

A Roadmap to Ontology Specification Languages

Oscar Corcho¹, Asunción Gómez-Pérez¹

¹Facultad de Informática, Universidad Politécnica de Madrid. Campus de Montegancedo
s/n. Boadilla del Monte, 28660. Madrid. Spain.

`ocorcho@delicias.dia.fi.upm.es, asun@fi.upm.es`

Abstract. The interchange of ontologies across the World Wide Web (WWW) and the cooperation among heterogeneous agents placed on it is the main reason for the development of a new set of ontology specification languages, based on new web standards such as XML or RDF. These languages (SHOE, XOL, RDF, OIL, etc) aim to represent the knowledge contained in an ontology in a simple and human-readable way, as well as allow for the interchange of ontologies across the web. In this paper, we establish a common framework to compare the expressiveness and reasoning capabilities of “traditional” ontology languages (Ontolingua, OKBC, OCML, FLogic, LOOM) and “web-based” ontology languages, and conclude with the results of applying this framework to the selected languages.

1 Introduction

In the past years, a set of languages have been used for implementing ontologies. Ontolingua [6] is perhaps the most representative of all of them. Other languages have also been used for specifying ontologies: LOOM [16], OCML [17], FLogic [12], etc. Protocols such as OKBC[4] have been also developed to access KR systems. KR paradigms underlying these languages and protocols are diverse: frame-based, description logic, first (and second) order predicate calculus and object-oriented.

In the recent years, new languages for the web have been created -XML [2], RDF [13] and RDF Schema [3]- and are still in a development phase. Other languages for the specification of ontologies, based on the previous ones, have also emerged: SHOE [15], XOL [11] and OIL [10]. Preliminary studies exist on the use of web-based languages for representing ontologies. In [9], an analysis is shown on the role of HTML, XML and RDF when providing semantics for documents on the Web.

The purpose of this paper is to analyse the tradeoff between expressiveness (*what can be said*) and inference (*what can be obtained from the information represented*) in traditional and web-based ontology languages. In Section 2, we will present a framework for evaluating the expressiveness and inference mechanisms of ontology specification languages. Section 3 will describe both the *so-called* traditional ontology languages and the web-based ontology languages. As a conclusion, section 4 presents a discussion on the results of the study.

2 Evaluation Framework

The goal of this section is to set up a framework for comparing the expressiveness and inference mechanisms of potential ontology languages. We use in our analysis the CommonKADS framework [18], which distinguishes between domain knowledge and inference knowledge. Figure 1 summarises the relationship between the KR components and the reasoning mechanisms of languages.

2.1 Domain Knowledge

The *domain knowledge* describes the main static information and knowledge objects in an application domain [18]. We identify the main kind of components used to describe domain knowledge in ontologies. Accordingly to Gruber [8], knowledge in ontologies can be specified using five kind of components: concepts, relations, functions, axioms and instances. Concepts in the ontology are usually organised in taxonomies. Sometimes the notion of ontology is somewhat diluted, in the sense that taxonomies are considered to be full ontologies [19]. Other components like procedures and rules are also identified in some ontology languages (i.e., OCML). For each one of the components outlined before (except for procedures, as it is very difficult to find common characteristics for them in all languages) we will select a set of features that we consider relevant.

Concepts [18], also known as classes, are used in a broad sense. They can be abstract or concrete, elementary or composite, real or fictious. In short, a concept can be anything about which something is said, and, therefore, could also be the description of a task, function, action, strategy, reasoning process, etc. The following questions identify the expressiveness of a language when defining classes:

- Is it possible to define **metaclasses** (classes as instances of other ones)? They are important in case that a KR ontology exists for the language.
- Is it possible to define **partitions** (sets of disjoint classes)?
- Does the language provide mechanisms to define **slots/attributes**? For example:
 - **Local attributes**. Attributes which belong to a specific concept. For instance, attribute *age* belongs to concept *Person*.
 - **Instance attributes (template slots)**. Attributes whose value may be different for each instance of the concept.
 - **Class attributes (own slots)**. Attributes whose value must be the same for all instances of the concept.
 - **Polymorph attributes**. Attributes (slots) with the same name and different behaviour for different concepts. For instance, the attribute *author* for concept *Thesis* is different from the attribute *author* for concept *Book*. Its type for *Thesis* is *Student*, and its type for *Book* is *Person*.
- Does the language provide the following **predefined facets** for attributes?
 - **Default slot value**, which will be used to assign a value to the attribute in case there is no explicit value defined for it.
 - **Type**, which will be used to constrain the type of the attribute.

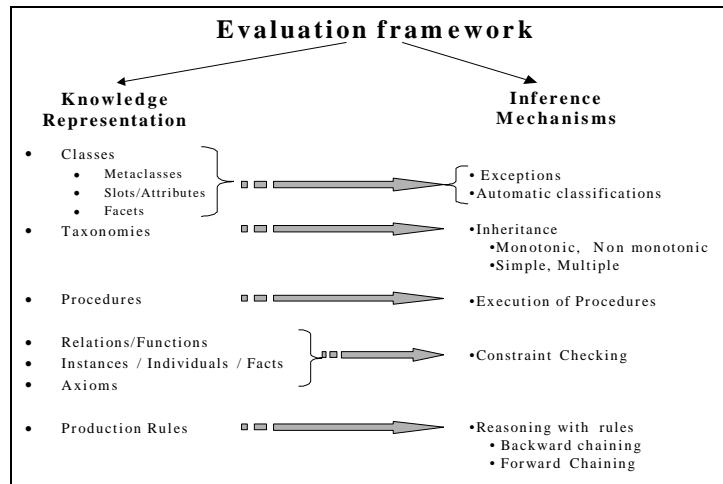


Fig. 1. Evaluation Framework

- **Cardinality constraints**, which will be used to constrain the minimum and maximum number of values of the attribute.
- **Documentation**, which could include a natural language definition for it.
- **Operational definition**, which could include the definition or selection of a formula, a rule, etc to be used, for instance, when obtaining a value for that attribute.
- May **new facets** be created for attributes?

Taxonomies. They are widely used to organise ontological knowledge in the domain using generalisation/specialisation relationships through which simple/multiple inheritance could be applied. Since there exists some confusion regarding the primitives used to build taxonomies, we propose to analyse whether or not the following primitives (which are based on the definitions provided by the frame ontology at Ontolingua) are predefined in the languages.

- **Subclass of** specialises general concepts in more specific concepts.
- **Disjoint decompositions** define a partition as subclass of a class. The classification does not necessarily have to be complete (there may be instances of the parent class that are not included in any of the subclasses of the partition).
- **Exhaustive subclass decompositions** define a partition as subclass of a class. The parent class is the union of all the classes that make up the partition.
- **Not subclass of** may be used to state that a class is not a specialisation of another class. This kind of knowledge is usually represented using the denial of the *subclass of* primitive.

Some languages have a formal semantics for those primitives, and others must define their semantics by using axioms or rules.

Relations [8] represent a type of interaction between concepts of the domain. They are formally defined as any subset of a product of n sets. First, we consider the relationship between relations and other components in the ontology. We will ask if concepts and attributes are considered, respectively, as unary and binary relations. **Functions** [8] are considered as a special kind of relations where the value of the last argument is unique for a list of values of the $n-1$ preceding arguments.

Second, we focus on the arguments (both in relations and functions):

- Is it possible to define **arbitrary n-ary relations/functions**? If this is not possible, which is the maximum number of arguments?
- May the type of arguments be constrained?
- Is it possible to define **integrity constraints** in order to check the correctness of the arguments' value?
- Is it possible to define **operational definitions** to infer values of arguments with procedures, formulas and rules, or to define its semantic using axioms or rules?

Axioms [8] model sentences that are always true. They are included in an ontology for several purposes, such as constraining its information, verifying its correctness or deducting new information. We will focus on the next characteristics:

- Does the language support building axioms in **first order logic**?
- And **second order logic** axioms?
- Are axioms defined as independent elements in the ontology (**named axioms**) or must they be included inside the definition of other elements, such as relations, concepts, etc? This feature improves readability and maintenance of ontologies.

Instances/Individuals/Facts/Claims. All these terms are used to represent elements in the domain. **Instances** [8] represent elements of a given concept. **Facts** [17] represent a relation which holds between elements. **Individuals** [6] refer to any element in the domain which is not a class (both instances and facts). **Claims** [15] represent assertions of a fact by an instance. It is important to highlight the inclusion of claims, since people on internet can make whatever claims they want. Hence, agents shouldn't interpret them as facts of knowledge, but as claims being made by a particular instance about itself or about other instances or data, which may prove to be inconsistent with others [15]. The following questions will be asked in this section:

- Is it possible to define **instances of concepts**?
- Is it possible to define instances of relations (**facts**)?
- Does the language provide special mechanisms to define **claims**?

Production rules. Production rules [16], which follow the structure *If ... Then ...*, are used to express sets of actions and heuristics which can be represented independently from the way they will be used. A set of questions will be asked about them:

- Is it possible to define disjunctive and conjunctive premises?
- May the **chaining mechanism** be defined declaratively?
- Is it possible to define **truth values** or **certainty values** attached to the rule?
- May **procedures** be included **in the consequent**? They are commonly used to change the values of attributes of a concept, add information to the KB, etc.

- Does the language support **updates of the KB**, performed by adding or removing facts or claims?

2.2 Inference Mechanisms

This dimension describes how the static structures represented in the *domain knowledge* can be used to carry out a reasoning process [18]. There is a strong relationship between both dimensions, as the structures used for representing knowledge are the basis for the reasoning process, as seen in Figure 1. We analyse whether the language supports the following features or not:

- Does the language provide an **inference engine** that reasons with the knowledge represented using the language? Is it **sound**? And **complete**?
- Does the inference engine perform **automatic classifications**?
- Does the inference engine deal with **exceptions**? Exceptions are considered when attribute Attribute1 is defined for concept C1 and concept C2, being C1 subclass of C2 and we analyse whether the definition of Attribute1 in concept C1 overrides the definition of Attribute1 in concept C2 or not.
- Is it possible to use **monotonic, non-monotonic, simple** and/or **multiple inheritance**?
- Are **procedures** executable?
- Do axioms perform any kind of **constraint checking**?
- When reasoning with rules, does the language allow **forward and backward chaining**?

3 Ontology Specification Languages

In this section, we show an analysis of ontology specification languages which have been and are widely used by the ontology community (Ontolingua, OKBC, OCML, FLogic and LOOM), other languages created in the context of Internet, which are recommendations of the W3C (XML, RDF and RDFS) and, finally, other new languages for the specification of ontologies (XOL, SHOE and OIL).

3.1 Traditional Ontology Specification Languages

Ontolingua [6] is a language based on KIF [7] and on the Frame Ontology (FO) [6], and it is the ontology-building language used by the Ontolingua Server [6].

KIF (Knowledge Interchange Format) was developed to solve the problem of heterogeneity of languages for knowledge representation. It provides for the definition of objects, functions and relations. KIF has declarative semantics and it is based on first-order predicate calculus, with a prefix notation. It also provides for the representation of meta-knowledge and non-monotonic reasoning rules.

As KIF is an interchange format, it is tedious to use for specification of ontologies per se. The FO, built on top of KIF, is a knowledge representation ontology that allows an ontology to be specified following the paradigm of frames, providing terms such as *class*, *instance*, *subclass-of*, *instance-of*, etc. The FO does not allow to express axioms; therefore, Ontolingua allows to include KIF expressions inside of definitions based on the FO. Summarizing, Ontolingua allows to build ontologies in any of the following three manners: (1) using exclusively the FO vocabulary (axioms cannot be represented); (2) using KIF expressions; (3) using both languages simultaneously.

Currently, an inference engine is being developed for Ontolingua. The OKBC API must be used in case we want to develop a customized one.

OKBC Protocol [4] is an acronym for *Open Knowledge Base Connectivity*, previously known as *Generic Frame Protocol*. It specifies a protocol (not a language). The protocol makes assumptions about the underlying KR system (frames), and it is complementary to language specifications developed to support knowledge sharing.

The GFP Knowledge Model, which is the implicit representation formalism underlying OKBC, supports an object-centered representation of knowledge and provides a set of representational constructs commonly found in frame representation systems: constants, frames, slots, facets, classes, individuals and knowledge bases.

It also defines a complete *tell&ask* interface for knowledge bases accessed using OKBC protocol, and procedures (with a Lisp-like syntax) in order to describe complex operations to perform in a knowledge base when accessing it over a network.

Eventually it has been developed the *OKBC-Ontology* for Ontolingua, which is fully compatible with the OKBC protocol.

In this study, when referring to OKBC we will mean the API, together with the maximum expressiveness permitted.

OCML [17] stands for *Operational Conceptual Modeling Language*, and was originally developed in the context of the VITAL project.

OCML is a frame-based language that provides mechanisms for expressing items such as relations, functions, rules (with backward and forward chaining), classes and instances. In order to make the execution of the language more efficient, it also adds some extra logical mechanisms for efficient reasoning, such as procedural attachments. A general *tell&ask* interface is also implemented, as a mechanism to assert facts and/or examine the contents of an OCML model.

Several pragmatic considerations were taken into account in the development of OCML. One of them is the compatibility with standards, such as Ontolingua, so that OCML can be considered as a kind of “operational Ontolingua”, providing theorem proving and function evaluation facilities for its constructs.

FLogic [12] is an acronym for *Frame Logic*. FLogic integrates frame-based languages and first-order predicate calculus. It accounts in a clean and declarative fashion for most of the structural aspects of object-oriented and frame-based languages, such as object identity, complex objects, inheritance, polymorphic types, query methods, encapsulation, and others. In a sense, FLogic stands in the same relationship to the object-oriented paradigm as classical predicate calculus stands to relational programming.

FLogic has a model-theoretic semantics and a sound and complete resolution-based proof theory.

Applications of FLogic go from object-oriented and deductive databases to ontologies, and it can be combined with other specialized logics (HiLog, Transaction Logic), to improve the reasoning with information in the ontologies.

LOOM [16] is a high-level programming language and environment intended for use in constructing expert systems and other intelligent application programs. It is a descendent of the KL-ONE family and it is based in description logic, achieving a tight integration between rule-based and frame-based paradigms.

LOOM supports a "description" language for modeling objects and relationships, and an "assertion" language for specifying constraints on concepts and relations, and to assert facts about individuals. Procedural programming is supported through pattern-directed methods, while production-based and classification-based inference capabilities support a powerful deductive reasoning (in the form of an inference engine: the classifier).

It is important to focus on the description logic approach to ontology modeling, which differs from the frame-based approach of the previously described languages. Definitions written using this approach try to exploit the existence of a powerful classifier in the language, specifying concepts by using a set of restrictions on them.

3.2 Web Standards and Recommendations

XML [2] stands for *eXtended Markup Language* deriving from SGML (*Standard General Markup Language*). It is being developed by the XML Working Group of the World Wide Web Consortium (W3C), and it is next to become a standard.

As a language for the World Wide Web, its main advantages are: it is easy to parse, its syntax is well defined and it is human readable. There are also many software tools for parsing and manipulating XML. It allows users to define their own tags and attributes, define data structures (nesting them), extract data from documents and develop applications which test the structural validity of a XML document.

When using XML as the basis for an ontology specification language, its main *advantages* are:

- The definition of a common syntactic specification by means of a DTD (*Document Type Definition*).
- Information coded in XML is easily readable for humans.

- It can be used to represent distributed knowledge across several web-pages, as it can be embedded in them.

XML also presents some *disadvantages* which influence on ontology specification:

- It is defined in order to allow the lack of structure of information inside XML tags. This makes it difficult to find the components of an ontology inside the document.
- Standard tools are available for parsing and manipulating XML documents, but not for making inferences. These tools must be created in order to allow inferences with languages which are based on XML.

XML itself has no special features for the specification of ontologies, as it just offers a simple but powerful way to specify a syntax for an ontology specification language (this is the reason why XML is not included in the comparison of section 5). Besides, it can be used for covering ontology exchange needs, exploiting the communication facilities of the WWW.

RDF [13] stands for *Resource Description Framework*. It is being developed by the W3C for the creation of metadata describing Web resources. Examples of the use of RDF in ontological engineering may be analyzed in [1] and [20].

A strong relationship stands between RDF and XML. In fact, they are defined as complementary: one of the goals of RDF is to make it possible to specify semantics for data based on XML in a standardized, interoperable manner. The goal of RDF is to define a mechanism for describing resources that makes no assumptions about a particular application domain nor the structure of a document containing information.

The data model of RDF (which is based in semantic networks) consists of three types: **resources** (subjects), entities that can be referred to by an address at the WWW; **properties** (predicates), which define specific aspects, characteristics, attributes or relations used to describe a resource; and **statements** (objects), which assign a value for a property in a specific resource.

RDF Schema [3] (RDFS) is a declarative language used for the definition of RDF schemas. The RDFS data model (which is based on frames) provides mechanisms for defining the relationships between properties (attributes) and resources. Core classes are *class*, *resource* and *property*; hierarchies and type constraints can be defined (core properties are *type*, *subclassOf*, *subPropertyOf*, *seeAlso* and *isDefinedBy*). Some core constraints are also defined.

3.3 Web-Based Ontology Specification Languages

XOL. [11] stands for *XML-Based Ontology Exchange Language*. It was designed to provide a format for exchanging ontology definitions among a set of interested parties. Therefore, it is not intended to be used for the development of ontologies, but as an intermediate language for transferring ontologies among different database systems, ontology-development tools or application programs.

XOL allows to define in a XML syntax a subset of OKBC, called OKBC-Lite. As OKBC defines a protocol for accessing frame-based representation systems, XOL

may be suitable for exchanging information between different systems, via the WWW. The main handicap is that frames (defined in OKBC) are excluded from this language, and only classes (and their hierarchies), slots and facets can be defined.

Many XML editing tools are available which allow to generate XOL documents.

SHOE [15] stands for *Simple HTML Ontology Extension*. It was developed first as an extension of HTML, with the aim of incorporating machine-readable semantic knowledge in HTML or other WWW documents. Recently, it has been adapted in order to be XML compliant. The intent of this language is to make it possible for agents to gather meaningful information about web pages and documents, improving search mechanisms and knowledge-gathering. The two-phase process to achieve it consists of: (1) defining an ontology describing valid classifications of objects and valid relationship between them; (2) annotating HTML pages to describe themselves, other pages, etc.

In SHOE, an ontology is an ISA hierarchy of classes (called categories), plus a set of atomic relations between them, and inferential rules in the form of simplified horn clauses. Therefore, classes, relations and inferential rules can be defined. An important feature included in SHOE is the ability to make claims about information, as discussed in section 2.

OIL [10], *Ontology Interchange Language*, is a proposal for a joint standard for describing and exchanging ontologies. It is still in an early development phase, and has been designed to provide most of the modelling primitives commonly used in frame-based and description logic ontologies (it is based on existing proposals, such as OKBC , XOL and RDF), with a simple, clean and well defined semantics, and an automated reasoning support.

In OIL, an ontology is a structure made up of several components, organized in three layers: the object level (which deals with instances), the first meta level or ontology definition (which contains the ontology definitions) and the second meta level or ontology container (which contains information about features of the ontology, such as its author). Concepts, relations and functions and axioms can be defined in OIL. The syntax of instances, rules and axioms has not yet been defined.

4 Results and Comparison of Languages

The results of applying the evaluation framework described in section 2 are presented in this section. It is worth mentioning that a common evaluation framework has been used for different knowledge representation languages (and different knowledge representation paradigms, such as frame-based, description logic and object-centered), and that the results have been achieved taking into account the experience of coding, in all the selected languages, an ontology for electronic commerce, which is not shown here due to the lack of space.

The trade-off between the degree of expressiveness and the inference engine of a language (the more expressive, the less inference capabilities) makes it difficult to establish a scoring of languages. Moreover, we claim that different needs in KR exist

nowadays for applications, and some languages are more suitable than others for the specific needs of a given application.

When developing domain ontologies for an application, it is not only necessary to study the KR and reasoning needs for the application, but also the KR and reasoning capabilities provided by the languages. This framework will avoid the developer of ontologies taking blind decisions on the selection of the ontology language(s) to use.

Information in tables of the next sections will be filled using ‘+’ to indicate that it is a supported feature in the language, ‘-’ for non supported features, ‘+/-’ for non supported features, but could manage to support it by doing something, ‘?’ when no information is available and ‘N.D.’ for features which are not restricted, but could be implemented in order to support them. The contents of tables represent the present situation of languages¹ and may change because of the evolution of them.

4.1 Domain Knowledge

Table 1 shows at first glance the main components of the ontology specification languages selected for this study.

Concepts, n-ary relations and instances can be defined easily in almost all languages. In OKBC and FLogic, which are frame-based languages, relations can be represented by using frames, but not as special elements provided by the language. In OKBC, axioms are only supported in the tell&ask part of the API, although neither deductive nor storage guarantees are made for all OKBC implementations.

Table 1. Definition of the main components of domain knowledge

	Onto	OKBC	OCML	LOOM	FLogic	XOL	SHOE	RDF(S)	OIL
Concepts	+	+	+	+	+	+	+	+	+
n-ary relations	+	+/-	+	+	+/-	-	+	+	+
Functions	+	+/-	+	+	+/-	-	-	-	+
Procedures	+	+	+	+	-	-	-	-	-
Instances	+	+	+	+	+	+	+	+	ND
Axioms	+	+/-	+	+	+	-	-	-	ND
Production Rules	-	-	+	+	-	-	-	-	ND
Formal semantics	+	+	+	+	+	+	-	-	-

Functions, procedures and axioms cannot be defined using web-based languages, except for some restricted forms of axioms, such as deductive rules, which are definable in SHOE.

It is worth mentioning that procedures are only definable in Lisp-based languages, and production rules are just definable in OCML and LOOM.

An additional row has been added to the table, analysing the presence of a formal semantics: some web-based languages, such as SHOE, RDF(S) and OIL lack of it, whereas traditional languages and XOL provide it.

¹ 'Onto' will be used to refer to Ontolingua. RDF(S) is the acronym used to refer to the combination of RDF and RDFS.

Concepts. Table 2 summarizes the most important features to be analyzed when describing concepts in an ontology. It is divided in 4 sections: metaclasses, partitions, definition of attributes and definitions of properties of attributes (facets).

Table 2. Definition of concepts

CONCEPTS	Onto	OKBC	OCML	LOOM	FLogic	XOL	SHOE	RDF(S)	OIL
Metaclasses	+	+	+	+	+	+	-	+	-
Partitions	+	-	-	+	-	-	-	-	-
ATTRIBUTES									
Template (instance attrs)	+	+	+	+	+	+	+	+	+
Own (class attrs.)	+	+	+	+	+	+	-	+	+/-
Polymorphic	+	+	+	+	+	-	-	-	+
Local scope	+	+	+	+	+	+	+	+	+
FACETS									
Default slot value	-	+	+	+	+	+	-	-	-
Type constraint	+	+	+	+	+	+	+	+	+
Cardinality constraints	+	+	+	+	+/-	+	-	-	+
Documentation	+	+	+	+	-	+	+	-	+
Procedural knowledge	-	-	+	+	-	-	-	-	-
Adding new facets	+	+	-	+	-	-	-	-	-

Only SHOE and OIL do not allow to define metaclasses, and partitions can only be defined in Ontolingua and LOOM.

Instance attributes and type constraints for attributes can be defined using any of the chosen languages. The results of the rest of the values depend on the languages, although a glance at the table shows us that traditional ontology languages allow us, again, to define more features than web-based languages.

Procedural knowledge inside the definition of attributes is only supported by OCML and LOOM, due to their operational behavior. It must be included in the definition of the OCML's attributes by means of special keywords, such as *:prove-by* or *:lisp-fun*, not as simple facets, or in the definition of the LOOM's attributes by means of keywords such as *:sufficient*, *:is*, *:is-primitive* or *:implies*.

FLogic just allows to define the maximum cardinality for slots as 1 or N, while the minimum cardinality is always set to 0.

Table 3. Definition of taxonomies

TAXONOMIES	Onto	OKBC	OCML	LOOM	FLogic	XOL	SHOE	RDF(S)	OIL
Subclass of	+	+	+	+	+	+	+	+	+
Exhaustive subclass partitions	+	-	+/-	+	+/-	-	-	-	-
Disjoint Decompositions	+	-	+/-	+	+/-	-	-	-	+/-
Not subclass of	+/-	-	-	+/-	-	-	-	-	+

Taxonomies. When defining taxonomies, there is just one primitive predefined in all languages and correctly handled by them: *subclass of*. Ontolingua and LOOM are the only languages which have the rest of primitives (except for *not subclass of*, which must be declared using the denial of primitive *subclass-of*). These primitives can be defined as relations in the rest of languages, but as a consequence, there is no special treatment for them. In FLogic, axioms must be defined in order to provide the semantics for them. OIL allows to define the primitive *not subclass-of*; hence it is also possible to define disjoint decompositions.

Relations and Functions. Relations are very important components in an ontology (hence they are supported by almost all the ontology languages), but not every desirable characteristic of relations is implemented in all languages. Functions are not included in some languages.

Table 4. Definition of relations and functions

RELATIONS FUNCTIONS	Onto	OKBC	OCML	LOOM	FLogic	XOL	SHOE	RDF(S)	OIL
Functions as relations	+	+	-	+	+	-	-	-	+
Concepts: unary rels.	+	+	+	+	-	-	+	-	+
Slots: binary rels.	+	+	+	+	-	+	+	+	+
n-ary rels./functs.	+	+/-	+	+	+/-	-	+	+	+/-
Type constraints	+	+	+	+	+	-	+	+	+
Integrity constraints	+	+	+	+	+	-	-	-	-
Operational defs.	-	-	+	+	+	-	-	-	-

Many languages represent concepts as unary relations. Attributes are usually considered as binary relations, except for FLogic, where they are considered as ternary ones.

Great semantic differences are found when analysing the role that functions play in different languages. Some languages, such as KIF (and consequently, Ontolingua), consider functions as a special case of relations in which the n^{th} element of the relation is unique for the $n-1$ preceding elements. LOOM consider functions as relations where the result can be calculated given the domain arguments. In OCML, functions are considered as modelling elements which play a role which is completely different to the one of relations. In FLogic, functions are considered as methods which are defined inside a concept. Their value is calculated by using a deductive rule associated to the method previously declared.

FLogic, OKBC, RDF(S) and OIL cannot define n-ary relations directly. They must define them as associative classes or by means of several binary relations.

All languages allow the definition of type constraints for arguments, and the main differences among traditional and web-based ontology languages lay on the definition of integrity constraints (the last ones don't allow to define them).

The last comments are on operational definitions for relations: just OCML, LOOM and FLogic allow to define operations inside relations, although there is a difference between them: while LOOM provides operational definitions just for an inferential

purpose, OCML also provides non-operational definitions which can be used for representational purposes [17]. In FLogic, this kind of operations must be defined by using axioms, which are defined apart. Ontolingua does not support user-defined Lisp lambda bodies for relations, but it has certain relations that have procedural attachments which are activated by the tell&ask interface (for instance, asking (+ 3 2 ?x) will reply with a single binding of 5 for ?x).

Instances. Instances of concepts and of relations (facts) are supported by all the languages. Claims, however, are just allowed in some of the web-based ontology languages. This is due to the fact that the management of information which comes from different sources is an intrinsic characteristic of the web environment and so these languages have specialised ways to treat this information.

Table 5. Definition of instances

INSTANCES	Onto	OKBC	OCML	LOOM	FLogic	XOL	SHOE	RDF(S)	OIL
Instances of concepts	+	+	+	+	+	+	+	+	ND
Facts	+	+	+	+	+	+	+	+	ND
Claims	-	-	-	-	-	-	+	+	ND

Axioms. This is a good measure of expressiveness. The richest the axioms defined, the more expressive the language is. Ontolingua allows the definition of first-order and second-order logic axioms. OCML and FLogic also allow to define first-order logic axioms independently of the rest of components of the ontology. LOOM just allows to define first-order logic axioms inside the definitions of relations, concepts and functions.

The rest of languages, except for XOL, only allow restricted types of axioms. So, OKBC just supports a subset of the axioms which can be represented with KIF (and they must be included as a frame or by using the tell&ask interface), and SHOE just allows to define deductive rules. In OIL, the syntax of axioms has not yet been defined, while in RDF(S) several studies are currently trying to specify the syntax and semantics for the most commonly used axioms.

Table 6. Definition of axioms

AXIOMS	Onto	OKBC	OCML	LOOM	FLogic	XOL	SHOE	RDF(S)	OIL
1st-order logic	+	+/-	+	+	+	-	+/-	+/-	ND
2nd order logic	+	+/-	-	-	-	-	-	-	-
Named axioms	+	+	+	-	-	-	-	-	-

Production rules. Production rules are components of an ontology in OCML and LOOM. LOOM distinguishes between purely deductive rules and side-effecting, procedural rules (production rules). OCML makes the same distinction, defining “backward” and “forward” ones. Therefore, OCML and LOOM allow to define the chaining when performing the reasoning with knowledge defined in the ontology.

As far as OIL is concerned, rules are just a weak form of general inclusion axioms.

Finally, SHOE does not allow to define production rules, but inference rules, as stated in the previous section.

Table 7. Definition of rules

PRODUCTION RULES	Onto	OKBC	OCML	LOOM	FLogic	XOL	SHOE	RDF(S)	OIL
PREMISES									
Conjunctive	-	-	+	+	-	-	-	-	ND
Disjunctive	-	-	+	+	-	-	-	-	ND
CONSEQUENT									
Truth values	-	-	-	-	-	-	-	-	ND
Execution of procedures	-	-	+/-	+	-	-	-	-	ND
Updating the KB	-	-	+	+	-	-	-	-	ND

4.2 Reasoning

A clear distinction between KR and reasoning exists for all languages, except for OCML. For instance, Ontolingua is maybe the most expressive of all the languages chosen for this study, but there is no inference engine implemented for it. OCML allows to define some features concerning reasoning inside representational elements (for instance, rules can be defined as backward rules or forward ones, so that the chaining is explicitly defined).

Just FLogic and OIL inference engines are sound and complete, which is a desirable feature, although it can make representation in the language more difficult.

Automatic classifications are performed by description logic-based languages (LOOM and OIL).

The exception handling mechanism is not addressed, in general, by language developers (FLogic is the only one handling exceptions). Works have been carried out in other languages, such as LOOM, to support them.

Single and multiple inheritance is also supported by most of the languages (except for XOL), but conflicts in multiple inheritance are not resolved. All languages are basically monotonic, although they usually include some non-monotonic capabilities. For instance, the only non-monotonic capabilities present in both Ontolingua and OCML are related to default values for slots and facets. In XOL and RDF specifications there is no explicit definition of the behaviour of inherited values.

All the languages which allow to define procedures, allow to execute them.

Constraint checking is performed in all the traditional ontology languages. Information about constraint checking in XOL is not available. In OKBC, constraint checking is guaranteed to be included in all implementations of it. However, it can be parameterised and even switched off. Constraint checking in SHOE is not performed because conflicts are thought to be frequent in the Web, and resolving them will be problematic. However, type constraint checking is performed when necessary.

Chaining used in SHOE is not defined in the language: freedom exists so that each implementation may choose between any of them. OCML allows to define the chaining of rules when defining them, although default chaining used is the backward one. LOOM performs both kinds of chaining, and FLogic's one is in between.

Table 8. Reasoning mechanisms of the language

REASONING	Onto	OKBC	OCML	LOOM	FLogic	XOL	SHOE	RDF(S)	OIL
INFERENCE ENG.									
Sound	-	-	+	+	+	-	-	-	+
Complete	-	-	-	-	+	-	-	-	+
CLASSIFICATION									
Automatic classif.	-	-	-	+	-	-	-	-	+
EXCEPTIONS									
Exception handling	-	-	-	-	+	-	-	-	-
INHERITANCE									
Monotonic	+	+	+	+	+	ND	+	ND	+
Non-monotonic	+/-	+	+/-	+	+	ND	-	ND	-
Single Inheritance	+	+	+	+	+	ND	+	+	+
Multiple inheritance	+	+	+	+	+	ND	+	+	+
PROCEDURES									
Execution of procedures	+	+	+	+	-	-	-	-	-
CONSTRAINTS									
Constraint checking	+	+	+	+	+	-	-	-	-
CHAINING									
Forward	-	-	+	+	+	-	ND	-	-
Backward	-	-	+	+	+	-	ND	-	-

5 Future works

Future works in this area will try to identify factors to choose among a set of languages when building a domain ontology for an application. Different needs in KR and reasoning exist, and some languages are more suitable than others. We recommend:

- Web based languages for the interchange of ontologies on the web.
- Traditional languages for the representation – modeling – of ontologies with high expressiveness needs. However, if ontologies are considered just as taxonomies, the use of web-based languages is not a problem.
- For performing reasoning inside agents, XML-based languages do not provide inference engines. However, some of the traditional ontology languages not only provide them but also translators to other computable languages.

Besides, an analysis of the existing tools for editing, managing, integrating and translating ontologies (which would extend the one described in [5]) will be useful for determining the most suitable language for our needs, and studies on the treatment of namespaces in different languages will be also interesting to analyse the easiness of integrating and scaling up ontologies.

Finally, the analysis on how components are codified in each language will also help to face up to the translation problem.

Acknowledgements

This paper would not be possible without comments and feedback of developers and users of the mentioned languages who verified our tables: V. K. Chaudhri (XOL), Stefan Decker (FLogic), Belén Díaz (LOOM), Yolanda Gil (LOOM), Jeff Heflin (SHOE), Ian Horrocks (OIL), Enrico Motta (OCML), James Rice (Ontolingua and OKBC) and Tom Russ (LOOM).

References

1. Amann, B., Fundulaki, I. *Integrating Ontologies and Thesauri to Build RDF Schemas*. 1999.
2. Bray, T., Paoli, J., Sperberg, C. *Extensible Markup Language (XML) 1.0*. W3C Recommendation. Feb 1998. <http://www.w3.org/TR/REC-xml>.
3. Brickley, D., Guha, R.V. *Resource Description Framework (RDF) Schema Specification*. W3C Proposed Recommendation. March, 1999. <http://www.w3.org/TR/PR-rdf-schema>.
4. Chaudhri, V., Farquhar, A., Fikes, R., Karp, P., Rice, J. *The Generic Frame Protocol 2.0*. July, 1997.
5. Duineveld, A., Studer, R., Weiden, M., Kenepa, B., Benjamins, R. *WonderTools? A comparative study of ontological engineering tools*. Proceedings of KAW99. Banff. 1999.
6. Farquhar, A., Fikes, R., Rice, J. *The Ontolingua Server: A Tool for Collaborative Ontology Construction*. Proceedings of KAW96. Banff, Canada, 1996.
7. Genesereth, M., Fikes, R. *Knowledge Interchange Format*. Technical Report. Computer Science Department. Stanford University. Logic-92-1. 1992.
8. Gruber, R. *A translation approach to portable ontology specification*. Knowledge Acquisition. #5: 199-220. 1993.
9. van Harmelen, F., Fensel, D. *Surveying notations for machine-processable semantics of Web sources*. Proceedings of the IJCAI'99 Workshop on Ontologies & PSMs. 1999.
10. Horrocks, I., Fensel, D., Harmelen, F., Decker, S., Erdmann, M., Klein, M. *OIL in a Nutshell*. Proceedings of the ECAI'00 Workshop on Application of Ontologies and PSMs. Berlin. Germany. August, 2000.
11. Karp, R., Chaudhri, V., Thomere, J. *XOL: An XML-Based Ontology Exchange Language*. July, 1999.
12. Kifer, M., Lausen, G., Wu, J. *Logical Foundations of Object-Oriented and Frame-Based Languages*. Journal of the ACM. 1995.
13. Lassila, O., Swick, R. *Resource Description Framework (RDF) Model and Syntax Specification*. W3C Recommendation. January, 99. <http://www.w3.org/TR/PR-rdf-syntax>.
14. Lenat, D.B., Guha, R.V. *Building Large Knowledge-based systems. Representation and Inference in the Cyc Project*. Addison-Wesley. Reading. Massachusetts. 1990.
15. Luke S., Heflin J. *SHOE 1.01. Proposed Specification*. SHOE Project. February, 2000. <http://www.cs.umd.edu/projects/plus/SHOE/spec1.01.htm>
16. MacGregor, R. *Inside the LOOM classifier*. SIGART bulletin. #2(3):70-76. June, 1991.
17. Motta, E. *Reusable Components for Knowledge Modelling*. IOS Press. Amsterdam. 1999.
18. Schreiber, G., Akkermans, H., Anjewierden, A. *Knowledge engineering and management. The CommonKADS Methodology*. MIT press, Massachusetts. 1999.
19. Studer, R., Benjamins, R., Fensel, D. *Knowledge Engineering: Principles and Methods*. DKE 25(1-2), pp:161-197. 1998
20. *Using Protégé-2000 to Edit RDF*. Technical Report. Knowledge Modelling Group. Stanford University. February, 2000. <http://www.smi.Stanford.edu/projects/protege/protege-rdf/protege-rdf.html>