USG
University of Strathclyde, Glasgow

Diogene

Survey on Methods, Standards and Tools for LO Knowledge Representation

Version 1.3

Revision History

Date	Version	Description	Author
27/05/2002	1.0	Initial version.	CRMPA
24/06/2002	1.1	Intermediate version.	Ruth Wilson and Robert Villa, USG
26/07/2002	1.2	Intermediate version. Sections on OWL, OKI and LIA added.	Ruth Wilson and Robert Villa, USG
09/09/2002	1.3	Final version. Changes to pagination.	Ruth Wilson , USG

Table of Contents

Survey on Methods, Standards and Tools for LO Knowledge Representation

1. Introduction

1.1 Purpose

The DIOGENE Project aims to design, implement and evaluate a training Web brokering environment geared toward ICT professionals and able to cover the whole lifecycle of ICT vocational training inside and outside enterprises' boundaries. It will design and implement a self-learning Web environment able to support the individual from the definition of objectives to the assessment of the results through the construction of custom self-adaptive courses. The system will be accessible through the Web and will take the form of a training portal.

An important feature of DIOGENE will be the possibility to use, in the dynamic course composition process, high quality contents (generally not for free) coming from registered content providers' servers and freeware contents drawn directly from the Web. This will be realised by means of different subsystems suitably interoperating and cooperating.

In relation to such subsystems, DIOGENE will address the following objectives:

1. To arrange high quality contents covering a set of ICT related topics and to make them available to the main system. This will include the provision of a methodology relying on emerging standards and a software tool to organise and publish high quality learning objects on content provider sites and make them available to DIOGENE as Web services. This will be achieved in a series of steps:

- A knowledge representation methodology able to describe learning objects in a machine-understandable way will be defined. An analysis of current standards for ontology description (e.g. SHOE, OIL, etc.) and learning object metadata description (e.g. IMS, LOM, etc.) will be carried out in order to let the proposed methodology be compliant with a subset of them.
- A Main Ontology covering the ICT domain will be created by experts in the field to individuate main concepts and how they are interrelated. This ontology will be exploited by the system for course tailoring and knowledge assessment. In order to manipulate the ontology, an Ontology Manager tool will be provided.
- Registered content providers (initially consortium partners) will arrange and make available courses covering a subset of ICT topics. The already existing course material will be indexed following the chosen methodology with a meta-tagger tool that will be provided for this purpose. Indexed contents will remain on content provider servers but will be available to DIOGENE as Web services. In this way the system will be able to construct custom courses by assembling learning contents from different providers. The system will be able to perform e-commerce transactions in order to fulfil rights.

2. To create an Information Retrieval subsystem to draw freeware contents from the Web and to make them available as learning objects. This will be achieved by exploiting the power of the Semantic Web, as announced by the W3C. Thanks to the annotation of Web pages with XML tags, online digital contents can, in fact, be described formally in a machine intelligible fashion. Moreover, thanks to the use of standard formats for the description of ontologies, it is possible to establish common dictionaries of interrelated concepts that will constitute the semantic substratum on which an artificial intelligence engine can make inferences.

- In order to retrieve useful content to be added to the learning environment while bypassing compatibility problems between different ontological representations of the same domain, DIOGENE will use a mixed approach constituting both keywords and ontologies.
- The Information Retrieval subsystem will be extended with the possibility of also drawing freeware

material from non-Semantic Web sources, relying on a classical keyword approach in order that the material can be exploited immediately with the actual Web structure. Material found in this way will have a limited pedagogical value, but it can be used as an additional source during training sessions.

The purpose of this document is to review current methodologies, standards and tools in the field of knowledge representation, with particular attention to learning object representation. Main standards related to learning object description (e.g. Dublin Core, IMS and LOM) and to ontology representation (RDF, OIL, SHOE, etc.) will be described and compared, and the most promising of these will be adopted in the DIOGENE knowledge model.

1.2 Scope

This document is related to another survey being produced by DIOGENE, a *Survey on Method and Standards for Student Modelling*. Together, the two surveys will inform the project's knowledge model.

1.3 Definitions, Acronyms, and Abbreviations

See Glossary [1].

1.4 References

[1] CRMPA DIOGENE Glossary, 2002.

[2] IEEE P1484.12 Learning Object Metadata Working Group. *LOM Approved Working Draft* 4. (http://ltsc.ieee.org/doc/wg12/LOM_WD4.PDF).

[3] IMS LIP Information Model Specification, 2001, (http://www.imsproject.com/profiles/lipinfo01.html).

[4] IMS Learning Resource Meta-Data Best Practice and Implementation Guide, 2001, version 1.2.1, Final Specification (http://www.imsproject.org/specifications.html).

[5] IMS Question and Test Interoperability: an overview, 2001. Public draft specification version 1.2 (http://www.imsproject.org/specifications.html).

[6] L. Aiello and A. Micarelli, 1990. SEDAF: An intelligent educational system for mathematics. *Applied Artificial Intelligence- An International Journal* 4 (15), pp. 15-36.

[7] J. R. Anderson, C. F. Boyle and G. Yost, 1985. The geometric Tutor. *Proceedings of the IJCAI,. Los Angeles*, CA.

[8] J. R. Anderson and B. J. Raiser, 1985. The LISP Tutor. *Byte 10 (4)*.

[9] T. Berners-Lee. *Semantic Web Road map*. World Wide Web Consortium Archive. http://www.w3.org/DesignIssues/Semantic.html.

[10] T. Berners-Lee. *Interpretation and Semantics on the Semantic Web*. World Wide Web Consortium Archive (http://www.w3.org/DesignIssues/Interpretation.html).

[11] D. G. Bodrow and T. Winograd, 1977. An overview of KRL, a knowledge representation language. Cognitive Science 1, pp. 3-46.

[12] B. Bos, 1999. *XML in 10 points*. World Wide Web Consortium Archive. (http://www.w3.org/XML/1999/XML-in-10-points.html).

[13] R. J. Brachman and H. J. Levesque, 1984. The tractability of subsumption in frame-based description languages. In *Proceedings of AAAI-84, 4th Conference of the American Association of Artificial Intelligence*, pp. 34-37.

[14] R. J. Brachman and H. J. Levesque editors, 1985. *Readings in knowledge representation*. Morgan Kaufmann, San Matteo CA.

[15] M. Cialdea, A. Micarelli, D. Nardi, J. C. Spohrer and L. Aiello, 1990. *Meta-level reasoning for diagnosis in intelligent tutoring systems*. Technical Report, Dipartimento di Informatica e Sistemistica, Universita' di Roma "La Sapienza".

[16] S. Cranefield, 2001. *UML and the Semantic Web*.

[17] S. Cranefield and M. Purvis, 1999. UML as an Ontology Modelling language. Proceedings of the Workshop on Intelligent Information Integration, 16th IJCAI.

[18] F. Donini, M. Lenzerini, D. Nardi and A. Schaerf, 1992. From subsumption to instance checking. Technical Report, Dipartimento di Informatica e Sistemistica, Universita' di Roma "La Sapienza".

[19] F. Donini, M. Lenzerini, D. Nardi and A. Schaerf, 1996. Reasoning in Description Logics. In G. Brewka editor, *Principles of Knowledge Representation and Reasoning*, Studies in Logic, Language and Information, pp. 193-238. CLSI Publications.

[20] D. Dubois and H. Prade, 1980. *Fuzzy sets and systems. Theory and Applications*. Academic Press.

[21] D. Fensel, F. van Harmelen, I. Horrocks: *OIL: A Standard Proposal for the Semantic Web*. Deliverable 0 in the European IST project OnToKnowledge. (http://www.ontoknowledge.org/oil/downl/otk.del02.pdf).

[22] Genesereth. *Logical Foundation of Artificial intelligence*.

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

[24] J. R. Hartley and D. H. Sleeman, 1973. Towards intelligent teaching systems. *International Journal of Man-Machine Studies*, 5, pp. 215-236.

[25] J. Heflin, J. Hendler, S. Luke. *SHOE: A Knowledge Representation Language for Internet Applications*. Technical Report CS-TR-4078 (UMIACS TR-99-71). 1999. (http://www.cs.umd.edu/projects/plus/SHOE/pubs/techrpt99.pdf).

[26] R. Iannella, 1998. An idiot's guide to the Resource Description Framework. *The new Review of Information Networking* 4.

[27] A. N. Kolomogorov, 1956. *Foundations of the theory of probability*. Chelsea, NY.

[28] H. J. Levesque and R. J. Brachman, 1987. Expressiveness and tractability in knowledge representation and reasoning. *Computational Intelligence* 3, pp. 78-93.

[29] J. W. Lloyd, 1986. *Foundations of Logic Programming*. Springer Verlag, Berlin.

[30] G. F. Luger and W. A. Stubblefield, 1998. *Artificial Intelligence. Structures and strategies for complex problem solvers*. Addison-Wesley Longman.

[31] A. Micarelli, F. Mungo, F. S. Nucci and L. Aiello, 1991. SAMPLE: An Intelligent Educational System for Eletrical Circuits. Journal of Artificial Intelligence in Education, 2 (3), pp. 83-99.

[32] E. Miller, 1998. An introduction to the Resource Description Framework (http://www.dlib.org/dilib/may98/miller/05miller.html).

[33] M. Minsky, 1975. A framework for representing knowledge. I P. J. Winston editor, *The Psychology of Computer Vision*, McGraw-Hill NY pp. 211-277.

[34] J. F. Nicaud and M. Vivet, 1988.Les tuteurs intelligents: Realisations et tendance de recherches. In *Technique et Science Informatiques*, 7 (1), pp. 21-45.

[35] W. Paper, 2001. Capitalizing on the Learning Object Economy. *Learning Objects Network*.

[36] R. Reiter, 1980. A logic for default reasoning. *Artificial Intelligence* 13, pp. 81-132.

[37] E. Rich and K. Knight. *Artificial Intelligence, second edition*. McGraw-Hill, 1991.

[38] S. Russel and P. Norvig. *Artificial Intelligence, a modern approach*. Prentice Hall, 1995.

[39] F. Sebastiani, 1996. *Logica e Rappresentazione della Conoscenza*. Servizio Editoriale Universitario di Pisa.

[40] D. H. Sleeman, 1987. PIXIE: A shell for developing intelligent tutoring systems. In *Artificial Intelligence and Education: Learning environments and tutoring systems*, R. H. Lawler and M. Yazdani Eds. Norwood NJ: Ablex Publishing, pp. 239-265.

[41] H. Stuckenschmidt and H. Wache, 2000. *Context Modelling and Transformation for Semantic Interoperability*. In: Knowledge Representation meets Databases - Proceedings of the Workshop at

ECAI 2000.
(http://www.tzi.de/~heiner/public/KRDB-00.pdf).

[42] L. A. Zadeh, 1975. Fuzzy logic and approximate reasoning. *Synthese* 30.

[43] W3C. *Resource Description Framework (RDF) Schema Specification 1.0*, 2000. W3C Candidate Recommendation 27 March 2000.
(http://www.w3.org/TR/2000/CR-rdf-schema-20000327).

[44] Advanced Distributed Learning. *SCORM Overview.*
(http://www.adlnet.org/index.cfm?fuseaction=scormabt).

[45] Advanced Distributed Learning (ADL). (http://www.adlnet.org/)

[46] Outline Processor Markup Language (OPML). (http://www.opml.org/)

[47] J. Allen. *Making a Semantic Web.* (http://www.xml.com/pub/a/2000/07/17/syndication/rss.html)

[48] R. E. Kent. *Conceptual Knowledge Markup Language: The Central Core.* The Ontology Consortium. (http://sern.ucalgary.ca/ksi/kaw/kaw99/papers/Kent1/CKML.pdf)

[49] R. Wille, 1982. Restructuring lattice theory: an approach based on hierarchies of concepts, in, I. Rival (ed.), *Ordered Sets*, Reidel.

[50] B. Ganter and R. Wille, 1989. Conceptual scaling, in, F. Roberts (ed.), *Applications of Combinatorics and Graph Theory in the biological and Social Sciences*, Springer-Verlag.

[51] B. Barwise and J. Seligman, 1997. *Information Flow: The Logic of Distributed Systems*, Cambridge University Press.

[52] M. Biezunski, M. Bryan, S. R. Newcomb (eds), 1999. *Topic Maps: Information Technology - Document Description and Markup Languages*, ISO/IEC 13250:2000.

(http://www.y12.doe.gov/sgml/sc34/document/0129.pdf)

[53] Edutella. (http://edutella.jxta.org/)

[54] W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmer, T. Risch, 2002. *Edutella: A P2P Networking Infrastructure Based on RDF*. WWW2002, Hawaii.
(http://edutella.jxta.org/reports/edutella-whitepaper/)

[55] Stanford University. Protégé 2000. (http://protege.stanford.edu)

[56] Stanford University. Planning a Protégé 2000 Project. *Protégé 2000 User Guide.*
(http://protege.stanford.edu/doc/users_guide/)

[57] Y. Sure, M. Erdmann, J. Angele, S. Staab, R. Studer, and D. Wenke. *OntoEdit: Collaborative Ontology Development for the Semantic Web. (* http://www.aifb.uni-karlsruhe.de/WBS/ysu/publications/ 2002_iswc_ontoedit.pdf*)*

[58] S. Bechhofer, I. Horrocks, C. Goble and R. Stevens. OilEd: a Reason-able Ontology Editor for Editor for the Semantic Web. (http://www.cs.man.ac.uk/~horrocks/Publications/ download/2001/oiled-dl.pdf)

[59] J. C. Apirez, O. Corcho, M. Fernandez-Lopez and A. Gomez-Perez, 2001. WebODE: a Scalable Workbench for Ontological Engineering. K-CAP'01, Canada.
(http://mkbeem.elibel.tm.fr/paper/KCAP2001-35.pdf)

[60] Stanford University Knowledge Systems Laboratory. Ontolingua.
(http://www.ksl.stanford.edu/software/ontolingua/)

[61] Information Sciences Institute. SENSUS Ontosaurus. (http://mozart.isi.edu:8003/sensus2/)

[62] U. Visser, H. Stuckenschmidt, H. Wache, T. Vogele, 2000. Enabling Technologies for Interoperability. NEC Research Index. (http://citeseer.nj.nec.com/459286.html)

[63] Stanford University. Scalable Knowledge Composition (SKC). (http://www-db.stanford.edu/SKC/)

[64] W3C. Requirements for a Web Ontology Language. W3C Working Draft 08 July 2002.
(http://www.w3.org/TR/webont-req/)

[65] MIT. Open Knowledge Initiative. (http://web.mit.edu/oki/)

[66] m-Learning Project. (http://www.m-learning.org/)

1.5 Overview

This survey on methods and standards for didactic knowledge modelling contains the following information:

- Chapter 2 introduces some general aspects of knowledge representation.
- Chapter 3 gives an overview of the most important formal logic systems used by Artificial Intelligence and Knowledge Representation communities.
- Chapter 4 shows the main knowledge representation techniques based on network data structures. The Unified Modelling Language is analysed as a general instrument for knowledge representation.
- Chapter 5 shows the main Intelligent Tutoring System architectures and how they exploit the general knowledge representation techniques introduced in the previous chapters.

Finally, Chapter 6 shows the most important knowledge representation standards capable of joining general representation techniques such those introduced in Chapters 3 and 4 with standardization issues.

2. General Purpose Knowledge Representation Methodologies

Information about a domain owned by a generic intelligent agent is composed of *explicit* and *implicit* knowledge. Explicit knowledge concerns a set of facts directly represented in the agent memory, while implicit knowledge regards all those facts which can be deduced by applying some kind of automatic reasoning to explicit knowledge. Thus, knowledge representation approaches must take into account both explicit and implicit ways to "encode" information about the agent world.

The Knowledge Representation community follows two different description methods. The first utilizes *logic languages* to represent explicit knowledge and *logic inference* to deduce implicit knowledge. The second represents knowledge by means of graph structures, following the approach proposed by Charles Pierce, who, in 1896, seven years after Peano developed what is now the standard notation for first order logic, proposed a graphical notation called "existential graphs". The latter are known today as *semantic networks*, and, since the time of Peano and Pierce, the Knowledge Representation community has debated the advantages and disadvantages of logic and semantic network approaches. At the beginning of '80s, logic began to be considered as a unifying language in which it is possible to precisely express semantics of a knowledge base. Furthermore, results about the decidability, tractability and expressiveness of logic formalisms can be applied to non-logic formalisms (such as semantic networks): it is enough to translate the latter into the former.

For these reasons, we first describe the most common logic formalisms used by the Knowledge Representation community, then we show different "graph-oriented" representation methodologies, underlining how semantic, computational and representational properties of these methodologies can be analysed, referring to their logical counterparts.

3. Logic Approaches

Logic studies *inference*: the way in which it is possible to (mechanically) deduce new facts from old ones. For this reason it is strictly linked to knowledge representation: logic languages, indeed, can be used to describe explicit (or "*declarative"*) knowledge and can infer new facts describing implicit knowledge. We can compare different knowledge representation methods by studying their logic structures. In this way, theoretical results on decidability, tractability and expressiveness of different logic formalisms can help one to choose the right trade-off between expressive power and computational costs of a representation method.

There are, of course, a lot of logic formalisms, each of which is used to study a particular aspect of human reasoning. The Knowledge Representation community has contributed to logic by proposing interesting new problems and new formalisms to solve them. In the following we will see the most common logic formalisms, not pretending to exhaustively cite all existing ones, and focusing our attention only on knowledge representation issues relating to logic studies.

First of all, we start with a definition of *logic formalism*.

Definition 1. A formalism (or formal system) is a tuple: $S = <A, WFF, RUL, AX>$, where:

- A is an alphabet of symbols.
- WFF (Well formed Formulas) is a set of finite words on V^*, i.e. is a subset of the set V^* of all the possible finite expressions that can be composed using V.
- $RUL = <R_1, ..., R_n>$ is a set of relations on WFF (inference rules), such that $R_i \subseteq WFF^{k}_{i}$ for some k_i. We say that the k_i-th element can be *derived* by the other $k_i - 1$ ones by means of R_i.
- AX is a set of formulas ($AX \subset WFF$) called *axioms*.

WFF is called the *language* of S, while RUL and AX are its "inferential mechanism". For example, if S is Prepositional Logic, we can have that A, B and $A \rightarrow B$ are three well formed formulas (from now on wffs) belonging to WFF and $R=(A, A \rightarrow B, B)$ is a rule (Modus Ponens) belonging to RUL.

Definition 2. Given a formal system S:

- A demonstration of a wff A_n from a set of wffs G (*assumptions*) is a sequence $A_1, ..., A_n$ such that each A_i is either an axiom, or a wff in G, or it can be derived by means of some $R_i \in RUL$ from a subset of $\{A_1..., A_{i-1}\}$. In this case, we write: $G \vdash_S A_n$.
- A *theory* G is a set of wffs such that $G = Der(G)$, where $Der(G)$ means the set of all wffs derivable by G (G is closed with respect to derivation).
- A set of wffs G is consistent iff $Der(G) \neq WFF$.

Thus, we have that, given a formal system S, with its language and inferential mechanism, we can express explicit knowledge in S using members of WFF (formulas): the set of all the facts explicitly known by an agent are its assumptions, i.e. a set of formulas G. Conversely, the implicit knowledge is the set of all the facts derivable by G using axioms and inference rules of S and not already belonging to G ($Der(G) - G$). All the knowledge (explicit and implicit) of the agent is $Der(G)$.

Finally, we introduce this property of a formal system:

Proposition 1 (Monotonicity). If $G \vdash_S A$, then $G \cup B \vdash_S A$.

The monotonic property of a formal system states that no new fact, acquired by an intelligent agent, can contradict a previous acquired or deducted one. We will see in the following that not all the logic

formalisms accepts this restriction.

We now show some of the most common logics used by the Knowledge Representation community, sketching, for each of them, their expressive power (depending on the language) and their computational capabilities.

3.1 Prepositional Logic

The most simple formal system is *Prepositional Logic* (PL_0), so called because it is based on atomic prepositions possibly linked with suitable *connectives*.

Formally, we have:

- **Alphabet (V).** It is composed by:

  A set of prepositional variables (*p, q, r, ...*).

  The following symbols: $\wedge, \vee, \rightarrow, \neg, (,)$.

- **Language (WFF).**

  Each prepositional variable is a wff.

  If *A,B* are wffs, then $A \wedge B, A \vee B, \neg A, A \rightarrow B, (A)$ are wffs.

- **Axioms (AX).** There are three axioms (more exactly axiom schemas), proposed by Hilbert, and known as *AK, AS* and $A\neg$, but they are quite unintuitive, and their treatment is out of the scope of the present work, so we will omit them. Opposed to Hilbert's style, *Natural Deduction* is a different way in which to express PL_0. In this second case, there is no axiom at all and all the inference power is based on rules, of greater number than Hilbert's approach.

- **Rules (RUL).** There is only one rule in PL_0: *Modus Ponens*. Formally, its relation on WFF is: $\{(A, A \rightarrow B, B) | A,B \in WFF\}$, that can be written as: $A, A \rightarrow B \mapsto B$ (from *A* and $A \rightarrow B$ follows *B*).

Notice that the symbol "$\rightarrow$" is syntactic, belongs to *V*, and it is different from the symbol "$\mapsto$", which is meta-syntactic (does not belong to *V*) and denotes *logic derivation*. We can have logic systems in which "$\mapsto$" is not included in *V* while derivation of a formula from others is however possible.

In PL_0 we can express statement such as: $((p \wedge q) \vee (\neg r)) \rightarrow t$. Prepositional variables such as *p,q,r,...* represent atomic sentences, such as "The sun rises", "The book is old", ... To give semantics to a wff we have to *assign* a truth value to each of its prepositional variables. For example, if we assign: *p = true, q = false, r =true, t = true*; then the above wff is *true* (we will not deal with semantics rules (such as *denotational semantics*) in this work, being interested only in showing general expressiveness properties of logic languages).

PL_0 can be used together with a set of assumptions *G* (sentences with a true value) to describe knowledge of an intelligent agent about a domain. Such knowledge must be composed of atomic sentences possibly composed by connectives.

PL_0 is decidable: proving that a formula α with $p_1, \ldots, p_n$ prepositional variables is a tautology (i.e. it is always true, whatever is the truth value of $p_1, \ldots, p_n$) is a NP-Complete problem. However, formulas not very large (*n* small), used in practical applications, can be checked quite easily.

The biggest shortcoming of PL_0 is its poor expressive power. The problem arises because we can not state properties common to individual objects of the domain. For example, it is impossible, in PL_0, to deduce something like: *if* "All humans are mortal" *and* "Socrates is human" *then* "Socrates is mortal". For this reason, in practical applications PL_0 is rarely used, and often one prefers to use more powerful logic systems like First Order Logic.

3.2 First Order Logic

If we say: "*x* is mortal", the truth value of this preposition is not univocally determined but depends on the variable *x*. This kind of preposition is called *predicate P(x)*. *P(x)* semantics is a set: the set of all individual objects of the domain which satisfy the property *P*. Furthermore, to assert that a given property *P* is true far all the objects in the domain or for at least one, we can use, respectively, two

quantifiers $\forall$ and $\exists$. For example, if *M* and H are two predicates and their meaning is, respectively, "Mortal" and "Human", the sentences: A_1 = "All humans are mortal", A_2 = "Socrates is human" and A_3 = "Socrates is mortal" became: A_1 = "$\forall x.H(x) \rightarrow M(x)$", A_2 = "$H(Socrates)$", A_3 = "$M(Socrates)$".

Predicate Calculus or *First Order Logic* (FOL) is the "traditional" logic formal system: there exist a lot of theoretical and practical results based on it, and programming languages (such as Prolog) implementing simplified forms of FOL inference.

FOL can be described in the following way:

- **Alphabet (V).**

  Individual variables (*x, y, z,...*).

  Predicate symbols (P^n, Q^m,...), where *n, m* are the ariety of *P* and *Q*.

  Constant and function symbols (f^n, g^m,...) , where *n, m* are the ariety of *f* and *g*. If $n = 0$, then *f* is a constant symbol.

  Prepositional connectives (and brackets): $\wedge, \vee, \rightarrow, \neg, (,)$.

  Quantifiers: $\forall$ and $\exists$.

- **Language (WFF).** First of all, we define the set of possible *terms T* of FOL:

  Each variable is a term.

  If $t_1, ..., t_n$, are terms and $f^n \in V$, then $f^n(t_1, ..., t_n)$ is a term.

  Hence, WFF is given by:

  If $t_1, ..., t_n$, are terms and $P^n \in V$, then $P^n(t_1, ..., t_n)$ is a wff.

  If *A,B* are wffs, then $A \wedge B, A \vee B, \neg A, A \rightarrow B, (A)$ are wffs.

  If *A* is a wff and *x* is a variable, then $\forall x.A, \exists x.A$ are wffs.

- **Axioms (AX).** All the axioms of PL_0 plus another two. For example, the S*pecialization* axiom (*Spec*) is: $\forall x.A \rightarrow [t/x]A$. Intuitively, Spec states that, if a formula *A* is true for all the elements of a domain, then it is true also for a particular element *t* of the domain.

- **Rules (RUL).** Modus Ponens plus *Generalization*: $A \mapsto \forall x.A$.

Example. Given a set of assumptions $G = \{A_1, A_2\}$, where A_1 and A_2 are the formulas above seen, it is possible to derive A_3 from *G* using axiom Spec applied to A_1 and obtaining A_1' = $H(Socrates) \rightarrow M(Socrates)$; then applying Modus Ponens to A_1' and A_2 to have, finally, A_3 = $M(Socrates)$.

The set of assumptions *G* has the aim to declare knowledge about function and predicate symbols. For example, the famous *PA* system, Peano's proposal to formalize arithmetic (Peano Arithmetic) is a formal system which includes FOL, plus a set of *characteristic axioms* (to distinguish them from *logical axiom AX*) such as: $\forall x.x + 0 = x$, where "0" is a constant, "+" a binary function and "=" a binary predicate belonging to PA Alphabet. PA characteristic axioms code knowledge about arithmetic rules.

PA is semi-decidable, and, generally, $G \mapsto A$ in FOL is semi-decidable. Nevertheless, FOL is very often used in practical applications, due to its very great expressive power. We can declare explicit knowledge of an intelligent agent using FOL, collecting a set of assumptions *G*. Constants represent individual objects (documents, persons or whatever our domain is composed of), like Socrates in the previous example. Functions are, instead, attributes of each single individual: in this way is possible to associate objects to real values. Functions can be computed in different manners, the most efficient of which is to attach procedural code to them. Finally, predicates are n-ary relations among individuals.

3.2.1 Programming Languages and Theorem Provers

Implicit knowledge can be computed exploiting the FOL inference mechanism. There are a lot of automatic system able to implement inferences in FOL. An example is logic programming by means of the *resolution rule*. Resolution can be applied to a knowledge base expressed by means of *Horn clauses* of the type:

$$\forall x.\forall y.\forall z.P(x,y,z) \leftarrow Q(x,y) \wedge R(y,z) \wedge S(z).$$

Usually, the universal quantifiers $\forall$ are omitted and the above formula is represented as:

$$P(x,y,z) \leftarrow Q(x,y) \wedge R(y,z) \wedge S(z).$$

The semantics of this clause is: *P(x,y,z)* is implied by *Q(x,y) and R(y,z) and S(z)*. The agent explicit knowledge is given by a list of *alternative* clauses (called *logic program*):

- $Parents(x,y,z) \leftarrow Child\_of(z,x) \wedge Child\_of(z,y)$;
- $Parents(x,y,z) \leftarrow Married(x,y) \wedge Child\_of(z,x)$;
- $Parents(x,y,z) \leftarrow Married(x,y) \wedge Child\_of(z,y)$;
- …
- $Sister\_of(x,y) \leftarrow Famel(x) \wedge Patents(m,f,x) \wedge Patents(m,f,y)$;
- …
- *Female(Barbara)*;
- *Female(Mary)* ;
- *Male(Albert)* ;
- *Male(Edward)* ;
- *Married(Barbara, Albert)*;
- *Married(George, Victoria)*;
- *Child_of(Edward, Barbara)*;
- *Child_of(Mary, Sthephen)*;
- *Child_of(Sthephen,George)*;
- …

The above example of logic program states some facts about the individual constants *Albert, Victoria, Edward, Mary…* and some alternative ways to derive relations such as *Parents* by other relations and facts. Indeed, *Parents(x,y,z)* can be derived by *Child_of(z,x) and Child_of(z,y)* **or** by *Married(x,y) and Child_of(z,x)* **or** … A resolution rule allows implicit knowledge to be derived in the form of *goal clauses* such as the query "$Sister\_of(Mary, Edward)$?" by means of explicit knowledge contained in the logic program.

An important example of automatic FOL reasoning system exploiting the resolution principle is Prolog, one of the most famous logic programming languages, in which the resolution rule is implemented by means of a backtracking mechanism. Prolog and Prolog-like systems are very common, nevertheless, this approach, in spite of its simplicity, has a number of shortcomings, the most important of which are:

1. Horn Clauses do not allow negation to be easily expressed.

2. The Backtracking mechanism used by Prolog to implement resolution (effectively a depth-first search) is very simple but it is not *complete*, i.e., it can diverge when a solution exists.

Moreover, Prologs unification algorithm, needed to unify a goal clause with clause heads, is not sound because, for efficiency reasons, it omits occur-check. However, there are extensions of Prolog which overcome some of its difficulties, allowing negation and adopting different search rules (breadth-first, iterative deepening, …) and sound unification algorithms. Furthermore, there exist a lot of theorem provers and logic programming languages based on FOL and exploiting different automatic reasoning approaches, such as tabloux. Examples of commercial theorem provers are: SAM, AURA, OTTER; examples of commercial logic programming languages are: MRS, Life.

3.2.2 Production Systems

Another important automatic reasoning metaphor able to exploit full FOL language expressive power are Production Systems. Let us give the following two definitions: an atom is a FOL wff composed only by a predicate symbol applied to a term, and a literal is an atom or the negation of an atom. Then, a typical Production System is based on the following points.

1. The working memory, containing the system explicit knowledge in the form of a set of positive literals with no free variables.
2. A set of production rules, each production rule being of the form: $p1 \wedge p2 \wedge ... \rightarrow act1 \wedge act2 \wedge ...$, where p_i is a literal and act_i is an action to take when all p_i are satisfied. Allowable actions are adding and deleting elements from the working memory.
3. A control structure, which decides what production rule to apply at each step. A rule can be applicable when its left-hand side matches with (is satisfied by) the current working memory (match phase). A rule is selected possibly according to heuristics (conflict resolution phase). The selected rule is applied to working memory according to actions contained in its right-hand size (act phase). Finally, some Production Systems provide failure recovering strategies such as backtracking.

It is worth noting that, although we have used FOL language to describe Production Systems, and often Production Systems work with a FOL knowledge representation, in their more general form they can adopt different knowledge representation formalisms, such as probabilistic reasoning, or nonmonotonic logics. In Section 3.6.2, for example, we will meet another type of Production System - Truth Maintenance Systems. Furthermore, Production Systems are not a purely logic inference mechanism, because of its notion of state (the working memory), continuously modified by actions of the selected rules. Indeed, Production Systems were originally proposed as a formal theory of computation and they are Turing-equivalent. Nevertheless, a Production System can be implemented by means of a FOL theorem prover, and this shows the expressive power of FOL.

One of the main differences between Prolog-like programming languages and Production Systems is the fact that the first are “backward chaining” reasoning methods, while the latter are “forward chaining”. This means that, in Prolog, the implicit knowledge is the query (goal clause) given to the system, and Prolog searches for a constructive proof to satisfy it. Conversely, Production Systems start from their explicit knowledge base and apply inference rules (Modus Ponens applied to knowledge represented in working memory and production rules) to derive new assertions.

Figure 1 shows an example of a Production System applied to geometric knowledge in a didactic domain. The system name is Geometric Tutor, developed by Anderson, Boyle and Yost (Anderson, Boyle and Yost, 1985 [7]). Below there is an example of forward chaining production rule of Geometric Tutor to represent the situation shown in Figure 1.

XY = UY ∧
YZ = YW ∧
Triangle(XYZ) ∧
Triangle(UYW) ∧
Collinear(XYW, UYZ) →
XYZ = UYW.

Other examples of (general-purpose) commercial Production Systems are: OPS-5, CLIPS, SOAR.

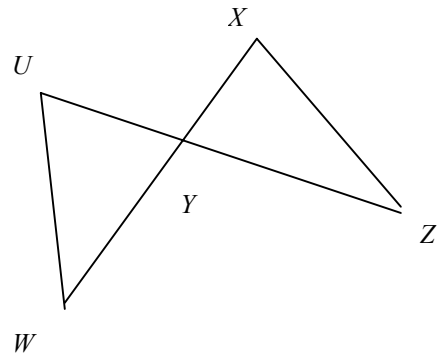


Figure 1. An example of a geometric problem treated by Geometric Tutor.

3.2.3 Some conclusions about FOL

FOL is a very powerful and expressive logic system. A lot of programming languages, theorem provers and production systems have been developed using FOL to represent explicit and implicit knowledge. Some of them are relatively efficient and if one needs all FOL expressive power, then they can be a good choice. We must remember, however, that FOL-based knowledge systems are intrinsically semi-decidable, i.e., any inference mechanism implementation can diverge if, for some input data and given assumptions, a solution does not exist (a formula can not be derived by the set of assumptions).

We will see less expressive formal systems, such as *Description Logics*, that have a better computational behaviour, but pay a price by having a less expressive language. For a lot of practical knowledge representation problems, Description Logics can be a better trade-off than FOL. Before we introduce them, for the sake of completeness, we will give a sketch of logics belonging to greater orders then FOL, although their practical interest is limited.

3.3 Second Order Logics

FOL can be generalized by introducing the possibility of quantifying over predicate and function symbols. For example, in a *Second Order Logic* (SOL) system, the following formula is well formed:

$$\forall P.P(x) \rightarrow P(y),$$

which means that for all (unary) predicate symbols of the language holds $P(x) \rightarrow P(y)$, i.e., for each set *P* of the domain, $P(x) \rightarrow P(y)$ holds. In this way we can explicitly assert facts about relations of our domain.

To this aim, we need to extend FOL alphabet (V) with a set of *predicate variables* and FOL language (WFF) with formulas of the type: $\forall P.A$, where *A* is a formula belonging to SOL WFF and *P* is a predicate variable.

The following is a (more interesting) sentence not valid in FOL but valid in Second Order Logic:

$$\forall I.\forall P.\forall x.Intervening(x,I,P) \leftarrow (x \in I) \wedge (I \subset P),$$

which means: for all individuals *x* and all sets *I* and *P*, the set *I* is an "intermediate" for *x* with respect to *P* if *x* belongs to *I* and *I* is a subset of *P*. $(x \in I)$ and $I \subset P$ stand for, respectively, *Belongs(x, I)* and *Is_Included_in(I, P)*, where *Belongs* and *Is_Included_in* are two predicates asserting knowledge about both individual (*x*) and predicate variables (*I* and *P*).

Nevertheless the appealing expressive power of SOL, it is rarely used in practical applications and we are not aware of automatic reasoning system able to derive implicit knowledge in SOL. Moreover, there are some theoretical problems with SOL semantics: for example, it is not *complete*, i.e., a *valid* formula may not be syntactically derivable. We will see later other logic formalisms, such as *epistemic logics*, which can *reify* (can treat a predicate as it were an object) so stating properties about relations without the need to use SOL (or higher order logics).

Finally, in SOL, if *G* is a set of assumptions and *A* a wff, then $G \mapsto A$ is not decidable. We now show less expressive but more tractable formalisms in which the decidability of implicit knowledge is achievable.

3.4 Description Logics

Description Logics (or *Terminological Logics*) can be alternatively viewed as:

- Logic evolutions of knowledge representation languages based on *frames* or *semantic networks* (see Section 4.1).
- Sets of "macro terms" of FOL.

Some of the first proposals of memory models based on frames or semantic networks were "Frames" (Minsky, 1975 [33]), KRL (Bodrow and Winograd, 1977 [11]), and KL-ONE (Brachman and Levesque, 1984 [13]). At the beginning of the '80s, there was a growing interest in logic formalization of these memory structures so as to better understand their scopes and limits. Description Logics (DLs) were originally proposed by the creators of KL-ONE (Brachman and Levesque, 1984). As we will see later, it is, indeed, possible to transform a language like KL-ONE, "translating" each of its statements in to logical expressions.

With respect to FOL, DLs do not have:

- Quantifiers.
- Connectives.
- Function symbols with ariety greater or equal to one (hence, they have only constant symbols).
- Predicate symbols with ariety greater or equal to three.

Nevertheless, they have some operators that can be viewed as fixed combinations ("macro") of them. The following is an example of well found formula for a DL:

(and Paper (atmost 2 Author) (atleast 2 Author))[paper1762],

which, in FOL, can be written as:

$$Paper(paper1762) \wedge$$
$$(\exists x.\exists y.\forall z.$$
$$Author(paper1762, x) \wedge$$
$$Author(paper1762, y) \wedge$$
$$x \neq y \wedge$$
$$Author(paper1762, z) \rightarrow (x = z) \vee (y = z))$$

Each DL contains operators (term constructors) by means of which it is possible to recursively build complex terms starting from an alphabet of unary and binary predicate symbols. Thus, a DL is syntactically characterized by its *terms*. More exactly, each DL is composed of three modules: a *terminological module* (TM), an *assertional module* (AM) and a *definitional module* (DM). MT is the characterizing part of each DL, describing what terms are syntactically possible in it, while AM and DM are two optional parts: they are the same for all DLs and describe how to obtain a well formed formula for the set of possible terms.

As an example, we show FL^-, a DL system proposed by Brachman and Levesque [13]:

- **Alphabet (V).** It is composed of:

  A set of individual constants $i_1, \ldots, i_n$.

  A set of unary predicate symbols $M_1, \ldots, M_m$.

  A set of binary predicate symbols $D_1, \ldots, D_o$.
- **Language (WFF).** First, we define a *term* in FL^-:

  A unary predicate M is a term.

  If $C_1, \ldots, C_n$ are terms, then *(and* $C_1, \ldots, C_n$*)* is also a term.

  If C is a term and D is a binary predicate , then *(all D C)* is also a term.

  If D is a binary predicate, then *(some D)* is a term.

This defines FL^- Terminological Module. Given the above TM, a wff in FL^- is:

If C is a term and i is an individual constant, then *C[i]* is a wff.

If D is a binary predicate and i_1, i_2 are two individual constants, then *D[*i_1, i_2*]* is a wff.

The above two definitions describe the Assertional Module. Finally, the following is the Definitional Module:

If M is a unary predicate symbol and C is a term, then $M \rightarrow C$ and $M \leftrightarrow C$ are wffs.

If D_1 and D_2 are two binary predicate symbols, then $D1 \rightarrow D2$ and $D1 \leftrightarrow D2$ are wffs.

Notice that each term represents a subset of the domain. Indeed, we can give semantics to terms associating them with sets defined with the help of FOL language:

- *(and* $C_1, \ldots, C_n$*)* = $C1 \cap \ldots \cap Cn$.
- *(all D C)* = $\{ x \mid \forall y.(x, y) \in D \rightarrow y \in C \}$.
- *(some D)* = $\{ x \mid \exists y.(x, y) \in D \}$.

This suggests that DL assertional and definitional formulas can be “translated” in FOL well formed formulas. Viceversa, there are FOL wffs that can not be expressed in a DL. An example is: $A \vee B$.

Indeed, in DLs, disjunction and negation cannot be completely expressed.

We can use AM and DM to build an explicit knowledge base G, while a term represents a query to G. For example, this is an FL^- Knowledge Base:

- Author(Gulliver's travels, Swift);
- Journalist(Smith) ;
- Journalist(Adams) ;
- Paper(paper1762);
- ...

And this is an example of term representing a query to it :

(and Article (some Author) (all Author Journalist)).

Its meaning is: "the set of all papers which have at least one author and which authors are all journalists. If we interpret DLs as Information Retrieval systems, a data base is given by means of a set of assertional and definitional assumptions G, a query by a term C, while the system answer is given by all the individual constants i such that:

$G \mapsto C[i]$.

A very important result is that $G \mapsto C[i]$ in FL^- is decidable in polynomial time! The reason for which FL^- is polynomial while PL_0 is NP-complete is that the former is not always more expressive then the latter. Indeed, although in FL^- we can express a limited form of quantification, neither disjunction nor negation, as mentioned above, can be completely expressed.

Other Description Logics can be derived starting from FL^- and adding to its TM different operators to build more expressive terms. Examples of common DL operators are: *atleast, atmost*, *c-some, sing*, … Table 1 shows their semantics. An example of modern and diffused commercial DL-based representation language is CLASSIC (Borgida et al., 1989).

Operator	Semantics
(c-some *D C*)	$\{x \mid \exists y.(x,y) \in D \wedge y \in C\}$
(atleast *n D*)	$\{x \mid \quad \| \{y \mid (x,y) \in D\} \| \geq n\}$
(atmost *n D*)	$\{x \mid \quad \| \{y \mid (x,y) \in D\} \| \leq n\}$
(sing *i*)	*{i}*

Table 1. Semantics of other commonly used DLs operators.

Generally, DLs are decidable, but often they have a computational complexity ranging among NP-complete, NP-hard or PSpace-complete.

We conclude this section remarking that it is not really correct to write $G \mapsto C[i]$. Indeed, we have given no axioms or rules for DLs able to derive wffs by other wffs. For historical reasons, in fact, DLs

have not been treated as "axiomatic" systems, it being generally preferred to give specific algorithms to infer knowledge. Thus, a more correct way to say that, from a set of assumptions (or knowledge base) *G*, we can infer a formula *C[i]* is: *C[i] is valid in G* (a semantic and not a syntactic concept), and we write:

$$G \Rightarrow C[i],$$

where the symbol "$\Rightarrow$" means semantic implication.

Finally, we cite that the most common algorithms used to implement automatic reasoning with DLs are variants of tabeaux calculus and they are based on a recursive splitting of DL terms in simpler and simpler ones trying to find an evident contradiction.

3.5 Modal and Epistemic Logics

Modal logics were initially introduced by Lewis in 1918 to discriminate among notions of "temporally true", "necessarily true" and "possibly true". To this aim logicians introduced two modal operators: □ ("box") and ◊ ("diamond"). Thus, in a modal logic, we can have: *A*, □ *A* and ◊ *A*, which, respectively, state: *A* is true, *A* is necessarily true and *A* is possibly true.

This seems to be more a theoretical or a philosophical question then a practical one, and, indeed, more recently modal logics have been studied mainly for the different interpretations of their modal operators. From the point of view of the Knowledge Reasoning community, modal logics are interesting first of all for their epistemic interpretation, which gives an instrument which is able to represent beliefs. The AI community, for example, exploits epistemic logics to describe different agents' knowledge about the world: for example, there exist proposals of automatic tutoring systems for modelling learners' beliefs through epistemic logics so as to provide a system of knowledge which can better interact with students [15]. Hence, we will follow the epistemic interpretation of modal logics and we will think of □ *A* as "the intelligent agent believes that *A* is true", while ◊ *A* as "the intelligent agent believes that *A* is possible". Indeed, ◊ is syntactically defined as:

$\Diamond = \neg \Box \neg$.

There are prepositional and first order modal logics. We will see only the former because they are the more common, because the most interesting representational features of modal logics are already present in their prepositional version, and because first order modal logics are a difficult subject due to their difficult semantics. From now on, we will say "epistemic logic" instead of "modal logics" and we will indicate □ *A* with: *Believe(A)*; finally, we will not use ◊ because the equivalence $\Diamond = \neg \Box \neg$ makes it redundant - usually epistemic logics use only the *Believe* operator.

The alphabet and the language of a prepositional epistemic logic is the same of PL_0, plus the modal operator *Believe*. Thus, if *A* is a wff in PL_0, then *Believe(A)* is a wff in all epistemic logics. This is also called a *reification* operation, because the *Believe* operator transforms a formula to a term and, hence, allows us to manage it as it were an object.

Historically, the first reification mechanism was proposed by Goedel, in his famous demonstration of PA incompleteness (1931). Goedel, indeed, used a predicate, *Theorem(x)*, which states that, if *x* is (a special code of) a generic Peano Arithmetic formula, then *Theorem(x)* is valid iff in PA there exists a demonstration of (the formula corresponding to) *x*. In this way, as we can see, it is possible to assert properties ("to be a theorem") of knowledge (the formula), i.e., by means of reification, we can express "knowledge about knowledge". The epistemic predicate *Believe* achieves the same result without the need of all PA axiomatization and Goedel complex coding mechanism, and provides a theory of suitable axioms and rules to specifically deal with the reification operation.

In more detail, we have:

- **Rules (RUL).** Modus Ponens plus *Necessity*: $A \mapsto Believe(A)$.
- **Axioms (AX).** All the axioms of PL_0 are axioms for epistemic logics. Moreover, there are other peculiar axioms, some of which are common to all epistemic logics, others only to some of them.

Indeed, we can distinguish an epistemic logic formalism from another by the set of ("optional") axioms it includes in its axiom set. The two most important of these axioms are:

1. $Believe(A) \rightarrow Believe(Believe(A))$, called *Introspection*.
2. $\neg Believe(A) \rightarrow Believe(\neg Believe(A))$, called *Negative Introspection*.

Epistemic logic systems which accept Introspection among their axioms state that, if an agent knows something (*A*), then it (or she-he…) is also *aware of knowing it*. Including Negative Introspection in the axiom set is a still more strong assumption, leading to this statement: if there is something (*A*), which the agent does not know, then the same agent *is aware that it does not know it*. In some contexts Negative Introspection is realistic because agents can be aware of what information is missing (if I do not know how to calculate square root math operation, then I am conscious of my ignorance), but in other contexts it seems to be too strong (if I were a peasant living in a medieval village, I may never have heard about square roots, so I can not be conscious of not being able to calculate them). Thus, different epistemic logics, characterized by different axiomatizations, can be useful for different knowledge representation needs.

To describe knowledge about a set of *n* different agents, we can use *multi-epistemic* logics. We only need to extend the alphabet with *n* epistemic operators, one for each agent: $Believe_1, ..., Believe_n$. For instance, the statement:

$Believe_1(\neg Believe_2(A))$

means that the first agent believes that the second agent does not know *A*.

Multi-epistemic logics can be extended with other interesting operators to denote "common knowledge": the set of facts about the domain that is the knowledge base common to all the *n* agents of the group. The first of these common knowledge operator is:

$E(A) = Believe_1(A) \wedge \ ... \wedge \ Believe_n(A)$.

E is a "macro" of $Believe_1, ..., Believe_n$. The second one is stronger and allows us to state not only that the fact *A* is known by all the agents, but also that all the agents knows that *A* belongs to group common knowledge:

$C(A) = E(A) \wedge EE(A) \wedge \ EEE(A) \ \wedge \ldots$

Notice that *C* is defined as a fixed point of an infinite expression and, thus, it is not derivable directly by *E* (it is not a "macro" of *E*).

All these epistemic logics (standard, multi-epistemic, with or without common knowledge operators, …), in their prepositional version, are decidable. Depending on the logic, we have computational complexity NP, PSpace or Exp-time. Nevertheless, general intractability results do not mean that in a specific domain for a practical application (with restrictions, for example, on the length of the formulas) epistemic logics can not be used with satisfactory time-consuming results.

3.6 Nonmonotonic Logics

If we ask someone if the sentence: "All birds fly" (formally, in a FOL-like language: "If *x* is a bird, then *x* flies"), it is likely she/he will answer yes. But if we reason about the fact that there exists some bird species which are not able to fly, such as penguins, the above statement is not correct. Nevertheless, humans usually think about the world by organizing their knowledge by means of "average" beliefs, i.e., facts that, in average (or "by default"), are true, but, *under special exceptions*, became false.

To describe such problems, logicians created nonmonotonic formal systems, in which previously believed facts can be successively retracted if new acquired knowledge is in conflict with them. The name "nonmonotonic" derives from the fact that these systems negate Preposition 1 (see Chapter 3). Indeed, if *S* is a nonmonotonic formal system and $G \vdash_S A$, then *this does not imply that* $G \cup B \vdash_S A$.

The need for nonmonotonic reasoning in AI had been recognized long before formal theories were proposed. In support of his argument against logic in AI, Minsky invoked the nonmonotonic nature of commonsense reasoning in one version of his 1975 "Frames" paper. Several knowledge representation languages, most notably KRL [11], already cited about Description Logics, specifically provided for "default reasoning".

The rest of this section is devoted to a critical examination of some of the most important formalization of nonmonotonic inferences.

3.6.1 The Closed World Assumption (CWA).

The CWA states that "all *basic* facts not contained in the agents explicit knowledge are assumed to be false". The CWA arises most prominently in the database theory, where it is assumed that all the relevant positive information has been specified. Any positive fact not so specified is assumed false. In the case of deductive data bases it is natural to understand that a positive fact has been specified if it is entailed by the data base, and that any fact not so entailed is taken to be false.

More formally, let *G* be a first order data base (i.e. any first order theory, given by considering both explicit and implicit knowledge: *Der(G) = G*). Reiter's definition of the CWA [36] is given by the following:

Closure(G) = G $\cup$ { $\neg P(t) | G \not\vdash P(t)$, where *P* is an *n*-ary predicate symbol and t is a *n*-tuple of

ground terms (without variables) formed using constant and function symbols}.

In other words, the *implicit* negative information of *G* sanctioned by CWA are those negative ground literals whose (positive) ground atoms are not entailed by the data base. Under CWA, queries are evaluated with respect to *Closure(G)*, rather then *G* itself.

There are several problems with CWA. The most obvious is that the data base closure might be inconsistent (see Definition 2), as would be the case for:

$G=\{\forall x.P(x) \vee Q(x)\}$.

Indeed, for a given *t*, $G \not\vdash P(t)$ and $G \not\vdash Q(t)$, so:

$Closure(G) = \{\forall x.P(x) \vee Q(x)\} \cup \{\neg P(t), \neg Q(t)\}$, that is clearly inconsistent.

In the case of Horn clause based databases (see Section 3.2.1), it can be shown that the Reiter Closure operator preserves the consistency of the database. However, even for nondeductive relational databases consisting only of ground atoms, Reiter's CWA yields incorrect results in the presence of so-called null-values. A null value is a constant symbol meant to denote an existing individual whose identity is unknown. For example, if *Supplies(s, p)* denotes that supplier *s* supplies part *p*, then the following is a simple database *G*, where *N* is meant to denote a null value:

$G = \{ Supplies(Acme, p_1), Supplies(Sears, p_2), Supplies(N, p_3)\}$.

So, we know that some supplier, possible the same as Acme or Sears, but possibly not, supplies p_3. Since $G \not\vdash Supplies(N, p_2)$, CWA sanctions $\neg Supplies(N, p_2)$, which, coupled with $Supplies(Sears, p_2)$, entail $N \neq Sears$. But this violates the intended interpretation of the null value as a totally unknown supplier: we have inferred something about *N*, namely that it is not Sears.

The CWA is very common in knowledge representations which take into consideration negative information but can not give a complete, explicit description of the domain (almost all the interesting data bases have this feature); so one must be careful in using it and take into consideration the above shortcomings.

There are, however, other definitions of nonmonotonic inference, which take care to avoid inconsistencies. They interpret the pattern: "In the absence of information to the contrary, assume *A*" as something like "If A can be consistently assumed, then assume it".

3.6.2 Nonmonotonic Logic.

McDermott and Doyle's nonmonotonic logic [22] appeals to a modal operator *M* in conjunction with FOL language. *M(A)* is intended to mean "*A* is consistent", so the flying bird example translates in their logic to:

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

There are, anyway, some difficult problems with the syntactic and semantic definition of the operator *M*. Nevertheless, Doyle gave an important and effective computational framework for default reasoning, that he called *Truth Maintenance System* (TMS). TMSs are deductive representational

structures which allow us to specify dependencies among facts. The basic idea is the following.

In a TMS one can assert some facts (explicit knowledge). Starting from these, the system can make some deductions (implicit knowledge). The novelty is that new deduced facts are associated with the assumptions from which they depend on, such that, if a new assumption negates some old one, also all deduced facts depending on this latter will be erased by knowledge base.

For instance, suppose that our database contains facts *A* = "Pat is a bird" and *B* = "Pat is alive" and that nothing is known about *C* = "Pat is a penguin". We mark *A, B* with the label "In" ("known to be valid"), and *C* with "Out". Moreover, suppose to have the following nonmonotonic formula:

$A \wedge B \wedge M(\neg C) \mapsto D$,

Where *D* = "Pat flies". Hence, we can derive *D*. We mark *D* with "In" and we associate to it two dependencies lists:

- *In_list(D) = (A, B)* , and:
- *Out_list(D) = (C)* .

Suppose now that new acquired knowledge leads us to believe *C*. Thus, *C* becomes "In", and all those inferences such as *D* in which *C* was supposed to be "Out" become "Out" (not believed any more).

TMSs (and their modified version JTMSs and ATMSs) allow us to automatically perform these kind of inferences through an efficient backtracking procedure called *dependency directed* backtracking, to distinguish it from the classical chronological one. TMSs are diffused algorithms for knowledge representation, also if their formal link with nonmonotonic logic is not very clear.

3.6.3 Default Logic.

Reiter 's default logic [36] differs from the nonmonotonic logic of McDermott and Doyle in that default statements are formally treated as rules of inferences, not as formulas in a theory. The flying bird default is represented by the following rule of inference (remember the difference between the symbol "$\rightarrow$" and the meta-symbol "$\mapsto$", as explained in Section 3.1)

$Bird(x) : Fly(x) \mapsto Fly(x)$.

This can be read as: "If *x* is a bird and can be consistently assumed to fly, then you can infer that *x* flies.". More generally, rules of the following form are permitted:

$A:B \mapsto C$,

which can be read as: if *A* holds and *B* can be consistently assumed, then you can infer *C*.

Also with Reiter's proposal there are some unsolved problems of syntax and semantics. Nevertheless, analyses using default logic have been applied to several settings in Knowledge Representation, such as inheritance hierarchies with exceptions (see Section 4.1).

3.7 Probabilistic and Fuzzy Logics

Knowledge representation cannot be completed if one does not take into consideration uncertainty. Nonmonotonic logics are a first example of the treatment of uncertainty, because they suppose to represent facts which are averagely true but that can be possibly negated by future knowledge acquisition. However, traditionally, logic is bivalent, accepting only two truth values (true or false), without intermediate probability values, while uncertainty intrinsically needs to be quantified by a larger set of values.

Uncertain knowledge is mostly treated by two different mathematical approaches: Bayesian rules and Kolmogorov axioms or Fuzzy logic. The former, is usually embedded in a complete knowledge representation system by means of Bayesian networks, as we will see in Section 4.3. Nevertheless, there have been several proposals to join logic and Kolmogorov probabilistic theories, and in the following section we will give a sketch of them. On the other hand, Fuzzy logic is a relatively new approach to modelling uncertainty, proposed by Zadeh [42]. It is a particular kind of logic, which is based on the rejection of the bivalent assumption, so it is quite a revolutionary logic formalism. This section is dedicated to both approaches.

Let us start by underlining that there exist different kinds of uncertainty, conceptually separate, even if human beings are not always aware of this:

1. Uncertain knowledge: "I think that the probability she will like this bottle is 0.91".
2. Statistic knowledge: "The probability that this bottle, randomly taken by a set of known bottles, contains drinkable water is 0.91".
3. Fuzzy knowledge: "This bottle contains drinkable water with a ratio of 0.91".

Notice the differences among the three cases. The first sentence states that who is speaking estimates that a fact about a bottle *is true* with a probability of 0.91. The second sentence states that the bottle is drinkable with a probability of 0.91, i.e., there are 91 possibilities on 100 that the bottle water is *completely drinkable*. The third sentence, instead, says that surly the bottle water is *partially* drinkable with a given percentage (the rest of the water in the bottle is not drinkable).

It is then natural to treat these three kinds of uncertainty with different methods. The first two are well suited to be formalized by means of Kolmogorov axioms and classic logic, because they state that a fact belongs to true or false fact sets with a given probability, while the third sentence states that a fact is only partially true (belongs only partially to the set of true facts), rejecting bivalent logic semantics.

3.7.1 Probabilistic Logics.

There are various proposal to join logic and probability by extending FOL with suitable probabilistic operators. Not one of the proposed operators can deal both with sentences of type one and two. To treat both it seems to be necessary to use two different operators. Here are some examples of a well formed formulas in PL3, a formal system containing both types of operators:

$GC(A) = 0.91$,

where A is the sentence: "She will like the bottle" and GC means: "the probability that"; notice that A can be a generic FOL wff. GC is the PL3 operator to treat uncertainty knowledge of the type 1.

$w_{<x>}\ Drinkable(x) = 0.91$,

where the term $w_{<x>}\ A$ means: "the probability that an individual x, randomly extracted from the domain, makes A true". $w_{<x>}$ is the PL3 operator to treat uncertainty knowledge of the type 2. In PL3 it is possible to use, in a same phrase, both operators GC and $w_{<x>,}$ as shown by the below example:

$w_{<x>}\ P(x) \geq GC(A)$.

An axiomatization for PL3 will contain:

- Axioms and inference rules for FOL.
- Axioms and inference rules for real numbers.
- Axioms and inference rules for w and GC operators. These will include Bayesian rules.

However, PL3 is not decidable nor semi-decidable: in these cases there can not exist a complete axiomatization (i.e. a recursive set of axioms completely describing the theory).

3.7.2 Fuzzy Logic.

Since its first proposal [42], Fuzzy logic gave rise to a long debate in the mathematical community about its utility. "Traditional" mathematicians think that Fuzzy Logic is only a different version of Bayesian formalization of probabilistic reasoning. However, we are not interested in entering into this debate. Since our point of view on formal systems is oriented to knowledge representation, our intention in this work is to give a sketch of the most important representation methods and to underline what are their scope and their limitations. Thus, we will not take into consideration the question whether Fuzzy logic can be completely substituted by Bayesian rules or not, but we speak about Fuzzy logic because it seems to be the most suitable instrument to represent the kind of uncertain knowledge of type 3.

With reference to sentence 3, the basic idea of Fuzzy logic is that some facts can be neither completely true nor completely false. The content of the bottle in the sentence is a typical example, because the statement *Drinkable(bottle)* is only partially true. Other typical examples are: Jim is tall (or he is intelligent or is able to do this thing or…). The set of tall people cannot be drastically distinguished from the set of short (not tall) people. It is, indeed, unnatural to fix a threshold and consider all tall

people to be higher than a single set height. It is more natural to think of the sets of tall people and the set of short people as overlapping, without continuity breaks.

From these considerations was born the idea of "fuzzy" set. Traditionally, we have that an individual *x* of a domain belongs or does not belong to a set *S*. This is described by the *characteristic function* of the set:

$$F_S(x)=\begin{Bmatrix} 1 & if & x \in S \\ 0 & if & x \notin S \end{Bmatrix}$$

Conversely, in fuzzy sets we have that an individual *x* of the domain belongs to a set *S* with a degree of membership variable in the continuous range [0,1]:

$0 \leq \mu_S(x) \leq 1$,

where $\mu_S(x)$ is called the *membership function* of the fuzzy set *S*.

As we saw in Section 3.2, in FOL a set is represented by a predicate *P(x)*, and, vice versa, each (unary) predicate *P* has an *extensional* definition given by: "*P* is a set *S* of elements of the domain. For all such elements *x, P(x)* is true". So, *P(x)* is such that:

$$P(x)=\begin{Bmatrix} true & if & Fs(x)=1 \\ false & if & Fs(x)=0 \end{Bmatrix}$$

But, if we adopt fuzzy sets and fuzzy membership functions, the truth value of *P(x)* must range in [0, 1] (instead of {0, 1}). Generalizing this concept, we have that a generic formula (such as a conjunction or disjunction of atoms, …) in Fuzzy logic is associated with a truth value ranging in [0, 1].

We give now an overview of Fuzzy logic syntax:

Alphabet (V). It is composed of:

- The two symbols 0 and 1.
- A set of individual variables $x_1, x_2, \ldots$
- Function symbols ($f^n, g^m, \ldots$) , where *n, m* are the ariety of *f* and *g*.
- Prepositional connectives (and brackets): $\wedge, \vee, \neg, (,)$.

Language (WFF).

1. 0, 1 and variables x_i are fuzzy expressions.
2. If $t_1, \ldots, t_n$, are fuzzy expressions and $f^n \in V$, then $f^n(t_1, \ldots, t_n)$ is a fuzzy expression.
3. If *f,g* are fuzzy expressions, then $f \wedge g, f \vee g, \neg f, (f)$ are fuzzy expressions.

So, if *f* and *g* are two fuzzy expressions, Fuzzy logic semantics is given by:

1. $0 \leq \mu(f) \leq 1$.
2. $\mu(\neg f) = 1 - \mu(f)$.
3. $\mu(f \wedge g) = \min(\mu(f), \mu(g))$.
4. $\mu(f \vee g) = \max(\mu(f), \mu(g))$.

The first important note about Fuzzy logic semantics is that it negates the Aristotelian principle of "tertium non datur". Indeed, from rule 2 follows that:

$\forall x \in]0,1[, \mu(x \wedge \neg x) \neq 0$,

while, in a bivalent theory, $A \wedge \neg A$ is shown to lead to inconsistency.

The second interesting point is that the previous definition of the semantics of the operators "and" and "or", leads to a different conclusion with respect to "classic" Bayesian probability. Indeed, in this latter approach we have that, for independent events, the probabilistic "and" operation corresponds to multiplication of the single probabilities, but this is not true for Fuzzy logic.

Suppose, for example, that *S* is the set of intelligent people and *T* is the set of tall people and *j* = "Jim". We can ask: "How tall and intelligent is Jim?" or: "What is the probability that Jim is tall and intelligent?".

Probabilistic calculus leads to a smaller value with respect to the two initial values for *S* and *T*, while fuzzy calculus states that Jim cannot be less "tall and intelligent" then he is tall or intelligent. Suppose, indeed, that $P(x \in T)$ is the probability that *x* is tall in a probabilistic formal system ("P" can be operator *GC*, for example). Similarly, $P(x \in S)$ is the probability that *x* is intelligent. Then we have that:

$P((j \in S) \wedge (j \in T)) = P(j \in S) \times P(j \in T)$.

If, for instance, we suppose $P(j \in S) = 0.9$ and $P(j \in T) = 0.6$, we have that:

$P((j \in S) \wedge (j \in T)) = 0.54$.

Conversely, if we suppose $\mu(S(j)) = 0.9$ and $\mu(T(j)) = 0.6$, where $S(x)$ and $T(x)$ are the membership functions of, respectively, *S* and *T*, for rule 3, we have that:

$\mu(S(j) \wedge T(j)) = min(\mu(S(j)), \mu(T(j))) = 0.6$.

3.8 Some Conclusions about Logic Approaches

We have seen various logic formalisms, each of them able to describe a particular aspect of human reasoning and knowledge representation. Generally, expressiveness power leads to greater computational complexity. The most expressive languages are not decidable, so each application must find its own trade-off.

Moreover, there does not exist a "general solution", i.e. a language able to cope all knowledge representation needs. FOL, generally regarded as one of the most complete formalisms, is not able to quantify on predicates, for example, or to express uncertainty. The human brain, indeed, seems to be able to switch among different contexts using, for each of them, the appropriate knowledge representation.

For these reasons, Knowledge Representation and AI communities believe there is the need to adopt an eclectic approach to knowledge coding, using different systems for different practical needs.

4. Knowledge Networks

In this chapter we will focus our attention on knowledge representation methods which exploit a model of memory structured in networks. The main difference between this approach and the logic approach is the way in which information is structured. In knowledge networks, explicit information is described by network nodes and links, both possibly labelled. In logic approaches, rather, explicit information is given by a list of strings (formulas). In network methods, implicit knowledge is derived by suitable algorithms following network links. In logic methods, implicit knowledge is deduced using automatic reasoning systems (theorem provers and so on) tailored on the sets of axioms and rules characteristic of the specific formal system chosen.

From this basic difference follow two important consequences. The first is the "representational format" with which a human reader can read a knowledge base described by logic systems or by knowledge networks. The two representational points of view adopt different notations (or formats): the former is textual and the latter is graphical, and this has an effect on its clarity for a human reader. Some things are easier to understand in a graphical notation, and some are better shown as strings of characters.

The second most important consequence is the computational cost of implicit knowledge deduction. Automatic logic reasoning systems have to perform, at each derivation step, a match between formulas' symbols. For example, in a resolution rule fashion (see section 3.2.1) each atom of the goal clause can be matched with one of the clause heads of the logic program. Thus, at each step, Prolog must check all of the logic program (i.e. the whole list of clauses of the knowledge base) and, if a match between atoms is possible, it must perform a unification between goal and clause variables. In a network structure, rather, it is not necessary to check all the knowledge base to perform a match, because the algorithm only needs to follow links to find the related facts it is looking for. Thus, generally speaking, network knowledge representations can be faster than logic ones. On average, the former require only a few machine cycles per inference step. However, if we look, for example, at the kind of computations done by compiled Prolog programs, we see there is not much difference. A compiled Prolog program for a set of subset and set-membership sentences, combined with general properties of categories, does almost the same computations as a semantic network.

From an expressiveness viewpoint, there is no network notation which cannot be translated into a logical one. So, logic representation expressiveness is at least as powerful as network-based representation. The reverse is not always true: i.e., not all logic systems' expressive power can be translated into network structures. For example, semantic networks usually have an expressive power equivalent to Description Logics. There are various proposals to extend semantic networks to deal with quantification or negation or disjunction, as we will see, but this leads to a growing complexity of the network and of its automatic reasoning algorithms.

It seems, then, that logic approaches have a greater expressive power but often require more computation time, and, as usual, the choice of the formalism to adopt strongly depends on the application needs.

In the following, we will give, for each type of graph-based representation that we will introduce, its corresponding logic translation with two aims in mind. The first is to better describe the semantics of each approach. The second is to allow a direct comparison of each (logic or not logic) representational formalism expressive power. Moreover, classifying networks' formalisms by means of logic ones will allow us to inherit results for corresponding logic systems, their scope and their computational limits.

4.1 Frames and Semantic Networks

As mentioned in Chapter 2, semantic networks derive from the work of Charles Pierce, who proposed, in 1896, a graphical notation called "existential graphs", now known as "semantic networks". In 1975, Minsky, in a paper [33] in which he attacked logic knowledge representation methods, formalised some basic concepts of "slot structures", or "frames".

Frame systems and semantic networks use the same metaphor: objects of the domain are nodes in a graph, these nodes are organized in a taxonomic structure, and links between nodes represent binary relations.

In frame systems binary relations are thought of as slots in one frame that are filled by another, whereas, in semantic networks, they are thought of as arrows between nodes. The choice between the frame metaphor and the semantic network metaphor determines whether you draw the resulting networks as nested boxes or as graphs, but the meaning and the implementation of the two types of systems can be identical. Examples of frame systems include: OWL, FRAIL, KODIAK. Examples of semantic networks include: SNEPS, NETL, Conceptual Graphs.

In the rest of this section we will say "semantic networks" to mean "semantic networks or frame systems".

4.1.1 Basic Concepts of Semantic Networks.

A node in a semantic network can represent an individual of the domain or a set (*class*) of individuals. Graphically, a node is drawn as a rectangle. Relations admissible are all binary and can be drawn with oriented arrows. If *A* and *B* are nodes, and *R* is a relation between them, we indicate it with *R(A,B)*. *A* can be an individual or a class of the domain, while *B* can be an individual, a class or a primitive data type (a real number, a string, a Boolean value, …), this last treated like an individual object. Two important relations common to all the nets are: *Subset(A,B)* and *Member_of(A,B)* (or *Is_a(A,B)*), indicating, respectively, that the class *A* is a subset of the class *B* and that the individual *A* is a member of the class *B*. *Subset* and *Member_of* links constitute a *taxonomic hierarchy* of the network. Generally, this hierarchy is not a tree but a Direct Acyclic Graph (DAG).

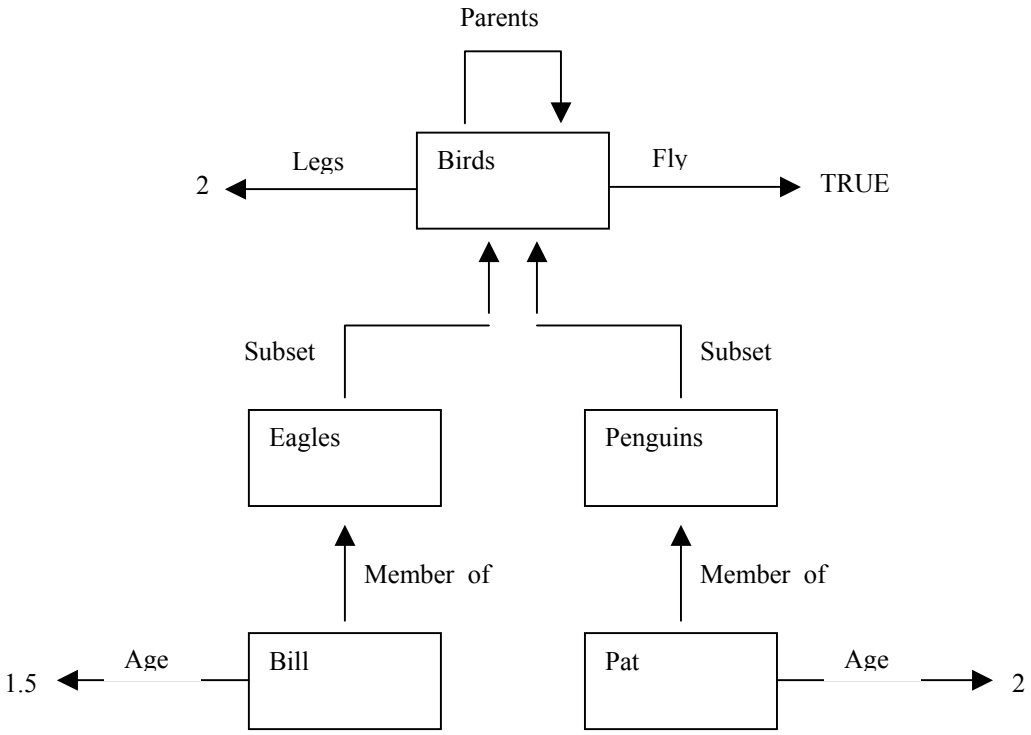


Figure 2. An example of a semantic network.

The other relations of a semantic network are specific to each system and depend on the domain one wants to describe. A semantic network, with its specific individuals and relations and with its link structure, is the equivalent of the set *G* of assumptions we saw in logic approaches, i.e. it constitutes the explicit knowledge of the system. The implicit knowledge is realized by means of a special-purpose algorithm able to follow network links.

If a relation *R(A,B)* holds, and if we can achieve *A* starting from *C* and following the taxonomic hierarchy (links *Member_of* and *Subset*) , then *R(C,B)* holds too. This property is called *inheritance*, and it is an example of the type of implicit knowledge contained in a network.

Figure 2 shows an example of a network.

4.1.2 Inheritance.

The previous figure shows a typical problem of inheritance property: the treatment of exceptions. Indeed, we know that Pat is a Penguin, that Penguins are Birds and that Birds fly, so, following taxonomy links, Pat flies. But if the network builder wants to avoid penguins flying in her/his network, and decides to update the net adding the relation *Fly(Penguins,FALSE)*, then Pat does not fly anymore, because the class Penguins is an intermediate between it and Birds, and this is enough to disable inheritance. This is a smart implicit treatment of nonmonotonicity, in which a fact (*Fly(Pat,TRUE)*) is automatically discarded when new knowledge makes it inconsistent.

The situation shown below is more difficult.

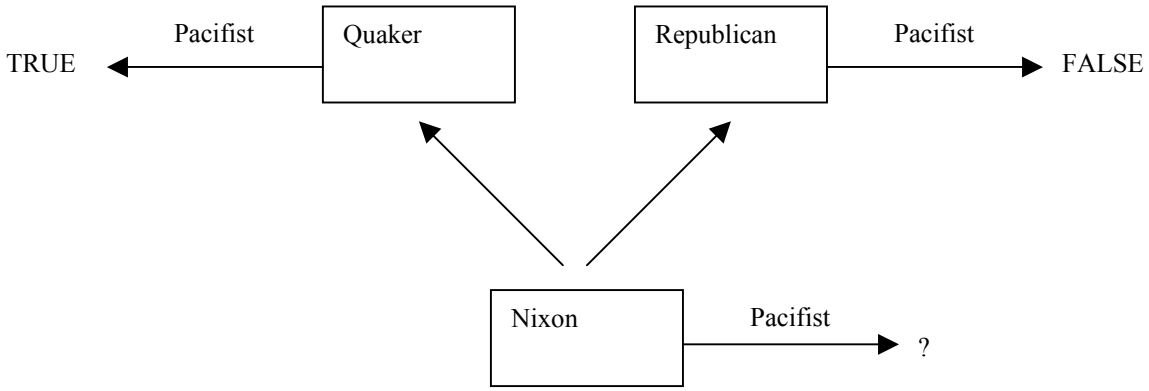


Figure 3. A traditional example of multiple inheritance problem, called "The Nixon diamond".

The above example is traditionally shown to explain the problem of *multiple inheritance*: an object can be a child of more then one class and, without additional information indicating some preference in the inheritance path to follow, there is no way to resolve the conflict.

4.1.3 Representation of Generic n-ary Predicates.

A relation in a semantic network is equivalent to a binary *ground* predicate in FOL. Nevertheless, representation of non-binary predicates in semantic networks is possible.

Concerning unary predicates, they can be represented in the way shown by the two previous examples, in which Boolean constant values TRUE and FALSE are objects of the domain (primitive data types) and, hence, can be referred to by ordinary relations. An example is: *Fly(Pat,TRUE)*, which, in FOL, is more naturally written as *Fly(Pat)*. A second approach utilizes relations *Member_of* and *Subset* in this way:

Pacifist(Pat)

can be written as:

Mamber_of(Pat, Pacifists).

Also generic n-ary predicates can be converted in binary form building a new object which represents the whole statement of the predicate and introducing binary predicates to describe relations of original arguments with the new object. If we suppose, for example, that:

Parents(Mary, Edward, Jim),

to mean that Mary and Edward are Jim's parents, then we can represent this fact in a semantic network building the object Jim_Family and then linking to it the three arguments of the predicate as shown in the following network:

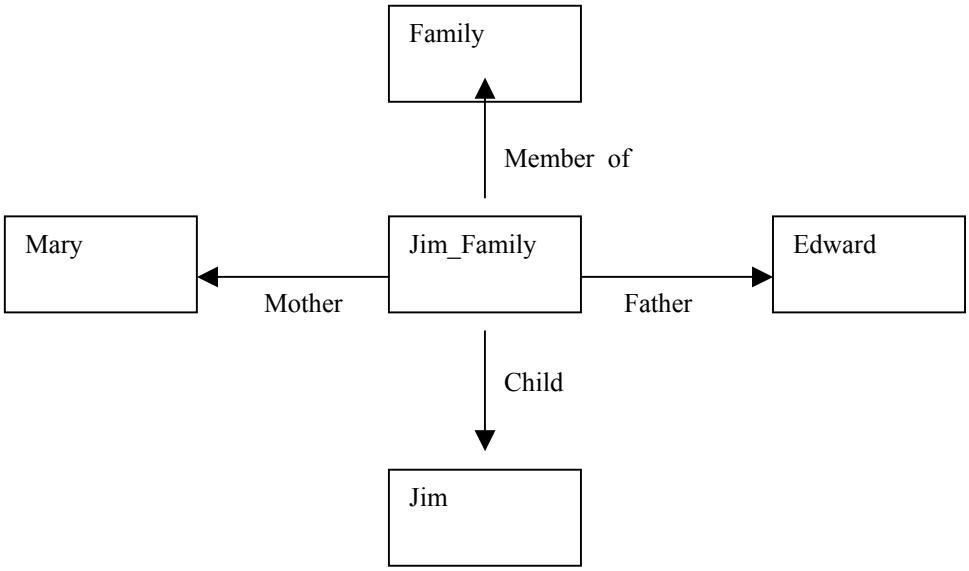


Figure 4. An example of representation through semantic network of a predicate with ariety 3.

4.1.4 Expressiveness and Computational Tractability.

In the following table we summarise the main types of relations of a semantic network, together with their "translations" in FOL language. All the examples refer to Figure 2.

Link Type	***A* Type**	**B *Type***	**Semantics**	**Example**
Subset(A,B)	Class	Class	$\forall x.x \in A \rightarrow x \in B$	*Subset(Eagles, Birds)* (1)
Member_of(A, B)	Object	Class	$A \in B$	*Member_of(Bill, Eagles)* (2)
Rel1(A, B)	Object	Object	*R(A, B)*	*Age(Bill, 1.5)* (3)
Rel2(A, B)	Class	Object	$\forall x.x \in A \rightarrow R(x,B)$	*Legs(Birds, 2)* (4)
Rel3(A, B)	Class	Object	$\forall x.\exists y.x \in A \rightarrow y \in B \wedge R(x,y)$	*Parents(Birds, Birds)* (5)

Table 2. The most common semantic network relation types.

It is interesting to note that all the above relations can be also translated into a Description Logic language:

1. $Eagles \rightarrow Birds$;
2. *Eagles[Bill]*;
3. *Age[Bill,1.5]*;
4. – 5. $Birds \leftrightarrow$ *(and (c-some Parents Birds) (c-some Legs sing 2))*.

Generally, a semantic network can be translated in a DL language (excepted its nonmonotonic treatment of inheritance with exceptions). From this follows two important consequences. The first is the decidability of semantic networks. Moreover, usually they are polynomially tractable and quite efficient because they do not need to retrieve and match sentences of their knowledge base because implicit knowledge is derived following suitable links. Conversely, the second consequence is negative and regards the expressive power, which is poorer then FOL, as we saw in Section 3.4. In particular, semantic networks, just like DLs, cannot completely express negation, disjunction and quantification.

Depending on the representational purposes, this may not be a problem. However, there were been various proposals to extend semantic networks expressiveness, first of all concerning quantification.

Extensions, of course, lead to a computational performance deterioration.

4.2 Metadata and Ontologies

Semantic network approaches do not specify the nature of the relations to be involved in a description, leaving this decision to the network engineer. Also logic approaches do not say what predicate and function symbols one must use. In other words, knowledge representation methods seen up to now give a general framework in which it is possible to express explicit knowledge and to derive implicit knowledge, but do not say anything about the type of representation to adopt.

One can choose, for example, FOL and Prolog inference engine for her/his knowledge needs, and thus is aware that she/he must use Horn clause, n-ary predicates and so on. But the choice of what predicates or constants to use is not on the responsibility of Prolog. In the same way, one knows how knowledge can be modelled in a semantic network, by means of taxonomic hierarchies and binary predicates, but, again, the method does not say what are the best relations to choose and does not pose any constraint on the kind of facts to represent.

This seems to be reasonable, because different domains have different properties to be modelled. On the other hand, different knowledge representation applications, on different domains but also on the same domain, using different languages, will not be able to exchange information between them. The need to unify languages arose when the AI and Data Base communities began to talk about automatic knowledge representation and reasoning systems. This problem was emphasized with the development of the Web. If Web power is its enormous mass of information and anarchic growth structure, it is also true that the same big mass and decentralized information distribution makes it hard to find what one is looking for.

Indeed, while centralized databases can adopt specific retrieval algorithms for specific data representations, the Web is a collection of data representations, each one specific to a particular application. If we think of the Web as an enormous library, then there is the need for a sort of catalogue to index the books: information must be organized to be efficiently retrieved. And, if we want different agents be able to exchange information, the language they adopt to describe data must be as uniform as possible.

The Knowledge Representation community responded to this need for standard languages by posing some constraints on knowledge descriptions. These constraints indicate common vocabularies of terms and common relations among terms. Vocabularies and relations are shared by all applications of a given domain and, possibly, also by applications of different domains in a hierarchy of standards.

Standard descriptions usually fall into two levels, different for expressive power and computational complexity: *Metadata* and *Ontologies*.

4.2.1 Metadata.

Metadata is a set of data about a particular resource. The name was born about 15 years ago, but it became particularly diffused with the development of Web. Metadata is a set of attributes (fields) needed to describe a given resource. Each field represents a feature of a generic resource (where a resource is an element of the domain) and can be instantiated with a value describing the specific feature of the indexed resource. Some fields can have multiple values, others can be *optional*.

Formally, we can define Metadata *M* as follows:

$M = \{A_1, \ldots, A_n\}$,

where A_i is a symbol belonging to a *shared vocabulary* and denotes an attribute of *M*, i.e. a relation between the indexed resource *R* and a set of primitive *data types* (such as strings, numbers, and so on) or between *R* and the shared vocabulary itself (in this case, the values of the attribute are fixed terms of the vocabulary).

The value of $A_i(R)$ can be a single value or a set of values. If the attribute is of the type optional, then the set of value can be empty. If $A_i(R) = t$ and t is a single value, in FOL we can write $A_i(R,t)$. Otherwise, if $A_i(R) = \{t_1, \ldots, t_m\}$, then in FOL we have: $A_i(R,t_1) \wedge \ldots \wedge A_i(R,t_m)$.

For example, if *R* is a document describing a book, and

$M = \{Title, Creator, Date, Publisher, \ldots\}$,

is metadata used to describe it (taken from Dublin Core Metadata standard, see Section 4.2.1), then *R* can be represented as:

M(R) = { Title = "Gulliver's travels",
Creator = "Jonathan Swift",
Date = 1726,
Publisher = ...,
... }.

Exploiting metadata, information can be retrieved using the fields which describe the resources instead of the text of the resources themselves, thereby augmenting efficiency and accuracy. At the same time, users who want to make their resources available to the whole Web community can label them with metadata so as to facilitate their retrieval.

The example shown in Figure 5 concerns searching in didactic subjects: users can choose among a list of fields and corresponding admissible values. Some fields can require the user to type a value such as the title. Searching consists of matching each user inserted value $A_i(R)$ with the corresponding values $A_i(R_j)$ for all indexed resources R_j.

Naturally, precision achievable exploiting metadata is much greater than keyword-based searching, with the essential condition that the knowledge engineer and the final users utilize the same terms for denoting attributes and attribute values.

Nevertheless, because of the enormous mass of different subjects retrievable on the Web, it is not easy to find a set of common terms able to describe each domain in a satisfactory manner. This consideration has led to various standardization initiatives working on different application domains. For example, the SCHEMA project, funded by European Community, constantly watches metadata standardization processes developed by various organizations in different sectors. In Table 3 we report the 21 initiatives analysed by SCHEMA.

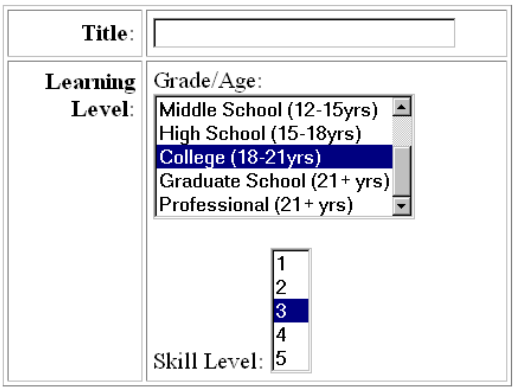


Figure 5. An example of didactic metadata.

In developing metadata schemas, each organization focuses on the solution to problems specific of its own applicative domain, problems depending on the type of data content and media used for data dissemination. However, there exist some generic standards (domain independent) which are interested in building a minimal shared vocabulary of terms for all application areas.

Together with standardization issues, it is important to underline a second aspect of metadata, that is, their low computational cost. Matching a list of attributes as above defined is, indeed, a very quick operation. Their negative aspect is (as usual in knowledge representation formalisms) the low expressive power.

A metadata base, composed of a set of instances of the type $M(R_j)$ to describe resources $R_1,R_2,...$is the system explicit knowledge and corresponds to a set of ground assumptions in FOL composed of binary predicates of the type: $A_i(R,t)$, where t is a primitive data type and cannot be a resource. With these

constraints no information about resources' interrelationships such as $A_i(R_1,R_2)$ is presentable.

4.2.2 Ontologies.

The name "ontology" comes from Greek philosophy and means "the study of the nature of being". Ancient philosophers studied what kinds of things exist in the world, trying to categorize them in different physical or meta-physical classes. The AI and Knowledge Representation communities use this word to mean the attempt "to categorize the kinds of things existing". The aim is not the philosophical enterprise of explaining all existing things, but, much more pragmatically, fixing a common vocabulary of terms able to describe as much knowledge about the world as possible and to subdivide this knowledge in a coherent class hierarchy, so as to create a shared knowledge representation language allowing different agents (accepting the same ontology) to understand each other.

An ontology is composed of the following features.

1. A vocabulary of (possibly standardized) terms.
2. Classes (or *categories*) denoting sets of objects of the domain.
3. Relations between data or between data and primitive data types (such as strings, numbers, Booleans, …).
4. Inheritance from other ontologies. An ontology, indeed, can extend another ontology deriving and extending properties from the original ontology.
5. Inference Rules of the type *if-then-else*.

1-4 extend metadata definitions to cope with all semantic networks' capabilities. Now, relations between Web resources such as $A_i(R_1,R_2)$ can be represented by means of 2. Concerning 5, rules' semantics usually need FOL language to be described, and Description Logics are not enough.

Acronymus	Complete Name	Organization	Category
BSR	Basic Semantic Registry	ISO	Industry
CEN/ISSS	CEN/ISSS Workshop Learning Technologies	CEN/ISSS	Didactic
CEN TC251	Health Informatics	CEN	Other
CERIF	Common European Research Information Format	CERIF	Research
Dublin Core	Dublin Core Metadata Initiative	Dublin Core	Generic
GELOS	Global Environmental Locator System Standard Element Set	GELOS	Other
GILS	Government Information Locator Service	GILS	Other
IMS	IMS Learning Resource Metadata Information Model	IMS Consortium	Didactic
ISO TC46/SC4	Information and Documentation: Computer applications in information and documentation;	ISO	Culture
ISO TC46/SC9	Information and documentation: Presentation, identification and description of documents	ISO	Publishing
ISO/IEC JTC1/SC32	Data management and Interchange; WG Metadata	ISO/IEC	Generic
LOM	Learning Object Metadata	IEEE	Didactic
MARC 21	Machine Readable Cataloguing	MARC	Culture
MPEG 21	Moving Pictures Expert Group: Digital Audio-visual Framework	MPEG	Publishing
MPEG 4	Moving Pictures Expert Group: Coding of audio-visual Objects	MPEG	Audio/Video
MPEG 7	Moving Pictures Expert Group: Multimedia Content	MPEG	Audio/Video

Acronymus	Complete Name	Organization	Category
	Description Interface		
NetCDF	Network Common Data Form	NetCDF	Research
PDS	Planetary Data System	PDS	Geography
RDF	Resource Description Framework	W3C	Generic
SMPTE	Society of Motion Picture and Television Engineers	SMPTE	Audio/Video
VRA	Visual Resources Association Data Standards Committee	VRA	Culture

Table 3. Some of the most important international metadata standards.

In conclusion, ontologies join standardization issues with quite good expressive power and computational costs. Generally, an ontology is structured as a semantic network, and its expressive power and computational cost can be compared with Description Logics. Indeed, typical DL-like languages such as those already mentioned KL-ONE (see Section 3.4) are often used to express ontologies. In this case, of course, not all the expressiveness of FOL can be represented by means of an ontology, but, and this is an important remark, ontologies are completely decidable and often tractable (i.e. have polynomial costs).

Finally, we cite formalism proposals such as Knowledge Interchange Format (KIF), very common in ontology representation, which matches the expressiveness of FOL (with the usual computational problems). KIF provides a Lisp-like syntax to express FOL sentences, together with the possibility of representing definitions and meta-knowledge. KIF is a low-level language, but Ontolingua, a Stanford University proposal for an ontology editing tool, allows users to build KIF ontologies at a higher level description.

In Chapter 7 we will see various ontology and metadata standards suited for didactic domains.

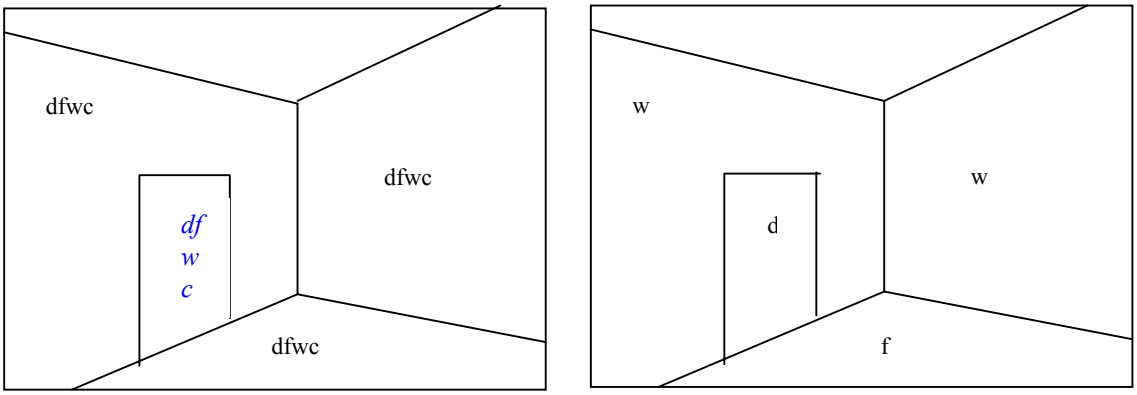
4.3 Bayesian Networks

Bayesian Networks (or Belief Networks or Probabilistic Networks) are knowledge representation structures organized in graphs. Their aim is to represent uncertainty knowledge in the form of the conditional probability of events given the probability of other events. If *A* and *B* are nodes of the Bayesian network associated with two random events, and there is an oriented edge from *A* to *B*, this means that the event *A* has an influence on the event *B* and this influence is quantified in a conditional probability estimate associated with the edge.

More precisely, a Bayesian network must satisfy the following properties:

- The network is a Direct Acyclic Graph (DAG)
- Each node of the network represents a random variable.
- If a direct edge *E* connects the node *A* with the node *B*, this means that the variable *A* has a direct influence on variable *B*. *E* is labelled with the value *P(B|A)*. *A* is called a *parent* of B and, vice versa, *B* is a *child* of *A*.
- A node can be associated with an a priori estimate of its probability.

There are various proposals to compute implicit knowledge in Bayesian networks. Here we sketch a method taken by Computer Vision applications. Rosenfeld proposed to represent visual information through Bayesian networks. An image is decomposed in primitive geometric elements. Each primitive region is labelled with all the possible interpretations and with the probability that the label is correct.



(a) (b)

Figure 6. A stylised segmented scene is decomposed in homogenous regions. (a) Each region is labelled with all possible interpretations: Door, Floor, Wall, Ceiling. (b) After an iterative process, the system is able to correctly choose the most probable interpretation.

Each pair region-label <*R, L*> is represented by a random variable associated with a node N in a graph which records the probability *P(R = L)* (referring to Section 3.7.1, P could be the probability operator *GC)*. *P(R = L)* is the a priori estimate that the region R could be interpreted as L, and comes from previous image local pre-processing.

Following the example of Figure 6, we have the following nodes:

N_1 associated with: $P(R_1 = d)$,
N_2 associated with: $P(R_1 = f)$,
…,
N_5 associated with: $P(R_2 = d)$,
…
N_{20} associated with: $P(R_5 = c)$.

Moreover, the network expresses conditional probabilities of the type:

$P(R_1 = f \mid R_2 = d)$, (label of the edge $E(N_5, N_2)$),
…,

and so on. Computation follows an iterative process, called *discrete relaxation* in which, starting from the initial values for $N_1,\ldots,N_{20}$, and following, in a forward chaining manner, links between nodes, new values for the nodes update the old ones until no change is possible (the process *converges*). The final state of the nodes' values is taken as the expression of the most likely probability distribution of labels on the regions.

4.4 UML as a Knowledge Representation Paradigm

In the database community, object-oriented techniques to model knowledge have been a popular paradigm for more then fifteen years. In recent years, UML has emerged as a unifying language in this area, and recently there have been some proposals to extend its scope to ontologies, Semantic Web issues, and other new knowledge representation aspects.

Some main features of object-oriented languages are:

- Individuals of the domain to be described are active objects and not passive ones. In other words, the classical mathematical-like paradigm of passive objects and active functions such as: *f(x) = y* (e.g. *age_of(Antony) = 15*), becomes, in the object metaphor, a universe in which objects embed functions in "methods" that can be asked about the object status (*x.f = y*, e.g., *Antony.age = 15*) or to modify the object status (in this case, *x.f* has a "side effect" on *x* internal representation).
- Objects are grouped in sets called classes. It is possible to define a taxonomic hierarchy of classes by means of generalizations or specializations. A class *A*, subset of a class *B*, inherits all properties of *B*: attributes, relations and methods are not redefined in *A*.

Recently, several authors have underlined the closeness of object paradigms with semantic network approaches to knowledge representation (although there are some important differences), proposing to use UML to cope with the standardization issues which typically surround the use of ontologies (see Section 4.2). Cranefield and Purvis in [17] observe that "object-oriented analysis, design and implementation is a maturing field with many industry standards emerging for distributed computation. The large user community and the commercial support for object-oriented standards warrants the investigation of standard object modelling techniques for ontology development.". Thus, considering that standards and tools for object-oriented programming have a wide and rapidly growing user community, they propose to investigate the possibility of using the UML standard object-oriented language to also represent ontologies.

The basic features of UML are the following:

- UML defines a *standard graphical* way of representing and drawing *diagrams* which model a knowledge base. These diagrams follows the database object paradigm and can be automatically translated into machine-understandable code by means of graphical tools, which allows the knowledge base engineer to draw the diagram having, as output, the code defining the corresponding classes in an object-oriented programming language (chosen from the most common commercial object-oriented languages).
- In UML diagrams, *classes* are represented by boxes with three parts: the name of the class; the attributes of the class, with their name, type and visibility; and the methods of the class, specified by name, argument list, return type and visibility.
- *Generalisation* is represented by lines with large hollow arrow heads pointing to the super class.
- *Associations* are represented by solid lines between pair of classes, with optionally named ends (or *roles*).
- Some associations can be labelled as *aggregation*: in this case, at the aggregate end of the association link there is a diamond.
- The ends of associations relationships may be annotated with *multiplicity* indicators giving the accepted range of numbers (e.g. [1..3] …) denoting how many instances of the class at the end of the relationship can be associated with each instance of the class at the other end. The symbol "*" represents infinity (e.g. [0..*] indicates an association with 0 or 1, 2, 3, …"any", instances).
- *Comments* can be expressed by means of large rectangles with folder corners, which contain uninterpreted pieces of text that may be anchored with dashed lines to model elements to provide informal clarification.

Another important feature of UML is the possibility of using the *Object Constraint Language* (OCL) to constrain attribute values and possible instances of the relationships. OCL still does not have completely defined semantics and its language is not as intuitive as FOL, but its potential

expressiveness is comparable with FOL.

In OCL language, the symbol *self* refers to the generic instance the constraint concerns. The value of an instance's attribute can be expressed by following *self* with a dot and the attribute's name, like in a general object-oriented language (see above: in this case, self takes the role of *x*). For example, *self.size* indicates the value of the attribute *size* of the instance. Using this notation, it is possible to express, in OCL, constraints on whatever instances appear in the UML diagram. Such constraints can be embedded in the diagram itself by using comments.

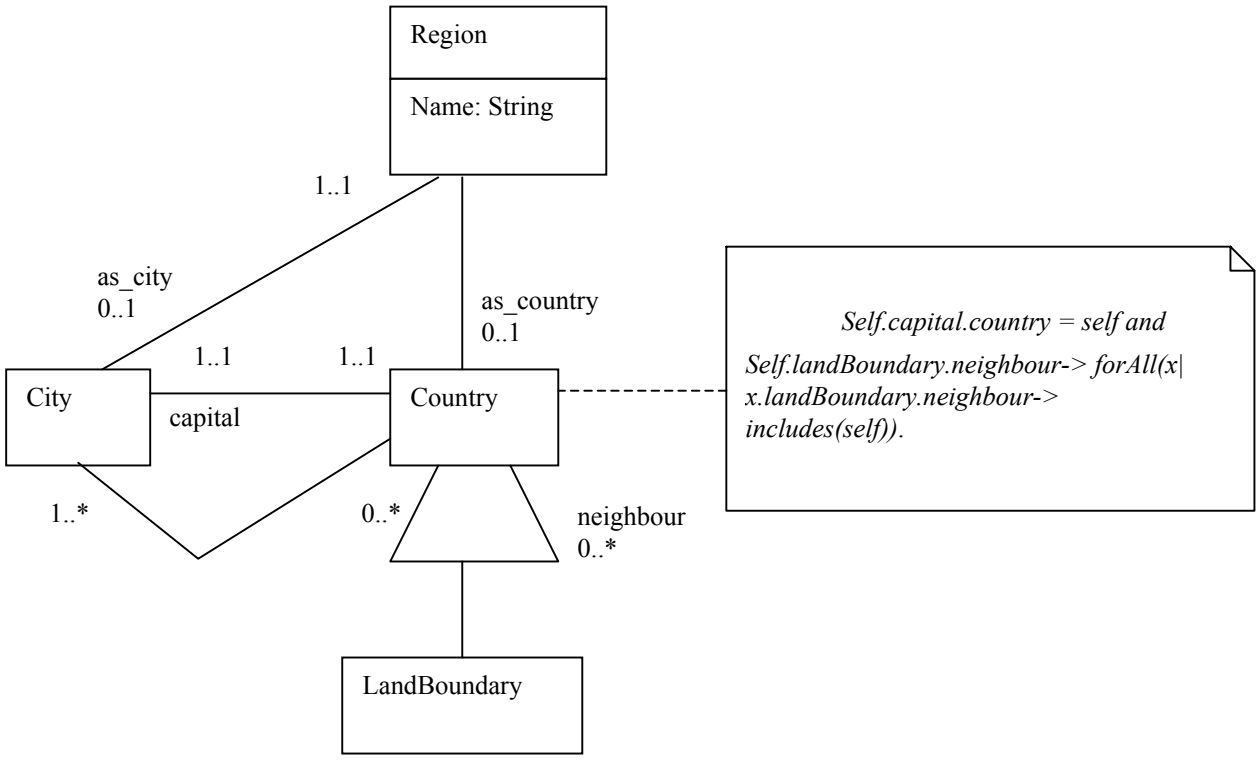


Figure 7. An example of UML ontology. Adapted from [16].

The following is an example of an OCL sentence taken from [16], and referring to "CIA World Factbook", a CIA ontology used to categorize geographical domains:

Self.capital.country = self and
Self.landBoundary.neighbour-> forAll(x | x.landBoundary.neighbour-> includes(self)).

The above formula gives some constraints on instances of the class *Country* and means: "a country's capital is a city in that country and, if a country *x* has another as a neighbour, then that neighbouring country has *x* as a neighbour".

Some conclusions about UML include the following. UML has a very large and expanding user community. Users of databases are generally familiar with this notation. Moreover, UML diagrams are based on a standard graphical representation which makes the knowledge base construction task easier. These facilities suggest a possible extension of "traditional" UML domains to cope with ontologies' issues. Cranefield and Purvis [17] suggest that UML is more suited to applications in which explicit

knowledge, represented by an ontology, is enough to satisfy knowledge representation needs. Vice versa, when a specific application needs to also compute deduced knowledge (implicit knowledge, in our definition), then OCL inferential mechanism semantics seem still to be ill-defined and unclear.

5. Knowledge Representation Methodologies specific for e-Learning

E-learning systems exploit knowledge representation methodologies seen in the previous chapters, adapting general methods to their information representation needs. In this chapter we provide an overview of Intelligent Tutoring Systems (ITSs), expert systems in the education domain, and in their knowledge representation methods.

There are various ITS proposals, each of which is structured in different ways, with different inference mechanisms and explicit knowledge representation approaches. ITSs are interested in describing two different types of knowledge: knowledge about their specific didactic domain (maths, languages, physics, and so on) and knowledge about students using the training platform, although not all ITS architectures provide both types of knowledge.

Hartley and Sleeman in [24] propose an architecture for ITSs composed of the following modules:

- An expert module on the subject domain, which is an expert system on the domain to be taught, capable of determining the correct solution to the problems proposed to the student during the teaching session.
- A module capable of building an explicit representation of the learning status of the student (*student model*) which allows the system to personalize teaching.
- A list of possible teaching operations.
- A set of means-ends guidance rules that execute teaching actions on the basis of the status of the student model.

This structure is based on the Production System approach, already described in Section 3.2.2. Indeed, the working memory here is composed of the student model, the production rules are the means-ends guidance rules, while teaching operations are the system actions.

Other traditional ITS architectures are shown in Figure 8 and 10. For example, Figure 8 shows Anderson and Reiser's proposal of ITS architecture, implemented in LISP Tutor and Geometry Tutor (about this last, we refer to Section 3.2.2 for some production rule examples). In Anderson and Reiser's approach, the student has the initiative, and the basic idea is to compare her/his actions with those of a domain expert, simulated by the system. For example, the system gives an exercise to the student and monitors her/his problem solving process. If the student's actions are not the same of the simulated domain expert, the system provides corrective feedback to the student. It is interesting to note, in this architecture, the presence of a "Bug Catalogue", i.e. a list of "misconceptions" about the domain or a set of the most common errors possibly made by students.

Knowledge about the domain and knowledge about the student profile can be modelled in various ways. Potentially, all the AI knowledge representation methodologies viewed in Chapters 3 and 4 can be adapted to this aim. Some of the most frequently used approaches to domain knowledge and student model representation are:

- FOL languages, used in conjunction with both production systems or programming language (Prolog-like) inference mechanisms to produce implicit knowledge.
- Epistemic logics, which enable the system to formalize different student models and to reason about them (we remark that epistemic knowledge reification mechanism is a instrument to express knowledge about knowledge, see Section 3.5). We refer to [15] for some examples of such systems.
- Semantic networks, in which knowledge is organized by means of attributes and objects' relationships rather than by using formulas. The growing interest in this area is proved by the international effort to standardize ontologies capable of treating didactic domains, as we will see in Chapter 7.

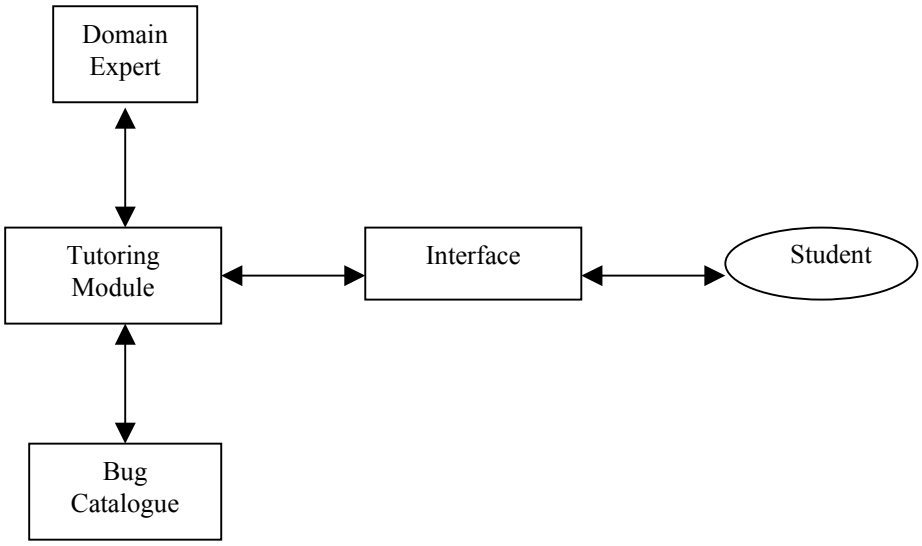


Figure 8. Anderson and Reiser's architecture. The student has the initiative, her/his actions are compared with those of the simulated expert: when they are different, the student is guided back to the expert's path. Stress is placed on the "mal-rules". Adapted from [34].

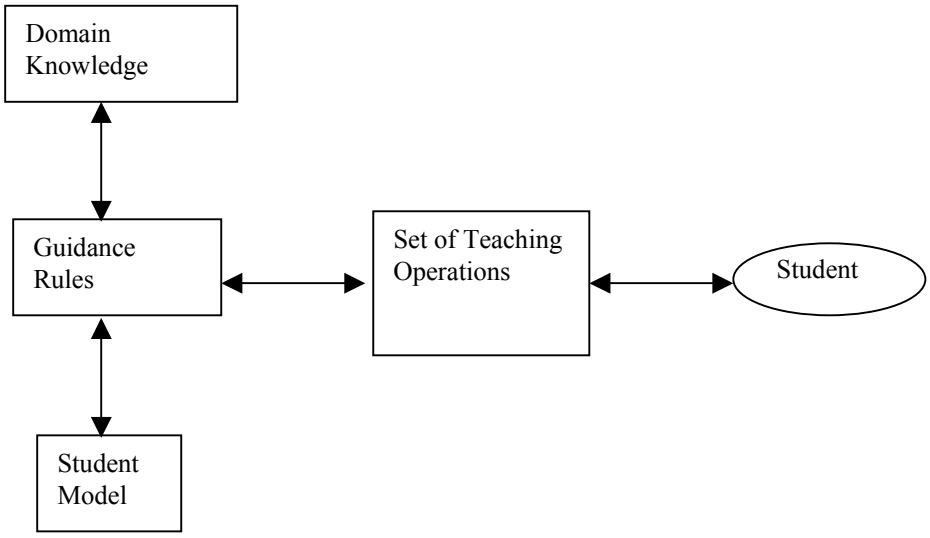


Figure 9. Hartley and Sleeman's architecture. In this architecture, the system guidance is strong. The core of the system is the student model, used as a "working memory" (see Section 3.2.2). The guidance rules take into account such a model. Adapted from [34].

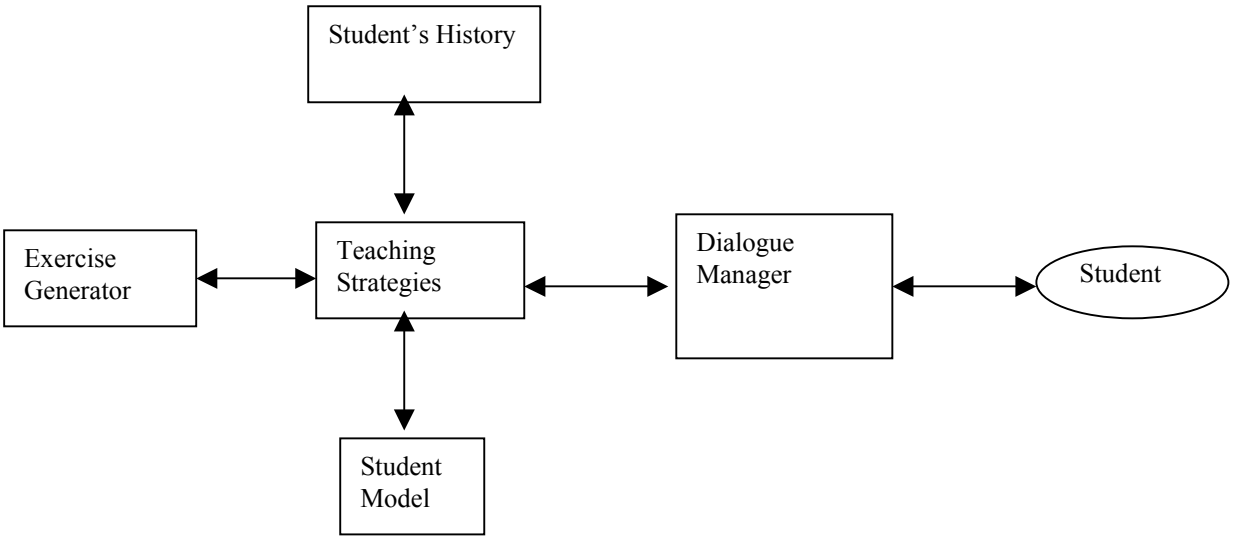


Figure 10. O'Shea's architecture. In this architecture stress is placed on a wide variety of teaching actions. Adapted from [34].

OKI

The Open Knowledge Initiative (OKI) [65] is being developed to support teachers who wish to use sophisticated and creative methods of online education, but are frustrated with available tools and products. Existing platforms are difficult to extend, either by building new educational tools on top or by linking the existing system to an institution's enterprise data services such as a digital library collection. OKI aims to develop "meaningful, coherent, modular, easy-to-use, web-based environments that are scalable, stable and open". These will allow teachers and learners to assemble, deliver and access educational resources regardless of time, place or platform. The principal outcome will be a standards-based reference framework, which specifies the systems components required for a pedagogically driven learning model.

Funded in part by a grant from the Andrew W. Mellon Foundation, and led by MIT, OKI is a collaborative project with the Universities of Stanford, Dartmouth, Harvard, North Carolina, Michigan, Pennsylvania, Wisconsin-Madison, Washington and Cambridge. OKI's partner institutions are playing important roles in defining the architecture and specifying the set of fundamental Web-enabled learning components that will provide standard course management features.

The OKI architecture will meet the challenges presented by the constant evolution of technology by using an innovative adaptation of a traditional software development technique. This involves collecting bits of functionality into a bundle and defining a limited number of ways to access that bundle. The adaptation of this traditional technique focuses on how the functionality is implemented. Traditionally, access to a function has been tightly bound to the actual implementation of that function. OKI's architecture will loosely bind access to a function with its implementation. In fact, it will be possible to change the implementation of a function that an educational application is executing. This dynamic binding of function access to implementation is at the heart of the OKI architecture. The definition of the access to a bundle of functionality is called an Application Programming Interface (API). Standards are also key elements of permanent solutions. OKI's philosophy is to anticipate where common services or standards will emerge; this enables interfaces to be structured in such a way that, over time, these emerging standards can be introduced without negatively impacting the other areas of the initiative.

An API gives a precise definition of a particular service within an application, so that a programmer can use that functionality without having to know how it's implemented.

There are several benefits that flow from an API approach. The most important benefit is that the work of building an application can take place independently of the services it will require from the API; the programmer doesn't need to know how the API-accessed service is implemented, including what kinds of network transport mechanisms are being used or even whether network connectivity exists at all. Another important benefit is that more than one implementation of a service is possible without requiring the application program to change. So long as an implementation of a service maintains the API, implementations can vary without requiring any changes in an application using the API.

Some examples of services that are familiar to most applications and suitable for an API, include database access, file access, authorization, authentication, messaging, logging, workflow, registration and so on. Examples of additional API services that might exist in OKI are publishing, collaboration, and assessment.

An API functions through the definition of the service it provides. To define an API it is necessary to define the objects that make up the service. For example, an Authorization API would contain objects such as a Person, a Function to be performed, a Qualifier that gives the context for performance of the Function, and the Authorization itself. Defining the objects involves not only naming them, but also determining what information they contain. To complete the definition of an object it is necessary to define how the information in an object can be accessed.

Definitions of objects associated with an API go a long way toward defining the rest of the API. For any collection of objects there is set of things that can be done with the objects. Examples of things that can be done with an Authorization API include: Determining who can perform which functions in a qualifier context, or determining who has authorization to perform a function.

Most of this work can be done outside the constraints of a programming language, but for a programmer to be able to use the API, it must be cast into a programming language. We will do this casting into Java; the API object definitions will become Java interfaces. The advantage of a Java interface is that it is well defined and the programmer can use it when writing the application, but it does not imply a specific implementation.

Functions performed with API objects are collected into a Java class that is called a Factory. The Factory collects Functions and binds them to an implementation of the API. The API objects are represented with Java interfaces. The Factory provides a way for the application to get instances of these API object interfaces. After an API is created, an application may access any implementation of the API through the Factory.

Architecture

The OKI architecture is founded on a number of features that will enhance its adaptability, and ensure a broad variety of uses for many different pedagogical systems.

By creating several distinct modules of functionality with well-defined interfaces, OKI gains several advantages. It makes it possible to more easily upgrade a specific functional area without impacting the entire system. Also entirely new modules of functionality can be added without negative impact on the system. The difficulty in this approach is to cleanly partition functionality.

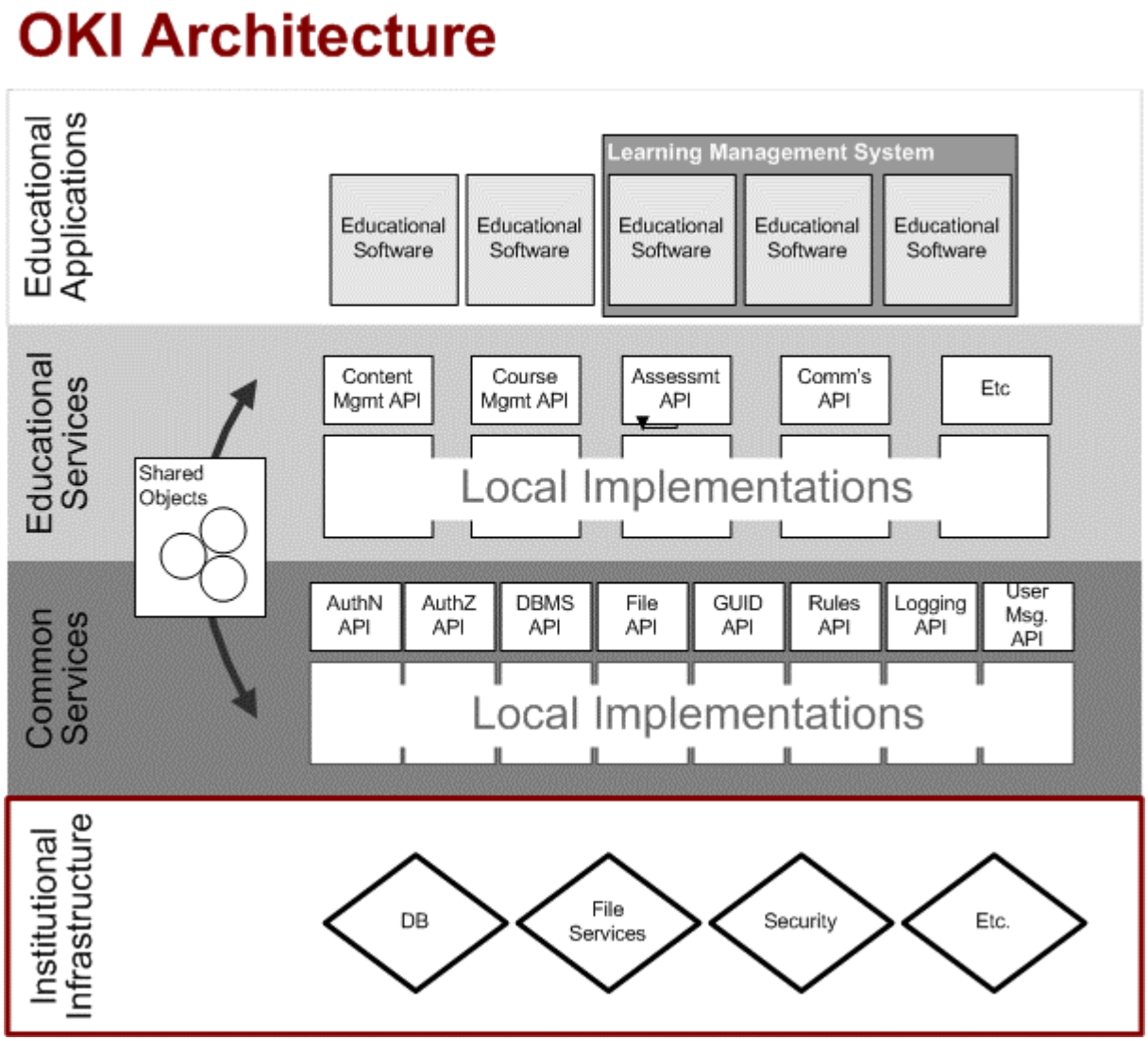


Figure 11. OKI architecture

Common Service APIs

Many application sectors need the same sets of functionality. For instance, there is a common need for authentication in most applications. But while there is a common need for authentication, there are many ways to provide it and several standards already exist in this area. In addition in an area such as authentication, many enterprises have enterprise wide systems that should be leveraged against enterprise applications.

It is through this Common Services API layer that OKI will allow for integration with enterprise infrastructure and help to assure adaptability to multiple and evolving standards and changing technology at this level. API modularity is designed to allow multiple OKI sites, say various schools at the same institution, to share some common infrastructure, like Authentication/Authorization, while keeping others local, like file management.

6. Standards for Knowledge Modelling

As we observed in Section 4.2, the importance of metadata and ontologies arises from standardization opportunities, which allow different applications to exchange data if their knowledge bases are described using the same language. For example, we can represent our world knowledge using a semantic network, in which attributes and relations (and their values) are expressed by means of a shared vocabulary of terms. This vocabulary is defined by standardization organizations and can be used by different applications to represent their data. Thus, although different applications can have different ways of inferring implicit knowledge (for example by ad hoc procedural methods which follow an ontologies' links), their own explicit knowledge (knowledge base) can be inquired by another application because both use the same terms and both understand the respective representation meanings.

This need is particularly important in Web applications, in which knowledge is distributed on the net and represented with different formats, making it increasingly difficult to find, access, present and maintain information. Tim-Berners-Lee [9, 10] envisioned a structure of the Web, called "the Semantic Web", in which all documents (Web resources) are described by means of metadata or ontologies expressed by means of a shared vocabulary of terms so to allow different applications to enquire each other. In this way, the Web can be consulted as if it were a centralized database, because all its knowledge is understandable by all computer-applications. The semantic web will provide intelligent access to heterogeneous and distributed information enabling software products (agents) to mediate between the user needs and the available information sources.

In this chapter we analyse some of the most promising standards for knowledge representation, both general-purpose oriented and specific to the didactic domain. In particular, the next section is dedicated to metadata standards, while the following focuses on metadata for Web applications.

6.1 Metadata Standards

In this section three standards are examined, Dublin Core, which is a general purpose metadata standard, and IMS and LOM, two metadata standards specific for didactic domain. Dublin Core is a rather diffused standard shared by different applications, while IMS and LOM are the main didactic metadata standards actually in competition between them to conquer a dominant position in the e-learning market.

6.1.1 Dublin Core

The Dublin Core Metadata Initiative (DCMI) is an organization born with the aim of defining a metadata standard composed of a small set of elements (hence the name "core") capable of describing the greatest number of Web resource typologies. Indeed, while several organizations dedicate their efforts to fix standards for specific application domains, Dublin Core is aimed at defining a minimal set of terms and categories usable "horizontally" by all domains to allow interoperability among applications coming from different areas.

The following are the main objectives of Dublin Core:

- *Development and management simplicity*. The set of elements is small and easy so that non-experts can build the records she/he needs easily and cheaply.
- *Shared semantics*. Information retrieval is sometimes difficult due to the differences in terminology and descriptive techniques among knowledge domains. Hence, the DCMI has established a shared set of elements, the semantics of which are universally clear, in order to augment the accessibility to all kinds of resources independently from their original field.
- *International definition*. Initially the DCMI developed the set of elements of Dublin Core in English. Successively, the standard was translated into more than 20 different languages.

The Dublin Core standard consists of exactly 15 different elements, each of which is optional and can be instantiated multiple times. Each element is an attribute of the metadata (see Section 4.2.1). There is no constraint on the order to use when we instantiate the attributes, and this rule is valid also for multiple instantiated attributes. The admissible set of values for some of the attributes of Dublin Core

belongs to a shared vocabulary containing an accurately selected set of terms. A shared vocabulary of terms for attribute values is important to improve search results, avoiding ambiguity errors due to aliases describing the same concept.

The elements (attributes) of Dublin Core (as specified in Dublin Core Version 1.1), are listed in Table 4. All data types must be strings containing English text.

Identifier	Definition	Comment
Title	A name given to the resource.	**Title** is the formal name of the resource.
Creator	The entity which is the principal creator of the resource.	A **Creator** can be a person, an organization, a service, …
Subject	The subject of the resource.	**Subject** can be expressed by means of key words or key sentences or classification codes. It is suggested to choose its value from a formal defined vocabulary.
Description	An explanation concerning the resource content.	For instance, a **Description** can be: an analytical summary, an index, a link to a graphic representation, text about the content, …
Publisher	The entity which is the actual publisher of the resource.	A **Publisher** can be a person, an organization, a service, …
Contributor	The entity which is a minor creator of the resource.	A **Contributor** can be a person, an organization, a service, …
Date	A date associated with an important event of the life-cycle of the resource.	Normally, **Date** is associated with the creation or the availability of the resource. For the value of this attribute is suggested to use the following format: YYYY-MM-DD[1]. If one wants to use another format, this must be univocally identified.
Type	The nature or the kind of the resource content.	**Type** includes terms describing general categories. It is suggested to choose this from a formal vocabulary: for example, the Dublin Core Types list (DCT1).
Format	The physical representation format of the resource.	Normally, **Format** includes the resource support type (software, hardware, …) or its size.
Identifier	The resource identifier in a given context.	It is suggested to identify the resource by means of a sequence of characters belonging to a formal identification system (URL, DOI, ISBN).
Source	A link to a resource from which the present resource is derived.	The present resource can be derived from **Source** in a complete or partial way. It is suggested to identify the resource by means of a sequence of characters belonging to a formal identification system.
Language	The language in which the resource content is described.	It is suggested to represent the values of the attribute **Language** following the format defined in RFC 1766[2].
Relation	A link to a related resource.	It is suggested to identify the resource by means of a sequence of characters belonging to a formal identification system.
Coverage	The resource scope.	Normally, **Coverage** includes a spatial location (the name of a place or its geographic coordinates) or a temporal range. It is suggested to take the value of the attribute from a formal vocabulary (for example, TGN).
Rights	Copy-right information.	Normally, **Rights** contains information about the resource management rights. If **Rights** is not instantiated, no hypothesis is made about resource copy-rights.

Table 4. The Dublin Core elements.

1 See standard ISO 8601, defined in http://www.w3.org/TR/NOTE-datetime.

2 RFC 1766 is available at: http://www.ietf.org/rfc/rfc1766.txt.

6.1.2 IMS

The IMS Global Learning Consortium proposes a set of integrated standards for e-Learning subjects. The first and most important of these is IMS Metadata Specification, describing generic didactic subjects. Referring to Chapter 5, this standard (called an "horizontal" standard) concerns the representation, through a metadata approach, of all the system knowledge, especially the Domain Knowledge, as provided, for example, by Hartley and Sleeman's architecture. Other IMS "vertical" standards concern specific system knowledge modules, such as the Student Module (IMS LIP) or the Student Interface (IMS QTI, which aim is to standardize tests). The following is a description of IMS Metadata Specification standard.

As usual, a metadata standard defines a set of categories in which it groups knowledge about individual objects of its domain. Below we show the categories proposed by IMS Metadata Specification about didactic domain, using the following notation: a *Learning Object* (LO) is an atomic object of the didactic domain (namely, of the system Domain Knowledge). A Learning Object is a generic resource such as a piece of knowledge represented by a textual document, or a slide, or a graphic, and so on.

Unlike Dublin Core, the IMS Metadata Specification is hierarchically structured. Indeed, a metadata schema is composed of a set of *elements* which can be recursively composed of sub-schemas or of atomic attributes the values of which can be defined by primitive *data types* (strings, numbers, …) or by terms taken from a *shared vocabulary*.

Thus, the metadata schema definition seen in Section 4.2.1 ($M = \{A_1, ..., A_n\}$) is extended in a hierarchical structure as follows:

$M = \{M_1, ..., M_n\}$,

Where each M_i can be an attribute:

$M_i = A_i$ (for its formal definition, see Section 4.2.1),

or a metadata itself:

$M_i = \{Mi_1, ..., Mi_m\}$.

The main IMS categories (i.e., the attributes of the root of the schema: M_i) and some examples of their fields (sub-schemas, or attributes at lower levels of the hierarchic metadata definition tree: Mi_j) are shown in the following:

1. **General**: groups general information needed to completely describe a *LO*, for example, the field *identifier* will contain the *LO* ID, *title* the *LO* name, *language* indicates the language in which it is possible to utilize the *LO*, and so on.
2. **Life-Cycle**: describes the resource history, its actual state, the last update, …
3. **Metametadata**: describes the resource description rather than the resource itself.
4. **Technical**: groups the *LO* technical features, such as: the field *format* to indicate the data type (video, mpeg application, text, html, and so on); *size* specifies the *LO* size; *location* its URL, and so on.
5. **Educational**: groups a set of attributes about the *LO* educative and pedagogical features.
6. **Rights**: describes general copyright features.
7. **Relation**: indicates some features of the *LO* with respect to other *LOs*.
8. **Annotation**: contains comments about the resource.
9. **Classification**: describes the *LO* features by means of some classifications.

6.1.3 LOM

LOM stands for *Learning Object Metadata* and is a standard defined by the *Learning Tecnology*

Standardization Committee (LTSC)[3] of IEEE. As in the IMS Metadata Specification, the LOM standard has a hierarchical structure. Each of its elements can be *mandatory*, *optional* or *conditional* (in this case, the attribute values depend on the presence or absence of a value for another optionally defined attribute). Some attributes can have, as a value, a list of terms rather than a single term. This list can be *ordered* or *unordered*. Finally, the shared vocabularies of terms of the metadata values can be defined as: *restricted* (only the values of the vocabulary are admissible values for that attribute) or *open* (the vocabulary groups a suggested list of terms frequently used by applications but also other terms are admissible values for that attribute).

The main elements of LOM schema actually are 47 (against the only 15 of Dublin Core) and are grouped in 9 categories as shown in Table 5.

Category	Description	Elements
General	Groups all context independent features together with the semantics descriptors of the resource.	Identifier, Title, CatalogEntry, Language, Description, Keywords, Coverage, Structure, Aggregation Level.
Life Cycle	Groups the features related to the life cycle of the resource.	Version, Status, Contribute.
Meta Metadata	Groups the features of the description itself rather than the resource which is described.	Identifier, CatalogEntry, Contribute, Metadata Scheme, Language.
Technical	Groups the technical features of the resource.	Format, Size, Location, Requirements, Installation Remarks, Other Platform Requirements, Duration.
Educational	Groups the didactic and pedagogical features of the resource.	Interactivity Type, Learning Resource Type, Interactivity Level, Semantic Density, Intended End User Role, Context, Typical Age Range, Difficulty, Typical Learning Time, Description, Language
Rights Management	Groups the resource features depending on the type of use we plan for it.	Cost, Copyright and Other Restrictions, Description.
Relation	Groups the resource features which relate it to other resources.	Kind, Resource, Identifier, Description, CatalogEntry.
Annotation	Allows comments to be made about the resource's educational content.	Person, Date, Description.
Classification	Describes a particular classification system in which the resource is defined (such as taxonomies, conceptual graphs, and so on).	Purpose, TaxonPath, Description, Keywords.

Table 5. LOM categories.

6.1.4 SCORM

The SCORM (Sharable Content Object Reference Model) [44] is a set of specifications for developing, packaging and delivering education and training materials whenever and wherever they are needed. SCORM is a product of the U.S. Government's initiative in **Advanced Distributed Learning** (ADL) [45], which aims to provide access to high-quality materials that are easily tailored to individual learner needs.

[3] Concerning **LTSC**, we refer to http://ltsc.ieee.org.

The SCORM applies current technology developments – from groups such as the IMS Global Learning Consortium, the Aviation Industry Computer-Based Training Committee, the Alliance of Remote Instructional Authoring and Distribution Networks for Europe (ARIADNE) and the Institute of Electrical and Electronics Engineers (IEEE) Learning Technology Standards Committee (LTSC) – to a specific content model to produce recommendations for consistent implementations by the vendor community.

Although choice and competition in the marketplace are generally a good thing, the rapid growth of LMS vendors has created a significant problem for content authors. Without a common specification for packaging online courses, LMS vendors organize their content databases in any fashion they choose. As a result, every vendor is using a different format for packaging their courses. Although they are all delivered by http, or some other standard protocol, if an author tries to move a course from one LMS to another, they find that this task is very time-consuming. In many cases, moving from one LMS vendor to another requires complete reconstruction of the course materials.

ADL is evolving a set of specifications for packaging online courses that will not only make it easier to transport a course from one LMS to another, it will also achieve other desirable goals as well. A course packaged following the SCORM specifications can be transported from one LMS to another with minimum modifications. SCORM-compliant courses leverage course development investments by ensuring that compliant courses are:

Accessible: the ability to locate and access instructional components from one remote location and deliver them to many other locations.

Interoperable: the ability to take instructional components developed in one location with one set of tools or platform and use them in another location with a different set of tools or platforms.

Durable: the ability to withstand technology changes without redesign, reconfiguration or recoding.

Reusable: the flexibility to incorporate instructional components in multiple applications and contexts.

As shown in Figure 12, all of the specifications and guidelines contained or referenced can be viewed as separate “books” gathered together into a growing library.

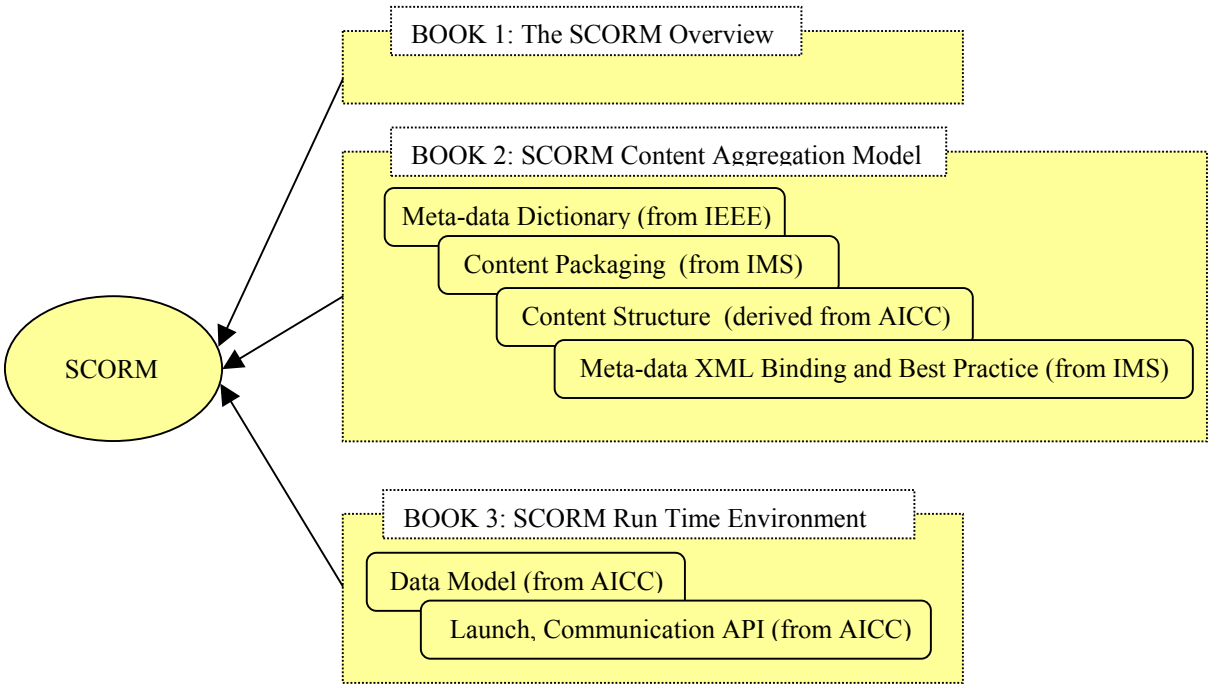


Figure 12. The SCORM as a collection of specifications.

These technical "books" are presently grouped under two main topics: "Content Aggregation Model" and "Run-time Environment".

The purpose of the SCORM Content Aggregation Model is to provide a common means for composing learning content from discoverable, reusable, sharable and interoperable sources. The SCORM Content Aggregation Model further defines how learning content can be identified and described, aggregated into a course or portion of a course and moved between systems that may include Learning Management Systems (LMS) and repositories. The SCORM Content Aggregation Model defines the technical methods for accomplishing these processes. The model includes specifications for aggregating content and defining metadata.

The purpose of the SCORM Run-time Environment is to provide a means for interoperability between Sharable Content Object-based learning content and Learning Management Systems. A requirement of the SCORM is that learning content be interoperable across multiple LMSs regardless of the tools used to create the content. For this to be possible, there must be a common way to start content, a common way for content to communicate with an LMS and predefined data elements that are exchanged between an LMS and content during its execution.

6.1.5 Edutella

Peer-to-peer (P2P) networks are a relatively new method of sharing information across distributed, heterogeneous networks, without the requirement of central point of control. The Edutella project [53] was begun in response to the fact that information in P2P networks is no longer organised in navigable hypertext-like structures, but is stored on numerous peers waiting to be queried. We need to know what we want to retrieve and which peer is able to provide the resources in order to submit a query [54]. In some cases, especially concerning the exchange of educational resources, queries are complex, building on standards like IEEE-LOM/IMS metadata with multiple metadata entries.

To complicate matters further, concentrating on domain specific formats has meant that current P2P implementations are fragmenting into niche markets rather than developing unifying mechanisms for future P2P applications. There is, in fact, a danger that unifying interfaces and protocols introduced by the WWW will become lost in the forthcoming P2P arena.

The Edutella project addresses these shortcomings of current P2P applications by building on the W3C metadata standard RDF. The project is a multi-staged effort to scope, specify, architect and implement an RDF-based metadata infrastructure for P2P-networks. To enable interoperability and reusability of educational resources, the infrastructure must be flexible enough to accommodate complex and varying metadata sets, and avoid creating another special purpose application suitable only for a specific application area which is outdated as soon as metadata requirements and definitions change.

The aim is to provide the metadata services required to enable interoperability between heterogeneous JXTA (Juxtapose) applications. The first application will focus on a P2P network for the exchange of educational resources (using schemas like IEEE LOM, IMS, and ADL SCORM to describe course materials).
JXTA is a set of XML-based protocols covering typical P2P functionalities. It provides a Java binding offering a layered approach for creating P2P applications. JXTA enables remote service access and provides additional P2P protocols and services, such as peer discovery, peer groups, peer pipes, and peer monitors.

In a typical P2P-based scenario in which learning resources are being exchanged, universities have two roles:

- Content provider. They do not lose control over their learning resources but still provide them for use within the network.
- Content consumer. Teachers and students benefit from having access not only to a local repository, but also to a whole network, using queries over the metadata distributed within the network to retrieve required resources.

Edutella connects highly heterogeneous peers, each making its metadata information available as a set

of RDF statements. By implementing a set of Edutella services, the distributed nature of the individual RDF peers connected to the network can be made completely transparent. Edutella services include a query service, replication, annotation and mapping:

- Query service. The query service is intended to be a standardised query exchange mechanism for RDF metadata stored in distributed RDF repositories and serves as both a query interface for individual RDF repositories located at peers, and as a query interface for distributed queries spanning multiple RDF repositories.
- Replication service. This service complements local storage by replicating data in additional peers to achieve data persistence and availability while maintaining data integrity and consistency. Replication of metadata is the initial focus.
- Mapping, mediation and clustering service. While groups of peers will usually agree on using a common schema (e.g. SCORM or IMS/LOM for educational resources), extensions or variations may be needed in some locations. The purpose of the Edutella Mapping service is to manage mappings between different schemata and use them to translate queries over one schema to queries over another schema. Mapping services will also provide interoperation between RDF- and XML-based repositories. Mediation services actively mediate access between different services, while clustering services use semantic information to set up semantic routing and semantic clusters.
- Annotation service. In order to be applicable in a wide range of application scenarios, the annotation tool must be independent from any particular domain, and support a wide range of semantic definitions.

6.2 Standards Related to the Semantic Web

As mentioned at the beginning of this chapter, standardization issues are particularly important for those knowledge bases thought to be inquired on the Web, because of the impossibility of fixing the same application valid for all the nodes of the net. In this section we will see some knowledge representation standards appositely thought with the aim of facilitating Web interoperability.

6.2.1 XML and DTDs

The World Wide Web Consortium (W3C) is the entity, founded by Tim Berners-Lee, which is interested in standardization issues related to the Web. Among other standards, W3C formulated the specifics of HTML, XML and RDF. HTML makes information processable by machines but not understandable; the mark-up only covers the way it should be presented on the page. However, the semantic web with machine processable contents will only be possible when further levels of interoperability are established. Standards must be defined not only for the syntactic form of documents, but also for the form of their semantic contents. W3C's standardisation efforts facilitate this semantic interoperability. XML can be seen as an extension of HTML in which tags can be user-defined, while RDF is the proposal for a general-purpose metadata (such as Dublin Core) based on XML language.

Let us, first of all, give an overview of the main features of XML.

XML was proposed as an extensible language allowing the user to define his own tags in order to indicate the type of content annotated by the tag. This enables the exchange of data in a structured way. The advantage of a textual representation of structured (symbolic) data arises from the possibility of looking at the data without the program which produced it. In this way we can achieve two objectives: the data in text format is human-readable (programmers can read data with debug purposes) and applications different from the parent of the data can read and use them.

XML syntax is similar to HTML. The main difference concerns the possibility, in XML, of using tags (words bracketed by "<" and ">") and attributes (expressed with the following format: *name="vale"*) whose semantics are completely defined by the applications that read the data.

DTD (document type definition) specifies the elements that can be used in an XML document. In the document, the elements are delimited by start and end tags. It has a type and may have a set of attribute specifications consisting of a name and a value. The additional constraints on a DTD refer to the logical

structure of the document, especially the nesting of tags inside the information body that is allowed and/or required.

There is a growing set of optional modules around XML that provide set of pre-defined tags and attributes for specific tasks. Among these, particularly important is XML Namespace, a specification which describes how you can associate a URL with a single tag or attribute of an XML document. As usual, the semantics of the URL strictly depend on the application which reads the document. As we will see, RDF metadata utilizes URLs in linking pieces of metadata to a file defining the type of that data.

Coding symbolic data in textual format may seem like a bad idea, from the point of view of space occupancy. Nevertheless, textual data can easily be compressed by commercial programmers such as zip or gzip. Indeed, the aim of XML code is not compression but the possibility of fixing a common syntax for all Web knowledge representation applications.

XML is purely structural in nature and describes data on the object level. We have to look at other approaches if we want to describe information on the meta level and to define its meaning. In order to fill this gap, the RDF standard has been proposed as a data model for representing metadata about web pages and their content using an XML syntax.

OPML [46] is essentially an XML format for storing outlines, and outlines are also "node-labelled" hierarchical structures, but it has an advantage over straight XML in that it restricts the "labels" on the nodes to be stored in a single attribute value. This might not be an advantage for some types of metadata, but it greatly simplifies things when you only want to represent a hierarchical structure. It's about the simplest node-labelled structure you can get.

Rich Site Summary (the name for RSS 0.91, which is the most widely-used version of RSS currently) is another XML-based node-labelled hierarchical format, similar to OPML, but it uses tag names to represent various metadata [38]. The metadata that are permitted to be used are laid out in the specifications, also in the Really Simple Syndication (RSS 0.92) specification. The metadata defined by these specifications are fairly general, but nevertheless define a vocabulary that needs to be used by interoperable implementations. It's a step more restrictive than OPML, but the act of defining some common metadata allows tools to count on the meanings of those metadata and build services. The metadata defined by RSS are meant to be meaningful primarily to syndication and aggregation tools. RSS 1.0 uses RDF to describe basically the same things as the original syntax, with a few additions. All of the RSS family of specifications are designed for tools that do syndication and aggregation. You can certainly put RDF and OPML/XML to other metadata uses besides just syndication and aggregation.

6.2.2 RDF and RDFS

W3C proposed the *Resource Description Framework* (RDF), a general-purpose metadata standard for cross-discipline applications. An RDF *resource* is any object uniquely identifiable by a Uniform Resource Identifier (URI). RDF extends XML to be specific for describing resources.

RDF is designed to facilitate semantic interoperability by representing a domain model in terms of simple object-relation-object triples, and techniques from Knowledge Representation can be used to help find mappings between two RDF descriptions. Of course this does not solve the general interoperability problem (i.e., finding semantic-preserving mappings between objects), but the usage of RDF for data interchange raises the level of potential reuse much beyond the parser reuse which is all that one would obtain from using plain XML. Moreover, since RDF describes a layer independent of XML, an RDF domain model (and software using the RDF model) could still be used, even if XML syntax changes or becomes obsolescent.

One of the most important peculiarities of RDF is that it provides the ability to unambiguously identify the semantics of shared vocabularies of terms used to describe data. This is done through the XML Namespace mechanism, which uniquely establishes the governing authority of the vocabulary. Thus, if you read the attribute ("property") *author* in RDF metadata, through the Namespace mechanism it is possible to understand its semantic simply by checking the vocabulary source.

Example.

```
<? xml version=”1.0” ?>
<RDF xmlns = http://w3.org/TR/1999/PR-rdf-syntax-19990105#
     xmlns:DC = http://purl.org/DC# >
     <Description about = http://dia.uniroma3.it/report.html >
       <DC:Title> Didactic Knowledge Representation Methodologies </DC:Title>
       <DC:Creator> John Smith </DC:Creator>
       <DC:Date> 2001-12-10 </DC:Date>
       <DC:Subject> Knowledge Representation, Didactic Domains, Standards, Logic,Semantic
       networks </DC:Subject>
     </Description>
</RDF>
```

The first line of the above example simply indicates that this is an XML document. The next line indicates two namespaces, respectively, for RDF and Dublin Core (DC) metadata. In this way we can inherit metadata schemas from multiple standard vocabularies. In our example, all the properties in the description will come from either RDF or Dublin Core standards.

The main section of the example (between <Description> tags) shows four properties that describe the resource pointed to by the URI in the *about* attribute in the <Description> tag. These properties come from Dublin Core namespace. In the example, case, Title, Creator, Date and Subject attributes are indicated for the resource at <= *http://dia.uniroma3.it/report.html* >.

By adding other XML namespaces in the second line of the description of the above example, it is possible to import other metadata schema. Thus, a RDF description can inherit attributes from different vocabularies, each of which is specified through its namespace URI.

If people want to share their resource descriptions, there has to be an agreement on a standard core of vocabulary in terms of modelling primitives that should be used to describe metadada. RDF schemas (RDF/S) attempt to provide this standard vocabulary. RDF provides a syntactical convention and a simple data model for representing machine-processable semantics of data; RDFS defines basic ontological modelling primitives on top of RDF. RDF schemas provide a notion of concepts (class), slots (property), inheritance (SubClassOf) and range restrictions (ConstraintProperty). However, at the moment, the semantics are not well-defined. It provides a common syntax + basic vocabulary, but needs an additional "logical level" that defines a clear semantics for RDF expressions and provides a basis for integration mechanisms.

In their current states neither XML nor RDF provides sufficient support for the integration of heterogeneous structures or different meanings of terms. The Semantic Web won’t be possible until agents have the means to figure out some things by themselves, given the data they have to work with. Fortunately, artificial intelligence gives us two tools to help make this possible. First, knowledge representation is a field that defines how we might represent, in computers, some of what is stored in our brains. This would give computers a chance of synthesising unclassified data at a useful speed. Second, inference is a way of using formal logic to approximate further knowledge from that which is already known. All of this forms a system of representing and synthesising knowledge htat is often referred to as an ontology. A further representation layer is needed on top of the currently available layers.

For sharing information and knowledge (i.e. for interoperability) between different applications, a shared set of terms describing the application domain with a common understanding is needed. More flexibility is gained if not just a flat set of terms is defined, but also the relationships between these terms. Such a set of terms is called an "ontology".

A broker has access to information (metadata) describing data sources registered with the broker. When a user submits a request for information, the broker uses this metadata to check which data source is

able to handle the request. If there is a match, the broker establishes a connection between the user and the source. This semantic mapping can be performed using ontologies (detailed descriptions).

For each data source, the respective ontology holds information about the underlying technology and concepts. Data sources A and B may be linked by automatically matching the concepts and terms described in Ontology A with those described in Ontology B. This way, no static filter function Filter A-B is needed (filters contain translation rules for mapping one database to another). The main advantages of this approach are:

- each data source has one ontological description, and the growth in the number of ontologies needed to link multiple data sources is linear, e.g. to link 10 data sources, 10 ontologies are needed.
- if a data source changes, only the respective ontology has to be updated.
- even without an update of the respective ontology, the semantic mapping with respect to the concepts and terms specified for the "old" data source remain functional.
- to integrate a new data source into an existing network of data sources, the new data source simply has to be registered with the broker. No new filters have to be written.
- implicit knowledge hidden in the data sources can be made explicit through automatic inference. This applies also to knowledge that is the result of the combined knowledge stored in many distributed data sources. [Buster]

Using the syntax of XML and RDF various proposals for ontologies of Web contents have been formulated. In the next two sections we will see two important and diffused cases.

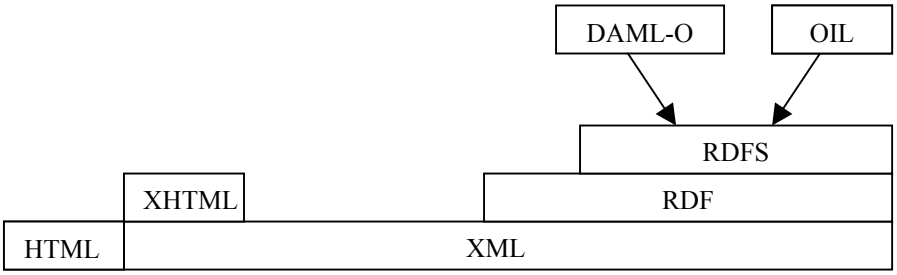


Figure 13. The layer language model for the WWW (Fensel, The Semantic Web and its Languages)

6.2.3 SHOE

Simple HTML Ontology Extensions (SHOE) is a language for knowledge representation based on the ontology approach (see Section 4.2.2) and XML syntax. Like XML, SHOE is also an extension of HTML.

In the present work on knowledge representation methods, we have often underlined the necessity of choice, in defining a representation formalism, between large expressive power or good computational costs. SHOE is not as expressive as other ontology systems such as Ontolingua, based on the logic language KIF (see Section 4.2.2). On the contrary, it is quite simple and has a tractable computational complexity.

As in the case of RDF, it is also possible in SHOE to include, in a same description, different ontology sources specifying their URL, as shown in the example below.

```
<HTML>
   <HEAD>
      <!—The ontology is consistent with SHOE 1.0 -->
      <META HTTP-EQUIV="SHOE" CONTENT="VERSION=1.0">
   </HEAD>
   <BODY>
      <!-- Name and version of the ontology we are defining -->
      <ONTOLOGY ID="dip-ontology" VERSION="1.0">
         <!-- Names and URLs of the ontologies we are including -->
```

```
                    URL="http://www.cs.umd.edu/projects/plus/SHOE/base.html">
```

The example shows the definition of an ontology named "dip-ontology", aimed at describing the domain of Academic Departments. Dip-ontology refers to the basic SHOE ontology through the attribute USE-ONTOLOGY ID, and each element of the basic ontology will be indicated using the prefix "base" (specified by means of the attribute *PREFIX="base"*)

We can establish the attribute of our ontology as shown below.

```
        <DEF-CATEGORY NAME="Organization" ISA="base.SHOEEntity">
        <DEF-CATEGORY NAME="Person" ISA="base.SHOEEntity">
        <DEF-CATEGORY NAME="Publication" ISA="base.SHOEEntity">
        <DEF-CATEGORY NAME="Research Group" ISA="Organization">
        <DEF-CATEGORY NAME="Department" ISA="Organization">
        <DEF-CATEGORY NAME="Worker" ISA="Person">
        <DEF-CATEGORY NAME="Student" ISA="Person">
        <DEF-CATEGORY NAME="Professor" ISA="Worker">
        <DEF-CATEGORY NAME="Researcher" ISA="Worker">
```

We have defined some classes (*Organization, Person, Publication*, *Research Group*…) and given them a hierarchic taxonomy using the relation ISA (semantic networks define subclass relationships in a similar manner: see Section 4.1.1).

Note that *Organization, Person* and *Publication* are subclasses of the class SHOEEntity declared in *base-ontology*. Indeed, the SHOE ontology extension mechanism allows the inheritance of class definitions from other ontologies, referred to by means of prefixes.

```
        <DEF-RELATION NAME="member">
           <DEF-ARG POS="1" TYPE="Organization">
           <DEF-ARG POS="2" TYPE="Person">
        </DEF-RELATION>
        <DEF-RELATION NAME="author">
           <DEF-ARG POS="1" TYPE="Publication">
           <DEF-ARG POS="2" TYPE="Person">
        </DEF-RELATION>
        <DEF-RELATION NAME="tutor">
           <DEF-ARG POS="1" TYPE="Professor">
           <DEF-ARG POS="2" TYPE="Student">
        </DEF-RELATION>
        <DEF-RELATION NAME="name">
           <DEF-ARG POS="1" TYPE="base.SHOEEntity">
           <DEF-ARG POS="2" TYPE="base.STRING">
        </DEF-RELATION>
      </ONTOLOGY>
   </BODY>
</HTML>
```

Now we have added to our ontology a set of simple relations between categories. Note that, in this case, some definitions refer to *base-ontology*.

Suppose, now, we want to instantiate this ontology schema using it in the description of a document. Let us assume the document concerns Nicola Capuano, a student in the "Dipartimento di Informatica dell'Università di Salerno", whose professor is Peter L. Found and who is the co-author of the paper: "Le Ontologie".

Suppose student Nicola Capuano has a registered home page at the URL **`http://www.unisa.it/capuano/index.html`** described by this HTML code:

```
<HTML>
   <HEAD><TITLE> My Home Page </TITLE></HEAD>
   <BODY>
      <P> This is my Home Page. I am Nicola Capuano, a student of the

      <P> This is a link to my professor's page:
          <A HREF="http://www.unisa.it/found"> tutor </A>.
      <P> Moreover, I published the following paper:
          <A HREF="http://www.unisa.it/capuano/paper.pdf"> Le Ontologie </A>.
   </BODY>
</HTML>
```

To index the above HTML page, it is enough to add, somewhere in the BODY section, the following code.

```
1.    <INSTANCE KEY="http://www.unisa.it/capuano/index.html">
2.       <USE-ONTOLOGY ID="dip-ontology" URL="http://www.unisa.it/dip-ontology.html"
                       VERSION="1.0" PREFIX="dip">
3.       <CATEGORY NAME="dip.Student">
4.       <RELATION NAME="dip.name">
            <ARG POS=1 VALUE="me">
            <ARG POS=2 VALUE="Nicola Capuano">
         </RELATION>
5.       <RELATION NAME="dip.member">
            <ARG POS=1 VALUE="me">
            <ARG POS=2 VALUE="http://www.unisa.it">
         </RELATION>
6.       <RELATION NAME="dip.tutor">
            <ARG POS=1 VALUE="http://www.unisa.it/found">
            <ARG POS=2 VALUE="me">
        </RELATION>
7.       <RELATION NAME="dip.author">
            <ARG POS=1 VALUE="me">
            <ARG POS=2 VALUE="http://www.unisa.it/capuano/paper.pdf">
        </RELATION>
      </INSTANCE>
```

The first thing to do is to uniquely identify the instance (the HTML document) we want to describe with our ontology. To ensure instances can be uniquely referenced, SHOE associates each instance with a specific URL. In our case, if we suppose that the home page is available at:

http://www.unisa.it/capuano/index.html,

then we must use a *tag* of the type `INSTANCE` as shown in line 1 of the example above. The next line specifies the ontology we will use to describe the home page (*dip-ontology*, just defined), stating that it is available at:

http://www.unisa.it/dip-ontology.html.

Then, line 3 declares the actual instance as a member of the class Student of *dip-ontology*. Furthermore, lines 4-7 declare relationships between the instance and other instances or data.

Finally, the following is an interesting example of the inference rule expressed with SHOE syntax. It states that: "If a student is the author of a paper, then her/his tutor is also an author of the same paper".

```
 <DEF-INFERENCE DESCRIPTION = "author(?tut, ?pap) if author(?stu, ?pap) and tutor(?tut,
?stu)">
    <INF-IF>
       <RELATION NAME="author">
          <ARG POS=1 VALUE="stu" USAGE=VAR> <ARG POS=2 VALUE="pap" USAGE=VAR>
       </RELATION>
       <RELATION NAME="tutor">
          <ARG POS=1 VALUE="tut" USAGE=VAR> <ARG POS=2 VALUE="stu" USAGE=VAR>
```

```
   </INF-IF> <INF-THEN>
      <RELATION NAME="author">
         <ARG POS=1 VALUE="tut" USAGE=VAR> <ARG POS=2 VALUE="pap" USAGE=VAR>
      </RELATION>
   </INF-THEN>
</DEF-INFERENCE>
```

The above examples show the expressive power of SHOE. As we can see, it is not only metadata, composed of a set of attributes, but it also allows the definition of relationships between the individual objects and the classes of the domain. It inherits, then, the expressive power of semantic networks characteristic of the ontology-based knowledge representation methods.

6.2.4 OML & CKML

OML (Ontology Markup Language) and CKML (Conceptual Knowledge Markup Language) are ontology specification languages developed at Washington University [47, 48]. They are based on description logics and conceptual graphs. The architectural structure of the languages is visualised in Figure 14.

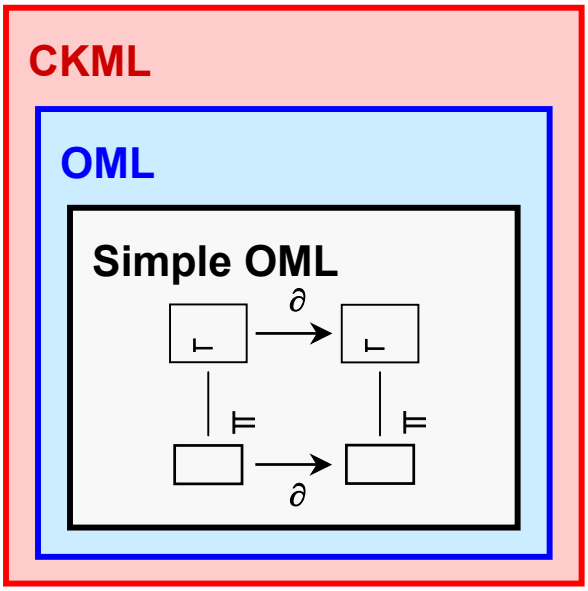


Figure 14. OML/CKML at a glance

Simple OML was designed for interoperability, offering the closest approach within OML to RDFS, while retaining the underlying principles of CKML. In addition to the central core of CKML, Simple OML represents functions, reification, cardinality constraints, inverse relations and collections. The first-order form of Simple OML is closely related to the Resource Description Framework with Schemas, and the higher-order form of Simple OML is intimately related to XOL (XML-Based Ontology Exchange Language), an XML expression of Ontolingua with the knowledge model of Open Knowledge Base Connectivity (OKBC).

OML represents ontological and schematic structure. Ontological structure includes classes, relationships, objects and constraints. The ability of a knowledge representation language to express constraints is a very important issue. OML has three levels for constraint expression:

Top – sequents

Intermediate – calculus of binary relations

Bottom – logical expressions

The top level models the theory constraints of information flow, the middle level arises both from the

practical importance of binary relation constraints and the category theoretic orientation of the classification-projection semantics in the central core, and the bottom level corresponds to the conceptual graphs knowledge model with assertions (closed expressions) in exact correspondence with conceptual graphs.

CKML provides a conceptual knowledge framework for the representation of distributed information. Earlier versions of CKML followed the philosophy of Conceptual Knowledge Processing [49, 50], a principled approach to knowledge representation and data analysis that "advocates methods and instruments of conceptual knowledge processing which support people in their rational thinking, judgment and acting and promote critical discussion". The new version of CKML continues to follow this approach, but also incorporates various principles, insights and techniques from Information Flow, the logical design of distributed systems [51]. This allows diverse communities of discourse to compare their own information structures, as coded in ontologies, logical theories and theory interpretations, with that of other communities that share a common terminology and semantics.

Beyond the elements of OML, CKML also includes the basic elements of information flow: classifications, infomorphisms, theories, interpretations, and local logics.

6.2.5 OIL

The *Ontology Inference Layer* (OIL) is another ontology representation language compatible with Web standards. It derives from *Frame* representation languages (Section 4.1) and *Description Logics* (Section 3.4); indeed, its objective is to build on top of RDF Schema by adding modelling constructs for Description Logics.

OIL is based on a multi-layer architecture. Each layer is an extension of the level below, obtained by adding expressiveness to it. The aim of this architecture is to allow intelligent agents not able to perform complex inferences to access only low layers of a description, in order to be able to partially understand it. Figure 15 shows the relation between OIL and RDF.

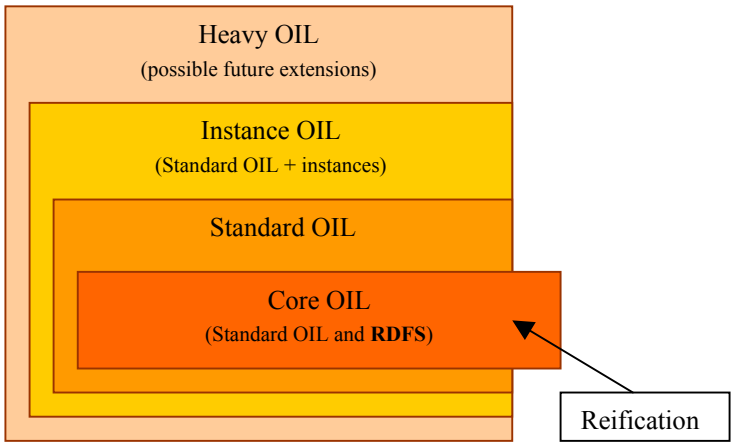


Figure 15. OIL layers and their relationships with RDF Schemas.

As shown in the figure, *Core OIL* largely overlaps with RDFS (exception for reification features). This means that simple RDF-based agents can also process OIL ontologies, although they will not be able to completely understand them.

Standard Oil includes a set of modelling primitives which ensure an expressive power enough to perform inferences. This layer allows individual variables to be expressed in term definitions.

Instance OIL includes a complete instance integration and database functionalities.

Heavy OIL, finally, is a layer providing possible future extension.

An OIL ontology is a structure whose elements can themselves be sub-structures. Moreover, these elements can be *mandatory*, *optional* (indicated by the symbol "?") or *multiple* (indicated by the symbol "+" or "*", depending on whether the empty value is admissible).

In particular, an OIL ontology is composed of the following three sections.

- **import?** A set of links to other OIL modules to be included in this ontology. It is possible to include different ontologies defining, for each of them, a different prefix (like in RDF and in SHOE).
- **rule-base?** A set of rules to perform logic inferences (also called *axioms*). Currently, there are no clear structure or semantics for these rules.
- **class and slot definitions.** Zero or more class definitions (*class-def*) and slot definitions (*slot-def*), whose structure is described below.

A **class definition** (class-def) associates a name with a description of a class. It is composed of the following elements:

- **type?** The type of the class: primitive or defined.
- **name.** The name of the class (a string).
- **documentation.** A description of the class (a string).
- **subclass-of?** A set of one of more "class expressions". A **class expression** is the name of a class or a Boolean combination of class names.

A **slot definition** (slot-def) associates the name of a slot with a slot description, where "slot" is, as in Frame language, a binary relation. A slot definition specifies some constraints about the relation. It is composed of the below elements:

- **name.** The name of the slot (a string).
- **documentation?** A description of the slot (a string).
- **subslot-of?** A set of one or more slots. The relation defined in this slot-def must be a sub-relation of each of the slots of the set.
- **domain?** A set of one of more class expressions. If the pair (x,y) is an instance of the slot, x must be an instance of each class expression of the set.
- **range?** A set of one of more class expressions. If the pair (x,y) is an instance of the slot, y must be an instance of each class expression of the set.
- **inverse?** The name of the slot S which is the inverse of the slot defined. If the pair (x,y) is an instance of the slot S, (y,x) must be an instance of the slot defined.
- **properties?** A set of one or more properties of the slot. Some valid properties are: transitive and symmetric properties.

6.2.6 DAML+OIL

DAML+OIL is a semantic markup language for web resources developed by DARPA (the DAML acronym means DARPA Agent Markup Language). DAML is based on the most recent standards such as RDF and RDF Schema, extending them by using primitive modelling derived from frame-based languages. DAML+OIL (March 2001) extends DAML+OIL (December 2000) by using datatypes deriving from XML Schema.

In particular, DAML+OIL has moved away from the original frame-like ideals of OIL and is, in a much stronger sense than OIL, an alternative syntax for a Description Logic. Assertions in DAML+OIL ontology (such as the superclasses or slot constraints applying to a class) are couched in terms of general axioms. The idea of a "frame" (a single place in which facts about a class are gathered) is lost, or is at least not inherent in the language.

A DAML+OIL ontology has a heading containing some information related to the ontology itself:

```
<Ontology rdf:about="">
<versionInfo>
$Id: reference           0 2001/04/11 16:27:53 mdean Exp $
</versionInfo>
<rdfs:comment>A
<imports rdf:resource="http://www.daml.org/2001/03/daml+oil"/>
</Ontology>
```

Class and property definitions follow the heading.

Class definitions: A class definition is introduced by the daml:Class element and contains the following components:

- **rdfs:subClassOf*** defines the C class as subclass of the defined class-expression, unlike RDF(S) specifications DAML+OIL does not forbid the cyclicity that can derive from this relation.
- **daml:disjointWith*** establishes that the C class and the defined class-expression have no instances in common.
- **daml:disjointUnionOf*** all the classes defined by the listed class-expressions must be disjoined and their union must be equal to C.
- **daml:sameClassAs*** C and the defined class-expressions have the same instances.
- **boolean combinations of class-expressions** The C class must be equal to the class deriving from the combination of two or more class-expressions.
- **Elements list?** The C class contains exactly the listed instances.

Class Expressions: A class-expression can indicate:

- a class name.
- an instances list.
- a property restriction.
- a boolean combination of class-expressions.

Instances list

The **daml:oneOf** element allows the definition of a class listing the instances that belong to it:

```
<daml:oneOf parseType="daml:collection">
<daml:Thing rdf:about="#Eurasia"/>
<daml:Thing rdf:about="#Africa"/>
<daml:Thing rdf:about="#North America"/>
<daml:Thing rdf:about="#South America "/>
<daml:Thing rdf:about="#Australia"/>
<daml:Thing rdf:about="#Antarctica"/>
</oneOf>
```

Constraints on properties

A property restriction defines an anonymous class (the class of instances that satisfy the constraints). The constraints can be concerned with properties that relate two classes or properties relating a class to a XML-Schema datatype. A constraint is introduced by the **daml:Restriction** element and followed by the **daml:onProperty** element that indicates the property to which the constraint is applied, and also by one or more of the following elements:

- **daml:toClass** <class-expression> It defines the x object class for which if (x,y) is a property instance, then y is a class-expression instance indicated by daml:toClass.
- **daml:hasValue** <instance or value of a datatype> If y is the instance, it defines the x object class for which (x,y) is a property instance.
- **daml:hasClass** <class-expression> It defines the x object class for which a y instance of the

class-expression exists such as (x,y) is a property instance. This does not exclude the existence of an (x,y') instance of y' property not belonging to the class-expression.

- **daml:cardinality** <Whole not Negative> **N**. It defines the object class that has exactly N different values for the property to which the restriction is applied.
- **daml:maxCardinality** <Whole not Negative> **N**. It defines the object class that has maximum N different values for the property to which the restriction is applied.
- **daml:minCardinality** <Whole not Negative> **N**. It defines the object class that has at least N different values for the property to which the restriction is applied.
- **daml:cardinalityQ** <Whole not Negative> **N**. It defines the object class that has exactly N different values for the property to which the restriction is applied and these values are instances of the class or the datatype indicated in the daml:hasclassQ element.
- **daml:maxCardinalityQ** <Whole not Negative> **N**. It defines the object class that has maximum N different values for the property to which the restriction is applied and these values are instances of the class or the datatype indicated in the daml:hasclassQ element.
- **daml:minCardinalityQ** <Whole not Negative> **N**. It defines the object class that has at least N different values for the property to which the restriction is applied and these values are instances of the class or the datatype indicated in the daml:hasclassQ element.

Class combinations

A class combination can be realized through one of the following elements:

- **daml:intersectionOf** <class – expression list> It defines the object class in common with the class-expressions of the list (logic conjunction).
- **daml:unionOf** <class – expression list> It defines the class whose members represent the union of the objects belonging to the class-expressions of the list (logic disjunction). º
- **daml:complementOf** <class – expression list> It defines the object class that does not belong to the class-expression (logic negation limited only to the objects).

Property Definition

The properties define binary relations between two entities; if the latter are two classes, then one talks about ObjectProperty; otherwise, if there is a relation between a class and a XML-Schema datatype, one talks about DatatypeProperty.

A P property contains the following elements:

- **rdfs:subPropertyOf** <property name> If (x,y) is an instance of P, then it is also an instance of the property indicated by subPropertyOf.
- **rdfs:domain** <class - expression> If (x,y) is an instance of P, then x is an instance of the class-expression. The presence of more than one rdfs:domain defines the domain of P as the intersection of each class-expression.
- **rdfs:range** <class - expression> If (x,y) is an instance of P, then y is an instance of the class-expression. The presence of more than one rdfs:range defines the range of P as the intersection of each class-expression.
- **daml:samePropertyAs** <property name> P is equivalent to the indicated property. daml:inverseOf <property name> If (x,y) is an instance of P, then (y,x) is an instance of the indicated property.

Properties can be introduced using the following elements too:

- **daml:TransitiveProperty** It establishes that the P property is transitive; if $(x,y) \in P$ and $(y,z) \in P$, then $(x,z) \in P$.
- **daml:UniqueProperty** If $(x,y_1) \in P$ and $(x,y_2) \in P$, then $y_1 = y_2$

- **daml:UnambiguosProperty** If $(x_1,y) \in P$ and $(x_2,y) \in P$, then $x_1 = x_2$, it should be noticed that the daml:UniqueProperty and the daml:UnambiguosPropert define global constraints for a property which are independent of the class to which the property is applied.

Instances

Instances (objects) are written in a syntax used for RDF(S); what follows is an example:

```
<continent rdf:ID="Oceania"/>
<rdf:Description rdf:ID="Oceania">
<rdf:type>
<rdfs:Class rdf:about="#continent"/>
</rdf:type>
</rdf:Description>
```

Datatype

To allow a correct interpretation of data with an XMLSchema datatype, one resorts to a particular method for writing values that indicates value and type of data.

For example, to write the value 10.5, one should write:

```
<xsd:decimal rdf:value=''10.5''>.
```

6.2.7 OWL

The OWL Web Ontology Language [64] is being designed by the W3C Web Ontology Working Group as a revision of DAML+OIL. It takes DAML+OIL as its basis and its expressiveness will be close to that of DAML+OIL. DAML+OIL is extended in two ways:

- A lower step-in threshold (which is perceived to be too high for DAML+OIL) is provided.
- A human-readable presentation syntax is provided, besides an RDF/XML-based syntax.

The W3C working draft specifies eight design goals for the Web ontology language:

- Shared ontologies. Ontologies should be publicly available and different data sources should be able to commit to the same ontology for shared meaning. Also, ontologies should be able to extend other ontologies in order to provide additional definitions.
- Ontology evolution. An ontology may change during its lifetime. A data source should specify the version of an ontology to which it commits. An important issue is whether or not documents that commit to one version of an ontology are compatible with those that commit to another. Both compatible and incompatible revisions should be allowed, but it should be possible to distinguish between the two. Note that since formal descriptions only provide approximations for the meanings of most terms, it is possible for a revision to change the intended meaning of a term without changing its formal description. Thus determining semantic backwards-compatibility requires more than a simple comparison of term descriptions. As such, the ontology author needs to be able to indicate such changes explicitly.
- Ontology interoperability. Different ontologies may model the same concepts in different ways. The language should provide primitives for relating different representations, thus allowing data to be converted to different ontologies and enabling a "web of ontologies."
- Inconsistency detection. Different ontologies or data sources may be contradictory. It should be possible to detect these inconsistencies.
- Balance of expressivity and scalability. The language should be able to express a wide variety of knowledge, but should also provide for efficient means to reason with it. Since these two requirements are typically at odds, the goal of the web ontology language is to find a balance that supports the ability to express the most important kinds of knowledge.
- Ease of use. The language should provide a low learning barrier and have clear concepts and meaning. The concepts should be independent from syntax.

- Compatibility with other standards. The language should be compatible with other other commonly used Web and industry standards. In particular, this includes XML and related standards (such as XML Schema and RDF), and possibly other modeling standards such as UML.
- Internationalisation. The language should support the development of multilingual ontologies, and potentially provide different views of ontologies that are appropriate for different cultures.

These design goals motivate a number of requirements for a Web Ontology Language. The Working Group currently feels that the requirements described below are essential to the language:

- Ontologies must be objects that have their own unique identifiers, such as a URI reference.
- Two terms in different ontologies must have distinct absolute identifiers (although they may have identical relative identifiers). It must be possible to uniquely identify a term in an ontology using a URI reference.
- Ontologies must be able to explicitly extend other ontologies in order to reuse terms while adding new classes and properties.
- Resources must be able to explicitly commit to specific ontologies, indicating precisely which set of definitions and assumptions are made.
- It must be possible to provide meta-data for each ontology, such as author, publish-date, etc. The language should provide a standard set of common metadata properties. These properties may or may not be borrowed from the Dublin Core element set.
- The language must provide features for comparing and relating different versions of the same ontology. This should include features for relating revisions to prior versions, explicit statements of backwards-compatibility, and the ability to deprecate terms.
- The language must be able to express complex definitions of classes. This includes, but is not limited to, sub classing and Boolean combinations of class expressions (i.e., intersection, union, and complement).
- The language must be able to express the definitions of properties. This includes, but is not limited to, sub properties, domain and range constraints, transitivity, and inverse properties.
- The language must provide a set of standard data types. These data types may be based on XML Schema data types.
- The language must include features for stating that two classes or properties are equivalent.
- The language must include features for stating that pairs of identifiers represent the same individual.
- In general, the language will not make a *unique names assumption*. That is, distinct identifiers are not assumed to refer to different objects (see the previous requirement). However, there are many applications where the unique names assumption would be useful. Users should have the option of specifying that all of the names in a particular namespace or document refer to distinct objects.
- The language must provide a way to allow statements to be "tagged" with additional information such as source, timestamp, confidence level, etc.
- The language must support the ability to treat classes as instances. This is because the same concept can often be seen as a class or an individual, depending on the perspective of the user.
- The language must support the definition and use of complex/ structured data types. These may be used to specify dates, coordinate pairs, addresses, etc.
- The language must the support the specification of cardinality restrictions on properties. These restrictions set minimum and maximum numbers of object that any single object can be related to via the specified property.
- The language should have an XML serialisation syntax.
- The language should support the specification of multiple alternative user-displayable labels for the objects within an ontology. This can be used, for example, to view the ontology in different natural languages.

- The language should support the use of multilingual character sets.
- In some character encodings, e.g. Unicode based encodings, there are some cases where two different character sequences look the same and are expected, by most users, to compare equal. Given that the W3C I18N WG has decided that early uniform normalization (to Unicode Normal Form C) as the usual approach to solving this problem, any other solution needs to be justified.

An OWL ontology is a sequence of axioms and facts, plus inclusion references to other ontologies which are considered to be included in the ontology. OWL ontologies are web documents, and can be referenced by means of a URI.

Axioms are used to associate class and property IDs with either partial or complete specifications of their characteristics, and to give other logical information about classes and properties.

Class axioms include the common, widely understood, frame idiom. The abstract syntax used here is meant to look somewhat like the syntax used in some frame systems. Each frame-like class axiom contains a collection of more-general classes; a collection of local property restrictions, in the form of restriction constructs; and a collection of descriptions. The restriction construct gives the local range of a property, how many values are permitted, and a collection of required values. Descriptions are used to specify boolean combinations of restrictions and other descriptions as well as construct sets of individuals. Classes can also be specified by enumeration or be made the same or disjoint.

Properties can be the equivalent to or subproperties of others; can be made functional, inverse functional, or transitive; and can be given global domains and ranges. However, most information about properties is more naturally expressed in restrictions, which allow local cardinality and range information to be specified.

Facts state information about particular individuals in the form of a class that the individual belongs to plus properties and values. Individuals can either be given an individualID or be anonymous (blank nodes in RDF terms).

6.2.8 LIA knowledge model

LIA (Learning Intelligent Advisor) is part of a bigger framework (funded by the EC under the 5th Framework Program – Information Society Technologies) named m-learning [66] whose objective consists in the development, the experimentation and the evaluation of product and service prototypes for the dissemination of didactical micromodules through mobile technologies to promote lifelong learning. LIA is the "intelligent" engine of the m-learning Course Management System able to offer automatic user assessment and course customization features. Such "intelligent" functionalities are realized by means of Artificial Intelligence techniques often applied in the Intelligent Tutoring Systems. LIA is composed of three main modules – didactic knowledge representation, the student model, and some planning procedures which are able automatically to create a course satisfying all the student learning requirements, taking into account both his/her present knowledge and learning preferences. The LIA knowledge representation model, which meets many of Diogene's needs and is being considered by the project as a suitable model to adopt, is outlined below.

LIA didactic knowledge is represented through different abstraction levels. The lowest level is composed of *Learning Objects*. A Learning Object (LO) is defined as any entity which can be used, re-used or referenced during technology-supported learning. In our case, a Learning Object is a logical container that represents an atomic Web-deliverable resource such as a Lesson (an HTML page), a Simulation (a Java applet), a Test (an HTML page with an evaluating form) and each kind of Web-deliverable object.

Learning Objects must be indexed in order to let LIA know what each one of them is about and how they can be used during the learning process. This is done by means of a second abstraction representation level (*Metadata*). Metadata is a collection of attributes about a Learning Object (LO) describing some features such as its type (text, simulation, slide, questionnaire,…), the required educational level (high school, university,…), the language, the interactivity level and so on. The IMS Global Learning Consortium's standard was adopted to specify the syntax and semantics of the attribute declarations.

Finally, a third abstraction level (*Ontology*) is used to represent *Domain Concepts* and their relations. A Domain Concept (DC) is a concept belonging to the described didactic domain and can be possibly explained by one or more Los. For example, in the *Maths* domain, we can have concepts such as "Mathematical Analysis" or the (sub-) concepts "Limits", "Derivatives", and so on, which, in turn, can be explained by the Los: "Introduction to Limites" (an HTML text), "Lesson 1: Limits" (slides), "Lesson 2: Derivatives" (slides), and so on. Furthermore, a DC can be a special LIA event (a Milestone) used to receive feedback from the student. A Milestone is associated with one or more Los (of type test).

Typical relations among concepts are: *IsPart_Of(Limits, Mathematical Analysis)* or *Requires(Derivatives, Limits)*, to indicate, respectively, a hierarchical relationship and a constraint on the learning order of the two concepts. DCs and their relations are described in LIA using SHOE.

The following shows a more formal description of the LIA admissible relation types linking DCs:

- *IsPart_Of(x,y)* means that *x* is a component of *y*.
- *Requires(x,y)* means that *x* needs *y* as a prerequisite. This relation poses a constraint on the delivery order of the DCs to the student.
- *Suggested_Order(x,y)* means that it is *preferable* to learn *x* and *y* in this order. Note that this relation also poses a constraint on the DCs' order but now it is not necessary to learn *y* if we are interested only in *x*.
- *IsTested_by(x,ML)* means that *ML* (a Milestone) concerns the verification of the concept *x*.

All the proposed relations have a binary ariety, this they can be easily represented by arcs in a graph data structure (where each note represents a DC). Besides the above relationships among DCs, we need a link between the Ontology and Metadata levels:

- *Explained_by(d,l)* means that the DC *d* can be explained by means of the LO which is described by the metadata l. Note that *d* can be a Milestone and *l* a test.

Finally, each metadata directly refers to the LO it describes. Figure 16 shows an example of knowledge representation. We used the following abbreviations: L=Limits, D=Derivatives, I=Integrals, S=Series, B=IsPart_of, R=Requires, SO=SuggestedOrder, E=Explained_by, LO_i=the metadata describing the i-th Learning Object. Note a LO can explain one or more DCs and each DC can be explained by more than one LO. Indeed the semantics of *Explained_by(x,y)* poses that y is enough to completely explain *x*. If *Explained_by(x,y)* and *Explained_by(x,z) (z≠y)* then we intend that *z* and *y* are mutually exclusive. Moreover, if *Explained_by(x,y)* and *Explained_by(z,y)* then *y* contains enough contents to explain both DCs *x* and *z*. Finally, the figure does not explicitly show the lowest knowledge representation level because each LO metadata directly refers to its indexed resource in a unique and trivial way.

To simplify, the term LO is used to refer to the structure given by the union of a learning resource and its metadata representation.

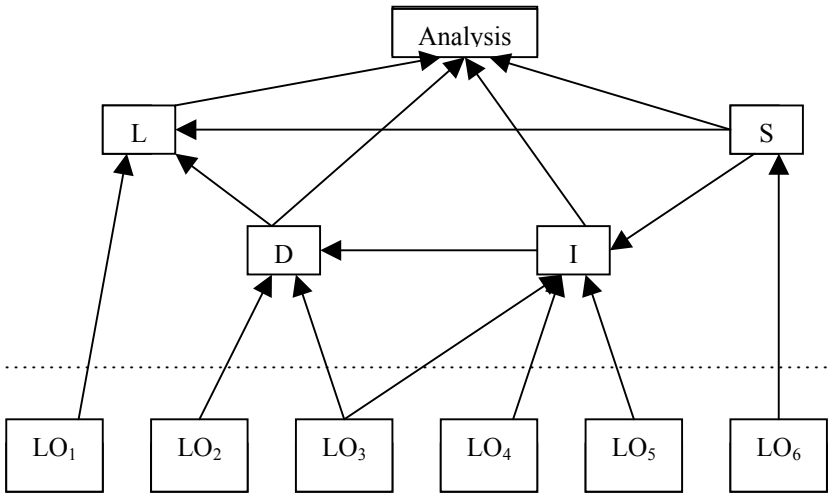


Figure 16. An example of LIA knowledge representation belonging to the didactic domain "Maths".

6.3 Ontology Editors

The following ontology editors have contributed to the development of the Semantic Web by enabling activity in the ontology community.

6.3.1 Protégé 2000

Protégé 2000 [55], developed by the Knowledge Modelling Group at Stanford University, provides an integrated knowledge-base editing environment and an extensible architecture for the creation of customised knowledge-based tools. It has been developed using a plug-in architecture, where new services can be added easily to the environment. It conforms to the Open Knowledge Base Connectivity (OKBC) protocol for accessing knowledge bases stored in knowledge representation systems.

Protégé 2000 is a tool a platform and a library. The tool allows the user to construct a domain ontology, customise knowledge acquisition forms and enter domain knowledge. The platform can be extended with graphical widgets to access applications embedded in other systems, and other applications can use the library to access and display knowledge bases.

The tool accesses classes, instances, slots and applications through a uniform GUI which enables convenient co-editing between these elements. It is designed to guide developers and domain experts through the process of system development, allowing the reuse of domain ontologies and problem-solving methods, thereby shortening the development time required. This is an iterative process, with cycles of revision to various components of the system. The development process is described in Figure 17 below (dotted arrows show places where revisions are usually necessary) [56].

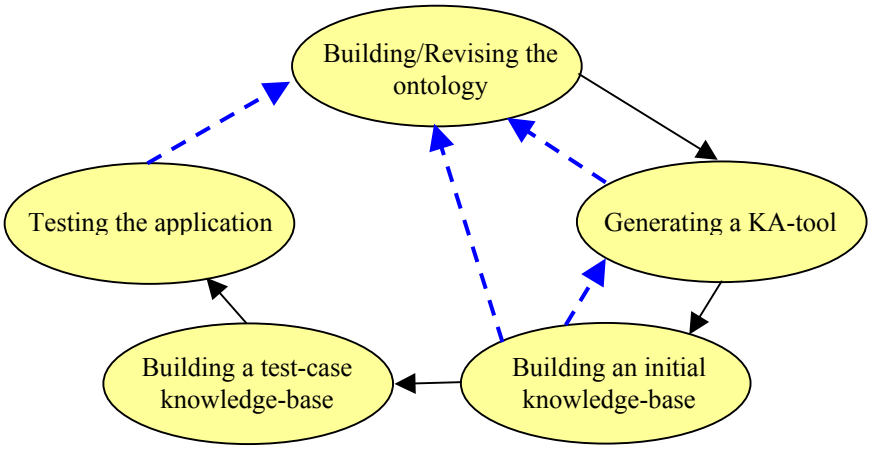


Figure 17. The typical pattern of use for the Protégé-2000 subsystems

6.3.2 OntoEdit

OntoEdit [57], developed by the Knowledge Management Group at the University of Karlsruhe, is an ontology editor that enables the design, adaptation and import of knowledge models for application systems, integrating numerous aspects of ontology engineering. It combines methodology-based ontology development with capabilities for collaboration and interfacing.

The three main components of OntoEdit (a repository of ontologies, an inference and query engine, and various translators) are integrated into OntoServer, the client-server architecture in which OntoEdit is embedded. OntoServer delivers built-in general-purpose deductive reasoning facilities for ontology engineering and applications clients.

Ontologies are specified using OXML, a flexible format for guiding the whole engineering process of ontology development. This process occurs in three stages: requirements specification, refinement, and evaluation. Firstly, the envisaged ontology is defined in terms of a description of the domain, the goal of the ontology, available knowledge sources, potential users and applications supported. This description is then refined and formalised and, finally, is evaluated according to the previously established requirement specifications.

6.3.3 OilEd

OilEd [58] was developed in the context of the OntoKnowledge EU project for the easy development of OIL languages. It is not intended as a complete ontology editor (able to support the development, migration and integration of ontologies), but rather "the notepad of ontology editors", enabling users to build ontologies and check them for consistency.

The purpose of OilEd is to provide a simple, freeware editor that demonstrates the use of OIL and, now, DAML+OIL. Its design has been heavily influenced by similar tools such as Protégé and OntoEdit, but has focused on demonstrating how a frame-based paradigm can deal with a more expressive modelling language, and the possibilities and benefits of using a FaCT reasoner to classify ontologies.

FaCT is a DL classifier featuring expressive logic, optimised tableaux implementation (now the standard for DL systems) and its CORBA based client-server architecture. Using its CORBA interface, OilEd can classify and organise concept hierarchies and spot inconsistencies, and enables the modeller to construct the model through the use of descriptions rather than by explicitly building hierarchies.

Basic functionality allows the definition and description of classes, slots, individuals and axioms within an ontology.

6.3.4 WebODE

WebODE [59], developed at the same time as Protégé 2000 and OntoEdit, is an ontological engineering environment that enables ontology editing and supports most of the activities of the ontology life cycle (reengineering, conceptualisation, implementation, etc.). Using its automatic exportation and importation services, it is possible to integrate WebODE ontologies with other systems, including XML, RDFS, OIL and DAML+OIL.

WebODE's ontology editor comprises form-based and graphical user interfaces, a user-defined-views manager, an inference engine (developed in Ciao Prolog), an axiom builder, a documentation service, and a consistency checker (which checks the type constraints, numerical value constraints, cardinality constraints and taxonomic consistency verification.

The WebODE platform has been built using a three-tier model: a data tier, a business logic tier and a presentation tier. An application server basis provides high extensibility and usability.

The knowledge model is mainly based on the set of intermediate representations of METHONTOLOGY. It allows the representation of concepts, groups of concepts, taxonomies (single and multiple inheritance), ad-hoc binary relations, constants, axioms and instances of concepts and relations. It integrates all its services in a well-defined architecture, stores its ontologies in a relational database and provides additional services such as the inference engine, the axiom builder, ontology acquisition or catalogue generation.

6.3.5 Ontolingua

Ontolingua is a tool for collaborative ontology construction, enabling developers to browse, create, edit, modify and use ontologies [60]. The Ontolingua development environment provides a suite of ontology authoring tools and a library of modular, reusable ontologies, designed to support the process of achieving consensus on common shared ontologies by geographically distributed groups.

Ontolingua's inclusion model for ontologies enables users to assemble new ontologies quickly from existing ones in the library. This model separates its simple formal semantics and the input/output properties of the system using it. The former deals with simple inclusion, circular inclusion dependencies, restrictions and polymorphic refinement. The latter is transparent to users and yields succinct external readable representations.

The web-based ontology editing environment, integrated with the Ontolingua server and incorporating the inclusion model, provides individual uses with a rich editing environment. It also supports collaboration between distributed groups of users, and provides access to the growing library of ontologies. The server also provides an important publishing medium for ontologies, as hypertext can reliably point to any document in the publicly accessible library.

6.3.6 Ontosaurus

Ontosaurus [61], developed with funding from DARPA, is a Web-based ontology and knowledge base development environment using the Loom KR system. Loom enables Ontosaurus to provide inference mechanisms and to manipulate various ontologies. It is able to identify subsumption relations implicit in user-specified ontologies, detect inconsistencies in definitions, and validate constraints on hierarchically organised roles and relations.

The system is being extended to support:

- A collaborative development environment, enabling large ontologies and knowledge bases to be developed and maintained by groups of experts.
- A view manager, which tailors information about a concept according to the needs of a user or domain. This enables a single ontology to be viewed from multiple perspectives.
- Tools for constructing new ontologies tailored to a particular domain by composing, pruning, extracting and merging components from a repository of ontologies and reusable domain knowledge bases.
- Semantic analysis and maintenance tools to inspect inferences and their logical dependencies,

including: a) validating the completeness of an ontology; b) identifying inconsistencies and conflicts within ontologies; c) identifying undefined concepts and relations; and d) highlighting semantic differences between successive versions of an ontology.

- Translation toolkit, enabling the ontologies developed in Ontosaurus to be made available in different languages and KR formalisms.

6.4 Knowledge Systems Interoperability

The concept of information integration, the resolution of semantic conflicts between concepts, is based on the desire of the information community to:

- Improve the quality of data by making it available as large and complete.
- Reduce the costs resulting from the multiple use of existing information sources.
- Avoid redundant data and the conflicts that can arise from redundancy.

However, the management of knowledge and its components faces obstacles in the form of organisational and technical problems. Firstly, a suitable information source must be located which contains the data needed for a given task. Once the information source has been found, access to the data contained therein has to be provided. Furthermore, access has to be provided on both a technical and an informational level. In short, information integration not only needs to provide full accessibility to the data, it also requires that the accessed data may be interpreted by the remote system. Technologies which enable systems to interoperate are fundamental to overcoming these barriers.

6.4.1 BUSTER

BUSTER (Bremner University Semantic Translation for Enhanced Retrieval) [62] attacks the problem of information integration on three levels: syntactic integration, structural integration, and semantic integration.

- Syntactic integration. The task of syntactic data integration is to specify the information source on a syntactic level. The standard technology used to overcome problems on this level are wrappers, which hide the internal data structure model of a source and transform the contents to a uniform data structure model.
- Structural integration. The task of structural data integration is to re-format the data structures to a new homogeneous data structure. This can be done with the help of a formalism that is able to construct one specific information source out of numerous other information sources. This is the classical task of middleware which can be done with CORBA on a low level or rule-based mediators on a higher level.
- Semantic integration. Semantic heterogeneities are the main problems that have to be solved within spatial data integration. In order to discover semantic heterogeneities, a format representation is needed. WWW standardised markup languages such as XML and RDF have been developed for this purpose. Ontologies are also useful for the integration/interoperation process.

BUSTER uses wrappers for the syntactic level, mediators for the structural level, and both context transformation rule engines and classifiers (mappers) for the semantic level. CORBA is used for the communication of the components.

During an acquisition phase all desired information for providing a network of integrated information sources is acquired. This includes the acquisistion of a Comprehensive Source Description (CSD) of each source together with the Integration Knowledge (IK) which describes how the information can be transformed from one source to another.

In the query phase, a user or an application formulates a query to an integrated view of sources. Several specialised components in the query phase use the acquired information, i.e. the CSDs and IKs, to select the desired data from several information sources and to transform it to the structure and the context of the query.

All software components in both phases are associated to three levels: the syntactic, the structural and

the semantic level. The components on each level deal with the corresponding heterogeneity problems. The components in the query phase are responsible for solving the corresponding heterogeneity problems whereas the components in the acquisition phase use the CSDs of the sources to provide the specific knowledge for the corresponding component in the query phase. A mediator, for example, which is associated with the structural level, is responsible for the reconciliation of the structural heterogeneity problems. The mediator is configured by a set of rules that describe the structural transformation of data from one source to another. The rules are acquired in the acquisition phase with the help of the rule generator.

An important characteristic of the BUSTER architecture is the semantic level, where two different types of tools exist for solving the semantic heterogeneity problems. This demonstrates the focus of the BUSTER system, providing a solution for this type of problem. Furthermore, the need for two types of tools exhibits that the reconciliation of semantic problems is very different and must be supported by a hybrid architecture where different components are combined.

6.4.2 SKC

The Scalable Knowledge Composition (SKC) [63] project offers another approach to resolving heterogeneity in information systems. The SKC approach will develop an algebra over ontologies that represents the terminologies from distinct, typically autonomous domains.

An intersection operation permits focusing on critical linkages. Logical partitioning of knowledge into chunks reduces computational complexity by exponential factors, while enabling distribution of computations to be processed in parallel over many processors.

The Ontology Algebra will itself be knowledge-driven, a necessary feature to deal with the complexities and inconsistencies that arise when distinct knowledge resources are merged. By formulating the needed operation as an algebra, SKC provides a sound basis for extensive and incremental knowledge manipulation. The knowledge that will drive the Ontology Algebra is limited to rules that enable articulation, the linking disjoint knowledge resources, and interoperation, the processing of information based on the articulated knowledge.

A disciplined manipulation of knowledge resources will be essential in achieving the following objectives, needed for effective use by real-world customers and their applications:

1. Correctness, i.e., consistency within a domain, needed to engender trust by the owners of the applications
2. Depth, i.e., linkage to atomic instances for realism in modelling and processing, so that information for decisions is grounded on a factual basis
3. Maintainability, i.e., clear identification and enabling of responsible maintainers of the domain knowledge
4. Effective use and reuse, i.e., the ability to create subsets, intersections, and compositions that do not overwhelm the application
5. Scalability of the knowledge resources. i.e., the ability by applications to compose chunks and their intersections without limit.

The ontology algebra to be demonstrated in SKC allows knowledge bases to be developed within manageable, limited-size domains. Within a modest domain the high-cost operations to validate local consistency, adequacy of depth, and up-to-date-ness can be bounded. By being able to assign development and maintenance responsibility of manageable chunks to responsible domain experts, the first three objectives listed will be achieved. The remaining two objectives, effective use and reuse, and scalability are enabled by having tools, based on the Ontology Algebra, to build the articulation knowledge needed for the joint use of the basic chunks.

The SKC approach is innovative in two further areas:

1. The distinct knowledge sources do not have to collaborate directly; their collaboration is managed by specialists in joining distinct knowledge bases. We will illustrate below that such functions are already being performed in current enterprises.

2. The base knowledge resources remain available to the applications for in-depth processing, using the linkages established during the process of determining the articulations. Their use is then equivalent to the delegation of detailed domain tasks to specialists.

In our Scalable Knowledge Composition (SKC) proposal we make two assumptions, both of them quite conservative (and hence lowering our research risk).

1. Within the context of a specific domain base ontology resource, all terms are consistent.
2. No term from one domain ontology matches a term from another ontology, unless a matching rule has been provided.

We will illustrate some of the matching rules we expect to be able to handle and then describe how they will be used in our ontology algebra. Many matching rules will have to state the obvious, since we assume that words from a different context domain do not refer to the same class, unless a matching rule exists:

HOUSE (carpentry) = HOUSE (householder)
TABLE (carpentry) = TABLE (householder)

Such rules are needed if the application deals with houses and their furniture in both the owner's and in a maintenance context. That application will probably not need a rule:

SALARY (carpentry) = SALARY (householder)

The house maintenance application may need the rule:

SINKER(carpentry) in NAIL (householder)

denoting that the householder will refer to large nails (SINKERs) as NAILs. However, the BRAD (CARPENTRY), not used by the householder, may not need to be so defined. The matching rules themselves may be bounded within an articulation context.

We will use another example, that of purchasing goods for a department store, where the goods are in the context of wholesale purchasing. Here the domain experts, defining the articulation knowledge, are the purchasing agents, who all should be members of the American Society of Purchasing Agents (ASPA). For SHOE purchases there will be the matching rule:

SHOE (store) = SHOE (factory)

The terms SHOE are not terminals in the store and factory ontologies, but rather the roots of subhierarchies, as PUMPS, WEDGIES, LOAFERS, SANDALS, etc.

{PUMPS, WEDGIES, LOAFERS, SANDALS}(store) ISA SHOE(factory)

Not all terms in the subhierarchies will match, so that the ASPA will have further rules, such as,

PUMPS (store) = SHOE(factory) if HEEL(factory) > 5cm.

Parts of shoes, needed to specify purchases, as HEEL, will also appear in the algebra's rule base; but by

the time the NAIL(factory) is reached, no matching rule is needed. Confusion with NAIL(anatomy), which may be an entry in the shoe store's ontology, is hence avoided. We have already an ontology for anatomy, if needed in the store for articulation with health problems due to high heels, provided by the College of Pathologists [ACP: 92].

The fact that a term does not appear in the articulation intersection does not mean that it is not accessible for domain-specific computation, using methods which are available to the application. For instance, NAIL(factory) may be a term used in a computational method DURABILITY whose execution remains local to the factory system, but which can be invoked from the articulated result, if there is a matching rule, say another concept entry that needs to be matched among factory and store is the SIZE of a SHOE. Here a table matching alternate size standards as well as a conditional to select a standard may be needed.

```
if (LOCATION (factory) =`EUROPE')
                then SIZE (factory) = SIZETABLE (SIZE (store));
                else SIZE (factory) = SIZE (store);
```

Color specifications will certainly need a table, since the factory is likely to use a code

```
COLORCODE (factory)= colortable (COLOR (store)),
```

allowing the store to refer to COLORCODE `XY14WZ' as `Spring Pink'. These examples should illustrate both the need for an Ontology Algebra and the approach needed to achieve its implementation. The number of rules needed to purchase shoes will be modest. Otherwise purchasing agents today would already have an impossible task.

Maintenance of the rules, to deal, say, with changes in shoe fashions is now assigned to ASPA rather than to the factory in the store. Stores could, of course, add their own mappings, a local issue, but one that can still be aided by having the ontology algebra. In general, subsidiary entries of shared entries, as SIZE and COLOR for SHOEs are candidates for sharing as well. The observation provides the basis for tools that aid in the creation and a maintenance of the articulation algebra.

7. Conclusions

In this survey we have given an overview of the most important and diffused AI methodologies for representing knowledge, both general-purpose and specialized for didactic domains. In Chapter 5 we saw the main Intelligent Tutoring System architectures, and how the knowledge they need is usually partitioned in a student model module (describing, for each student, a specific profile) plus a domain knowledge module (which codes the ITS knowledge about its specific learning field). Both modules, and especially the latter, use various AI representation techniques to describe information. All such techniques are, first of all, characterised by an *explicit* part and an *implicit* ("calculated" or "derived") part. The former is composed of a set of facts directly coded by the knowledge base constructor, while the latter is composed of a set of facts *potentially* known by the intelligent agent, i.e., all those facts derivable by the system inference mechanism starting from its explicit knowledge.

Following this guideline, we examined various approaches towards coding and deducing information, from the most general-purpose to the most specific, used by e-learning community.

After a general introduction in Chapters 1 and 2, we examined, in Chapter 3, logic knowledge representation methods. Recently, the Logic paradigm has been the most used in the Knowledge Representation community, for its foundational aspects. Indeed, non-logic approaches can always be translated in a logic formalism, in order to better characterize their semantics together with their expressive power and computational complexity. Studying logic formalisms we saw how the expressive scope of a formal knowledge representation technique always must find a trade-off with its computational cost, and the most expressive formalisms, such as First Order Logic, are partially decidable.

In Chapter 4 we saw other general-purpose representation methodologies, not (directly) based on a logic language but rather on network structures. In this case, knowledge is coded by means of facts and links among facts rather then through strings (logic formulas). Implicit knowledge derivation in network structures is more efficient because following links is a quicker operation than matching sentences. However, network approaches often do not completely reach FOL expressiveness, being based on a "Description Logic-like" metaphor. We also analyzed the possibility of utilizing the Unified Modelling Language as a general knowledge representation paradigm. Indeed, UML has the advantage of being a very diffused language in database and object-oriented programming communities. Furthermore, UML (like all the object-oriented programming languages) derives from Minsky's original work on Frames, the famous paper on knowledge representation approaches based on Semantic Networks (called also "Frames").

In Chapter 5 we saw how some of the previous knowledge representation techniques have been used in ITS data coding, giving an overview of the most diffused choices (FOL, Production Systems, Epistemic Logics, Semantic Networks, and so on).

In Chapter 6 we provided an overview of the most prominent standards for knowledge representation, both in general-purpose and in didactic domains. These standards define international accepted vocabularies of terms and coding structures to be used in describing knowledge by means of Metadata or Ontologies. Metadata and Ontologies are Semantic Network derived methodologies which join the advantages of a link-based knowledge representation with standardization issues so as to allow data exchange between different platforms. Finally, the LIA didactic knowledge representation model for the m-learning project was outlined, and is being considered for adaptation to Diogene's purposes. Finally, a set of ontology editing tools is outlined, and projects responding to the problem of knowledge systems integration are discussed.

The aim of this survey was to facilitate the choice of a representational paradigm for the Diogene project, taking into account its specific objectives. Such a decision may now be made, based on this survey.