra, A. (2000). XML Schema Part 2: Datatypes. W3C Note [On-line]. Available: <http://www.w3.org/TR/xmlschema-2/>

Borgida, A., Brachman, R. J., McGuinness, D. L., & Resnick, L. A. (1989). CLASSIC: A structural data model for objects. In Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data (pp. 59-67). Portland, Oregon.

Bos, B. (2001). Web style sheets. W3C Resource Page [On-line]. Available: <http://www.w3.org/Style/>

Bos, B., Wium Lie, H., Lilley, C., & Jacobs, I. (1998). Cascading Style Sheets, level 2. W3C Recommendation 12-May-1998 [On-line]. Available: <http://www.w3.org/TR/1998/REC-CSS2-19980512>

Bosak, J. (1996). DSSSL Online Application Profile. Drafted Specification [On-line]. Available: <http://www.ibiblio.org/pub/sun-info/standards/dsssl/dsssl0/do960816.htm>

Bosak, J. (1997). XML, Java, and the future of the Web. Online article [On-line]. Available: <http://www.ibiblio.org/bosak/xml/why/xmlapps.htm>

Brachman, R. J., Fikes, R. E., & Levesque, H. J. (1983). KRYPTON - A functional-approach to knowledge representation. Computer, 16, 67-73.

Brachman, R. J. & Levesque, H. J. (1984). The tractability of subsumption in frame-Based description languages. In Proceedings of AAAI-84. Austin, Texas.

Brachman, R. J. (1979). On the epistemological status of semantic networks. In N.V.Findler (Ed.), Associative Networks: Representation and Use of Knowledge by Computers (pp. 3-50). New York: Academic Press.

Brachman, R. J. & Schmolze, J. G. (1985). An overview of the KL-ONE knowledge representation system. Cognitive Science, 9, 171-216.

Bray, T. (2001). What is RDF? xml.com [On-line]. Available: <http://www.xml.com/pub/a/2001/01/24/rdf.html>

Bray, T., Hollander, D., & Layman, A. (1999). Namespaces in XML. W3C Recommendation 14 January 1999 [On-line]. Available: <http://www.w3.org/TR/1999/REC-xml-names-19990114/>

Bray, T., Paoli, J., Sperberg-McQueen, C. M., & Maler, E. (2000). Extensible Markup Language (XML) 1.0 (Second Edition). W3C Recommendation 6 October 2000 [On-line]. Available: <http://www.w3.org/TR/2000/REC-xml-20001006>

Brickley, D. & Guha, R. V. (2000). Resource Description Framework(RDF) Schema Specification 1.0. W3C Candidate Recommendation 27 March 2000 [On-line]. Available: <http://www.w3.org/TR/rdf-schema>

Brodie, M. L. (1992). The promise of distributed computing and the challenge of legacy information systems. In Proceedings of the IFIP WG2.6 Database Semantics Conference on Interoperable Database Systems (pp. 1-31). Lorne, Australia.

Broekstra, J., Klein, M., Decker, S., Fensel, D., & Horrocks, I. (2000). Adding formal semantics to the Web: Building on top of RDF Schema. In Proceedings of the Workshop "ECDL 2000 Workshop on the Semantic Web". Lisbon, Portugal.

Caplan, P. (1995). You call it corn, we call it syntax-independent metadata for document-like objects. *The Public-Access Computer Systems Review* 6 [On-line]. Available: <http://info.lib.uh.edu/pr/v6/n4/capl6n4.html>

Chepesiuk, R. (1999). Organizing the Internet: The "core" of the challenge. *American Libraries*, 30, 60-63.

Clark, J. (1997). Comparison of SGML and XML. World Wide Web Consortium Note 15-December-1997 [On-line]. Available: <http://www.w3.org/TR/NOTE-sgml-xml>

Clark, J. (1999). XSL Transformations (XSLT) Version 1.0. W3C Recommendation 16 November 1999 [On-line]. Available: <http://www.w3.org/TR/1999/REC-xslt-19991116>

Clark, J. & DeRose, S. (1999). XML Path Language (XPath) Version 1.0. W3C Recommendation 16 November 1999 [On-line]. Available: <http://www.w3.org/TR/1999/REC-xpath-19991116>

Collins, A. M. & Quillian, M. R. (1970). Facilitating retrieval from semantic memory: The effect of repeating part of an inference. In A.F.Sanders (Ed.), Acta Psychologica 33 Attention and Performance III (pp. 304-314). Amsterdam: North-Holland Publ.

Connolly, D. RDF Syntax: An XML Schema Approach. www-rdf-comments@w3.org Mailing List. www-rdf-interest@w3.org Mail Archives . 2001.
Ref Type: Internet Communication

Davis, R., Schrobe, H., & Szolovits, P. (1993). What is a Knowledge Representation? AI Magazine, 14, 17-33.

Dean, M. (2001). DARPA Agent Markup Language (DAML) update. Presentation at CoABS Principle Investigator Meeting [On-line]. Available: <http://www.daml.org/2001/02/coabs-daml/Overview.html>

Decker, S., Fensel, D., van Harmelen, F., Horrocks, I., Melnik, S., Klein, M., & Broekstra, J. (2000a). Knowledge Representation on the Web. In Proceedings of the 2000 International Workshop on Description Logics (DL2000). Aachen, Germany.

Decker, Stefan, Melnik, Sergey, Van Harmelen, Frank, Fensel, Dieter, Klein, Michel, Broekstra, Jeen, Erdmann, Michael, and Horrocks, Ian (2000b). The Semantic Web: The roles of XML and RDF. IEEE Internet Computing, 4, 63-73.

DeRose, S., Maler, E., & Daniel, R. Jr. (2001). XML Pointer Language (XPointer) Version 1.0. W3C Last Call Working Draft 8 January 2001 [On-line]. Available: <http://www.w3.org/TR/2001/WD-xptr-20010108>

DeRose, S., Maler, E., & Orchard, D. (2001). XML Linking Language (XLink) Version 1.0. W3C Recommendation 27 June 2001 [On-line]. Available: <http://www.w3.org/TR/2000/PR-xlink-20001220/>

DeRose, S. J. (2001). XML XLink Requirements Version 1.0. W3C Recommendation 27 June 2001 [On-line]. Available: <http://www.w3.org/TR/1999/NOTE-xlink-req-19990224>

Donini, F. M., Lenzerini, M., Nardi, D., & Schaerf, A. (1996). Reasoning in description logics. In G.Brewka (Ed.), Principles of Knowledge Representation and Reasoning (pp. 193-238). CLSI Publications.

Dublin Core Metadata Initiative (1999). Dublin Core Metadata Element Set, Version 1.1: Reference Description. Dublin Core Metadata Initiative Recommendation [On-line]. Available: <http://purl.org/dc/documents/rec-dces-19990702.htm>

Dublin Core Metadata Initiative (2000). Dublin Core Qualifiers. Dublin Core Metadata Initiative Recommendation [On-line]. Available: <http://purl.org/dc/documents/rec/dcmes-qualifiers-20000711.htm>

Fensel, D. (2001a). Ontologies and electronic commerce. IEEE Intelligent Systems, 16, 8.

Fensel, D. (2001b). Ontologies: Silver bullet for knowledge management and electronic commerce. Berlin: Springer-Verlag.

Fensel, D., van Harmelen, F., Horrocks, I., McGuinness, D. L., & Patel-Schneider, P. (2001). OIL: An ontology infrastructure for the Semantic Web. IEEE Intelligent Systems, 16, 38-45.

Fikes, R. & McGuinness, D. L. (2001). An axiomatic semantics for RDF, RDF-S, and DAML+OIL. DAML Home Page [On-line]. Available: <http://www.daml.org/2001/03/axiomatic-semantics-071601.html>

Freese, E. (2000). Topic Maps vs. RDF. Extreme Markup Languages 2000 [On-line]. Available: <http://ep.open.ac.uk/PubSys/resources/html/free0000.html>

Frege, G. (1892). On sense and reference. In P.Geach & M. Black (Eds.), Translations from the Philosophical Writings of Gottlob Frege Oxford: Blackwell.

Genesereth, M. R. & Nilsson, N. J. (1987). Logical foundations of artificial intelligence. Los Altos, CA: Morgan Kaufmann Publishers, Inc.

Ginsberg, M. L. (1987). Readings in nonmonotonic reasoning. Los Altos, CA: Morgan Kaufmann.

Gradmann, S. (1998). Cataloguing vs. Metadata: old wine in new bottles? In 64th IFLA General Conference August 16 - August 21, 1998.

Gruber, T. R. (1993). A translation approach to portable ontologies. Knowledge Acquisition, 5, 199-220.

Hayes, P. (1999). Knowledge Representation. MIT Encyclopedia of Cognitive Science [On-line]. Available: <http://cognet.mit.edu/MITECS/Entry/hayesp>

Heflin, J. & Hendler, J. (2000a). Dynamic ontologies on the Web. In Proceedings of 17th National Conference on Artificial Intelligence (AAAI-2000) (pp. 443-449). Menlo Park, CA: MIT-AAAI Press.

Heflin, J. & Hendler, J. (2000b). Semantic interoperability on the Web. In Proceedings of Extreme Markup Languages 2000 (pp. 111-120). Graphic Communications Association.

Heflin, J. & Hendler, J. (2001). A portrait of the Semantic Web in action. IEEE Intelligent Systems, 16, 54-59.

Heflin, J., Hendler, J., & Luke, S. (1999a). Coping with changing ontologies in a distributed environment. In AAAI-99 Workshop on Ontology Management.

Heflin, J., Hendler, J., & Luke, S. (1999b). SHOE: A knowledge representation language for Internet applications (Rep. No. Technical Report, CS-TR-4078 (UMIACS TR-99-71)). Dept. of Computer Science, University of Maryland at College Park.

Hendler, J. & McGuinness, D. L. (2000). The DARPA Agent Markup Language. IEEE Intelligent Systems, 15, 72-73.

Hollunder, B. & Nutt, W. (1990). Subsumption algorithms for concept languages (Rep. No. RR-90-04). Saarbruecken, Germany.

Horrocks, I. (2000). A denotational semantics for Standard-OIL and Instance-OIL. OIL Homepage [On-line]. Available:
<http://www.ontoknowledge.org/oil/down/semantics.pdf>

Horrocks, I., Fensel, D., Broekstra, J., Decker, S., Erdmann, M., Goble, C., van Harmelen, F., Klein, M., Stabb, S., Studer, R., & Motta, E. (2000). The Ontology Inference Layer OIL (Rep. No. Technical Report). University of Manchester / Vrije Universiteit Amsterdam.

Horrocks, I. & Sattler, U. (2001). Ontology reasoning for the semantic web. In Proceedings of the 17th Int.Joint Conf.on Artificial Intelligence (IJCAI'01).

Horrocks, I. & van Harmelen, F. (2000). DAML+OIL (December 2000). DAML Home Page [On-line]. Available: <http://www.daml.org/2000/12/daml+oil-index.html>

Horrocks, I., van Harmelen, F., & Patel-Schneider, P. (2001a). DAML+OIL (March 2001). DAML Home Page [On-line]. Available:
<http://www.daml.org/2001/03/daml+oil-index.html>

Horrocks, I., van Harmelen, F., & Patel-Schneider, P. (2001b). DAML+OIL (March 2001): A Datatype Extension to DAML+OIL (December 2000). DAML Home Page [On-line]. Available: <http://www.daml.org/2001/03/differences-daml+oil.html>

Institute of Electrical and Electronics Engineers (1990). IEEE standard computer dictionary: A compilation of IEEE standard computer glossaries. New York, NY.

International Federation of Library Associations (2000). Digital libraries: Metadata resources. IFLANET Electronic Collections [On-line]. Available:
<http://www.ifla.org/II/metadata.htm>

International Organization for Standardization (1986). ISO 8879: Information processing---Text and office systems---Standard Generalized Markup Language (SGML). ISO Standard [On-line]. Available: <http://www.iso.ch/>

International Organization for Standardization (1996). ISO/IEC 10179: Document Style Semantics and Specification Language (DSSSL). ISO Standard [On-line]. Available: <http://www.iso.ch/cate/d18196.html>

International Organization for Standardization (1997). HyTime: ISO 10744 Hypermedia/Time-based Structuring Language - 2nd edition. ISO Standard [On-line]. Available: <http://www.ornl.gov/sgml/wg8/docs/n1920/html/n1920.html>

International Organization for Standardization (1999). ISO/IEC 13250: Topic Maps. ISO Standard [On-line]. Available: <http://www.y12.doe.gov/sgml/sc34/document/0129.pdf>

Jelliffe, R. (2000). W3C XML Schema Datatypes Reference. xml.com [On-line]. Available: <http://www.xml.com/pub/a/2000/11/29/schemas/dateref.html>

Jon.Bosak@eng.Sun.COM (Jon Bosak). Re: XSL and DSSSL-O. dsssl@lists.mulberrytech.com. the DSSSL users' mailing list . 9-13-1997.
Ref Type: Internet Communication

Kashyap, V. & Sheth, A. (1994). Semantics-based information brokering. In Proceedings of the 3rd International Conference on Information and Knowledge Systems (pp. 363-370).

Keong Ng, W., Peng Lim, E., & Yan, G. (2001). Standardization and integration in business-to-business electronic commerce. IEEE Intelligent Systems, 16, 12-14.

Lacher, M. S. & Decker, S. (2001). On the integration of topic maps and RDF data. In Proceedings of the Semantic Web Working Symposium. Stanford University, California, USA.

Lagoze, C. (1996). The Warwick Framework: A container architecture for diverse sets of metadata. D-Lib Magazine, July/August 1996 [On-line]. Available: <http://www.dlib.org/dlib/july96/lagoze/07lagoze.html>

Lagoze, C., Lynch, C., & Daniel, R. Jr. (1996). The Warwick Framework: A container architecture for aggregating sets of metadata. Cornell Computer Science Technical Report TR96-1593 [On-line]. Available: <http://cs-tr.cs.cornell.edu/Dienst/UI/2.0/Describe/ncstrl.cornell/TR96-1593>

Lassila , O. (1997). RDF metadata and agent architectures. Workshop on Compositional Software Architectures, 6-8 January 1998 [On-line]. Available: <http://www.objs.com/workshops/ws9801/papers/paper056.html>

Lassila , Ora (1998). Web metadata: a matter of semantics. IEEE Internet Computing, 2, 30-37.

Lassila , O. & McGuinness, D. L. (2001). The role of frame-based representation on the Semantic Web (Rep. No. KSL Tech Report Number KSL-01-02).

Lassila , O. & Swick , R. R. (1999). Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation 22 February 1999 [On-line]. Available: <http://www.w3.org/TR/REC-rdf-syntax/>

Levesque, H. J. & Brachman, R. J. (1985). A fundamental tradeoff in knowledge representation and reasoning (revised version). In R.J.Brachman & H. J. Levesque (Eds.), Readings in Knowledge Representation (pp. 41-70). San Mateo, CA: Morgan Kaufmann.

Luke, S. & Heflin, J. (2000). SHOE 1.01 Specification. SHOE Project [On-line]. Available: <http://www.cs.umd.edu/projects/plus/SHOE/spec.html>

Luke, S., Specter, L., & Rager, D. (1996). Ontology-based knowledge discovery on the World Wide Web. In A. Franz & H. Kitano (Eds.), Working Notes of the Workshop on Internet-Based Information Systems at the 13th National Conference on Artificial Intelligence (AAAI96) (pp. 96-102). AAAI Press.

Malhotra, A. & Maloney, M. (1999). XML Schema Requirements. W3C Note 15 February 1999 [On-line]. Available: <http://www.w3.org/TR/NOTE-xml-schema-req>

McCarthy, J. (1960). Programs with common sense. In Proceedings of the Teddington Conference on the Mechanization of Thought Processes. H. M. Stationery Office.

McCarthy, J. (1980). Circumscription - a form of nonmonotonic reasoning. Artificial Intelligence, 13, 27-39.

McCarthy, J. (1986). Applications of Circumscription to formalizing common sense knowledge. Artificial Intelligence, 28, 89-116.

McDermott, D. & Doyle, J. (1980). Non-monotonic logic I. Artificial Intelligence, 13, 41-72.

McGuinness, D. L. (2001). Ontologies come of age. In D.Fensel, J. Hendler, H. Lieberman, & W. Wahlster (Eds.), To appear in The Semantic Web: Why, What, and How MIT Press.

McGuinness, D. L., Resnick, L. A., & Isbell, C. (1995). Description logic in practice: A CLASSIC application. In Proceedings of the 1995 International Joint Conference on Artificial Intelligence.

McGuinness, D. L. & Wright, J. R. (1998). An industrial strength Description Logic-based configurator platform. IEEE Intelligent Systems, 13, 69-77.

Megginson, D. (2000). SAX 2.0: The Simple API for XML. Megginson Technologies [On-line]. Available: <http://www.megginson.com/SAX/index.html>

Melnik, S. & Decker, S. (2000). A layered approach to information modeling and interoperability on the Web. In Proceedings of the ECDL'00 Workshop on the Semantic Web. Lisbon, Portugal.

Melnik, S. e. al. (1999). Generic Interoperability Framework. Working Paper [On-line]. Available: <http://www-diglib.stanford.edu/diglib/ginf/WD/ginf-overview/>

Merriam-Webster Collegiate® Dictionary (2001). Merriam-Webster Collegiate® Dictionary. Merriam-Webster Collegiate® Dictionary [On-line]. Available: <http://www.m-w.com/>

Milstead , J. & Feldman, S. (1999). Metadata: Cataloging by any other name ... ONLINE, January 1999 [On-line]. Available: <http://www.onlineinc.com/onlinemag/OL1999/milstead1.html>

Minker, J. (1993). An overview of nonmonotonic reasoning and logic programming. Journal of Logic Programming, Special Issue, 17.

Minsky, M. (1975). A framework for representing knowledge. In P.Winston (Ed.), The Psychology of Computer Vision (pp. 211-277). New York: McGraw-Hill.

Moore, R. C. (1985). Semantical considerations on nonmonotonic logic. Artificial Intelligence, 25, 75-94.

Morgenstern, L. (1999). Nonmonotonic Logics. MIT Encyclopedia of Cognitive Science [On-line]. Available: <http://cognet.mit.edu/MITECS/Entry/morgenstern>

Neches, R., Fikes, R., Finin, T., Gruber, T., Patil, R., Senator, T., & Swartout, W. R. (1991). Enabling technology for knowledge sharing. AI Magazine, 12, 36-56.

Noy, N. F., Sintek, M., Decker, S., Crubezy, M., Ferguson, R. W., & Musen, M. A. (2001). Creating Semantic Web contents with Protege-2000. IEEE Intelligent Systems, 16, 60-71.

Pepper, S. (2000). The TAO of Topic Maps: Finding the way in the age of infoglut. XML Europe 2000 [On-line]. Available: <http://www.gca.org/papers/xmleurope2000/papers/s11-01.html>

Quillian, M. R. (1967). Word concepts: a theory and simulation of some basic semantic capabilities. Behavioral Science 410-430.

Raggett, D. (1997). HTML 3.2 Reference Specification. [On-line]. Available: <http://www.w3.org/TR/REC-html32>

Raggett, D., Hors, A. L., & Jacobs, I. (1998). HTML 4.0 Specification. [On-line]. Available: <http://www.w3.org/TR/1998/REC-html40-19980424/>

Reiter, R. (1978). On closed world data bases. In H.Gallaire & J. Minker (Eds.), Logic and databases (pp. 119-140). New York: Plenum.

Reiter, R. (1980). A logic for default reasoning. Artificial Intelligence, 13, 132.

Robinson, J. A. (1965). A machine-oriented logic based on the resolution principle. Journal of the Association for Computing Machinery, 12, 23-41.

Russell, S. & Norvig, P. (1995a). Artificial Intelligence: A modern approach. New Jersey: Prentice-Hall, Inc.

Russell, S. & Norvig, P. (1995b). Probabilistic reasoning systems. In Artificial Intelligence: A Modern Approach New Jersey: Prentice-Hall, Inc.

Schaerf, A. (1994). Reasoning with individuals in concept languages. Data and Knowledge Engineering, 13, 141-176.

Schmidt-Schauß, M. & Smolka, G. (1991). Attributive concept descriptions with complements. Artificial Intelligence Journal, 48, 1-26.

Schwarz, P. M. & Roth, M. T. (1997). Don't scrap it, wrap it! A wrapper architecture for legacy data sources. In Proceedings of the 23rd VLDB Conference (pp. 266-275). Athens, Greece.

SGML Users' Group (1990). A brief history of the development of SGML. SGML Users' Group [On-line]. Available: <http://www.sgmlsource.com/history/sgmlhist.htm>

Shafir, E. (1999). Probabilistic reasoning. MIT Encyclopedia of Cognitive Science [On-line]. Available: <http://cognet.mit.edu/MITECS/Entry/shafir>

Sheth, A. (1999). Changing focus on interoperability in information systems: from system, syntax, structure to semantics. In Norwell, Massachusetts: Kluwer Academic Publishers.

Sowa, J. F. (2000). Knowledge Representation: Logical, philosophical, and computational foundations. Pacific Grove, CA: Brooks/Cole.

Sperberg-McQueen, C. M. & Burnard, L. (2001). A gentle introduction to SGML. TEI Guidelines for Electronic Text Encoding and Interchange [On-line]. Available: <http://www-tei.uic.edu/orgs/tei/sgml/teip3sg/index.html>

Spring, M. B. (1996). Reference model for data interchange standards. IEEE Computer, 29, 87-88.

Stein, L. A., Connolly, D., & McGuinness, D. L. (2000). DAML-ONT initial release. DAML Home Page [On-line]. Available: <http://www.daml.org/2000/10/daml-ont.html>

Swartout, B., Patil, R., Knight, K., & Russ, T. (1996). Toward distributed use of large-scale ontologies. In Proceedings of the Tenth Knowledge Acquisition for Knowledge -based Systems Workshop.

Swick, R. R. (1999). RDF: weaving the web of discovery. netWorker, 3, 21-25.

Thompson, H. S., Maloney, M., & Mendelsohn, N. (2000). XML Schema Part 1: Structures. W3C Note [On-line]. Available: <http://www.w3.org/TR/xmlschema-1/>

Topicmaps.Org Authoring Group (2000). Minutes of Topicmaps.Org Authoring Group (AG) [13-15 Oct 2000]. doctypes.org [On-line]. Available: <http://www.doctypes.org/xtm/org/minutes20001013.html>

Topicmaps.Org Authoring Group (2001). XML Topic Maps (XTM) 1.0, Aug 2001. topicmaps.org Specification [On-line]. Available: <http://www.topicmaps.org/xtm/1.0/>

van Harmelen, F. (2000). OIL ontology inference and interchange. Presentation at 14th European Conference on Artificial Intelligence ECAI-00 [On-line]. Available: <http://www.ontoknowledge.org/oil/presentations/index.shtml#ECAI00>

van Harmelen, F., Patel-Schneider, P., & Horrocks, I. (2001a). A model-theoretic semantics for DAML+OIL (March 2001). DAML Home Page [On-line]. Available: <http://www.daml.org/2001/03/model-theoretic-semantic.html>

van Harmelen, F., Patel-Schneider, P., & Horrocks, I. (2001b). Annotated DAML+OIL (March 2001) ontology markup. DAML Home Page [On-line]. Available: <http://www.daml.org/2001/03/daml+oil-walkthru.html>

van Harmelen, F., Patel-Schneider, P., & Horrocks, I. (2001c). Reference description of the DAML+OIL (March 2001) ontology markup language. DAML Home Page [On-line]. Available: <http://www.daml.org/2001/03/reference.html>

Vckovski, A. (1999). Interoperability and spatial information theory. In Norwell, Massachusetts: Kluwer Academic Publishers.

W3C (2000). Extensible Markup Language (XML) activity. W3C Architecture Domain Activity Statement [On-line]. Available: <http://www.w3.org/XML/Activity>

W3C (2001). Semantic Web activity statement. W3C Technology & Society Domain Activity Statement [On-line]. Available: <http://www.w3.org/2001/sw/Activity/>

W3C DOM Working Group (2001). Document Object Model (DOM). W3C Architecture Domain [On-line]. Available: <http://www.w3.org/DOM/>

Weibel, S. (1995). Metadata: The foundations of resource description. D-Lib Magazine, July 1995 [On-line]. Available: <http://www.dlib.org/dlib/July95/07weibel.html>

Wiederhold, G. (1992). Mediators in the architecture of future information systems. IEEE Computer, 25.

Wiederhold, G. (1994). An algebra for ontology composition. In Proceedings of 1994 Monterey Workshop on Formal Methods (pp. 56-62). U.S. Naval Postgraduate School.

Wium Lie, H. & Bos, B. (1999). Cascading Style Sheets, level 1. W3C Recommendation 17 Dec 1996, revised 11 Jan 1999 [On-line]. Available: <http://www.w3.org/TR/REC-CSS1>

Woods, W. A. (1975). What's in a link: foundations of semantic networks. In D.G.Bobrow & A. M. Collins (Eds.), Representation and Understanding: Studies in Cognitive Science (pp. 35-82). New York: Academic Press.

Woods, W. A. & Schmolze, J. G. (1992). The KL-ONE family. Computers & Mathematics with Applications, 23, 133-177.

Wool, G. (1998). A meditation on metadata. In Wayne Jones (Ed.), E-Serials: Publishers, Libraries, Users, and Standards (pp. 167-178). The Haworth Press, Inc.

Wright, J. R., Weixelbaum, E. S., Brown, K., Vesonder, G. T., Palmer, S. R., Berman, J. I., & Moore, H. H. (1993). A knowledge-based configurator that supports sales, engineering, and manufacturing at AT&T network systems. In Proceedings of the Innovative Applications of Artificial Intelligence Conference (pp. 183-193).

Acknowledgement

The author would like to thank Dr. Michael Spring for his advice and effort that result in the quality of this paper. The author would like to thank Dr. Roger Flynn for his careful reading to this paper. The author would like to thank the Ph.D. committee: Dr. Paul Munro, Dr. Michael Spring, Dr. Stephen Hirtle, Dr. Douglas Metzler, Dr. Hassan Karimi and Dr. Peter Brusilovsky, for their comments and suggestions.