# Evaluating Knowledge Representation and Reasoning Capabilities of Ontology Specification Languages

**Oscar Corcho**[1] and **Asunción Gómez-Pérez**[2]

**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 of "traditional" ontology languages (Ontolingua, OKBC, OCML, FLogic, LOOM) and "web-based" ontology languages. As a result of this study, we conclude that different needs in KR and reasoning may exist in the building of an ontology-based application, and these needs must be evaluated in order to choose the most suitable ontology language(s).

## 1 INTRODUCTION

In the past years, a set of languages have been used for implementing ontologies. Ontolingua [1] is the most representative of all of them, and it is considered as a standard by the ontology community. Other languages have also been used for specifying ontologies: LOOM [2], CycL [3], OCML [4], FLogic [5], etc. KR paradigms underlying these languages are diverse: frame-based, description logic, first (and second) order predicate calculus, object-oriented, etc.

In the recent years, new web standard languages have been created -XML [6], RDF [7]- and are still in a development phase. As a consequence of this ever-changing context, new XML-based ontology specification languages have also emerged: SHOE [8], XOL [9], OIL [10], as well as RDF Schema [11] and XML Schema [12]. The role of new languages in this scenario is twofold: they can be used to provide the semantics of information contained in electronic documents or can be used for the exchange of ontologies across the web. A study about ontologies and web-based languages for representing them is presented at [13], where 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 readability (*how things are said*), 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 potential languages which could be used to specify ontologies. It is based on a set of criteria that we consider relevant from the knowledge representation (KR) and inference mechanisms point of view. Section 3 will describe the *so-called* traditional ontology languages. Section 4 will focus on web-based ontology languages. As a conclusion, section 5 presents a discussion on the results of the study.

[1] Facultad de Informática. Universidad Politécnica de Madrid. Campus de Montegancedo s/n. Boadilla del Monte, 28660. Madrid. Spain. Tel: +34 91 3366604, Fax: +34 91 3367412, email: ocorcho@delicias.dia.fi.upm.es

[2] Facultad de Informática. Universidad Politécnica de Madrid. Campus de Montegancedo s/n. Boadilla del Monte, 28660. Madrid. Spain. Tel: +34 91 3367439, Fax: +34 91 3367412, email: asun@fi.upm.es

## 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 framework proposed in CommonKADS [14], which distinguishes between domain knowledge and inference knowledge. Figure 1 summarizes the main dimensions of the framework and the relationship between the KR components and the reasoning mechanisms of the language.

### 2.1. Domain knowledge

The *domain knowledge* describes the main static information and knowledge objects in an application domain [14]. We identify the main kind of components used to describe domain knowledge in ontologies. Gruber [15] stated that knowledge in ontologies can be formalized using five kind of components: concepts, relations, functions, axioms and instances. Concepts in the ontology are usually organized in taxonomies. Sometimes the notion of ontology is somewhat diluted, in the sense that taxonomies are considered to be full ontologies [16]. 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.

#### *2.1.1. Concepts*

Concepts [14] are used in a broad sense. They can be abstract or concrete, elementary (electron) or composite (atom), 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 try to identify the expressiveness of a given language when we define concepts:

- Is it possible to define **Metaclasses** (classes as instances of other ones)?

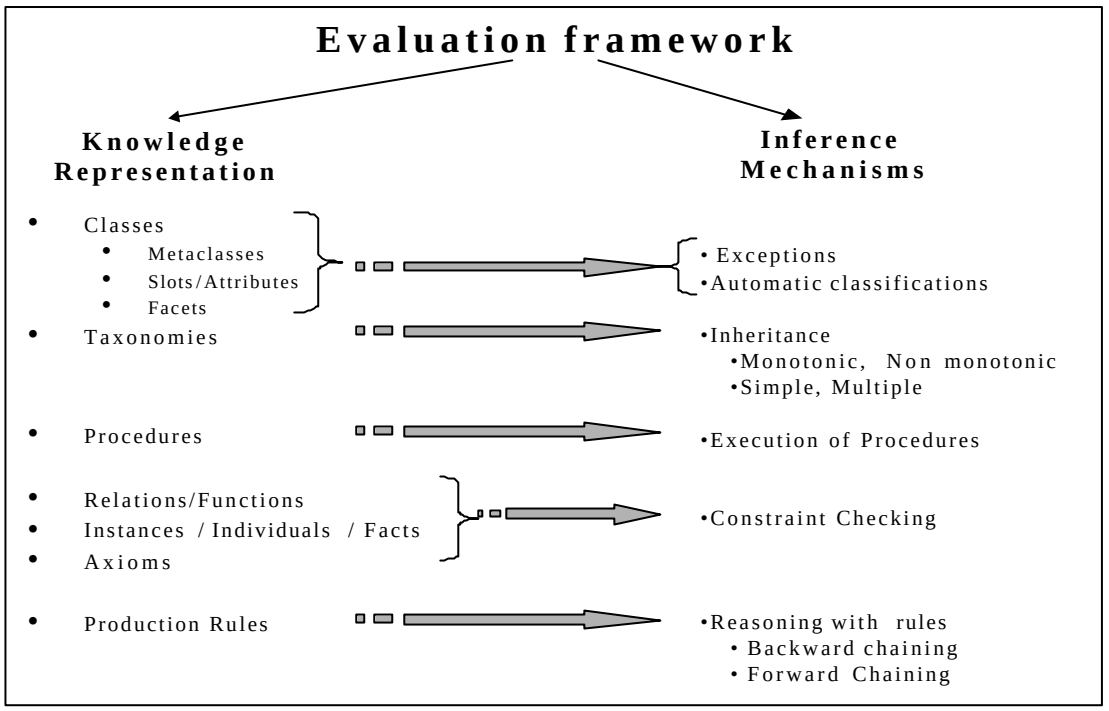


**Figure 1. Evaluation framework.**

- 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.
  - **Cardinality constraints**, which will be used to constrain the minimum and maximum number of values of the attribute.
  - **Documentation**, which will allow to include a natural language definition for the attribute.
  - **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?

### *2.1.2. Taxonomies*

They are widely used to organize ontological knowledge in the domain using generalization/specialization relationship through which simple/multiple inheritance could be applied. Since there exist some confusion regarding the primitives used to build taxonomies, we propose to analyse whether or not the following primitives are predefined in the languages. Their semantic is based on the definitions provided by the frame ontology at Ontolingua [1].

- **Subclass of** specializes general concepts in more specific concepts.
- **Partitions** define a set of disjoint classes.
- **Disjoint decompositions** define the set of disjoint subclasses as subclasses of the parent class. This classification does not necessarily have to be complete, that is, there may be instances of the parent class that are not included in any of the subclasses of the partition.
- **Exhaustive subclass decompositions** define the set of disjoint subclasses of the partition as subclasses of the parent class, where the parent class is defined as a union of all the classes that make up the partition.
- **Not subclass of** may be used when we wish to state that a given class is not a specialization of another class. Usually this kind of knowledge can be represented using denial of *subclass of* primitive.

Some languages do not use the above primitives, but they allow to define them as relations, and their semantic is defined using axioms or rules.

### *2.1.3. Relations*

Relations [15] represent a type of interaction between concepts of the domain. They are formally defined as any subset of a product

of n sets, that is, R: C1 x C2 x … x Cn. Examples of binary relations are *part-of* and *connected-to*.

First, we consider the relationship between relations and other components in the ontology. We will ask if concepts are considered as unary relations and if attributes are considered as binary relations. Special attention deserve **functions** [4], which are defined as mappings between a list of input arguments and its output argument. Formally, functions are defined as F: C1 x C2 x … x Cn-1 → Cn. In this case, we should ask if they are considered as a special kind of relations.

Second, related to 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?
- And **operational definitions** to infer values of arguments by means of procedures, formulas and rules, or to define its semantic using axioms or rules?

#### *2.1.4. Axioms*

Axioms [1] are used to model sentences that are always true. They can be included in an ontology for several purposes, such as constraining the information contained in the ontology, 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?

#### *2.1.5. Instances/Individuals/Facts/Claims*

All these terms are used to represent elements in the domain. Instances usually represent elements of a given concept. **Facts** [4] is the term commonly used to represent a relation which holds between elements. **Individuals** [1] are used in Ontolingua and OKBC to refer to any element in the domain which is not a class (both instances and facts). The term **Claims** [8] refers to the assertion of a fact by an instance. Special attention deserves the inclusion of claims, since people on internet can make whatever claims they want. Hence, agents shouldn't interpret claims 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 [8].

The questions to be asked in this section are the following ones:

- 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**?

#### *2.1.6. Production rules*

Production rules [2], 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 so-called *domain knowledge* can be used to carry out a reasoning process [14]. There is a strong relationship between inference mechanisms and domain knowledge components, as the structures used for representing knowledge are the basis for the reasoning process, as seen in Figure 1. We analyse the following features, asking whether they are supported by the language:

- 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 in the sense that 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 **inheritance?** Which kind of inheritance is allowed: monotonic, non monotonic, simple and/or multiple?
- Are **procedures executable**?
- Is it performed any kind of **constraint checking** by using axioms defined in the language?
- When reasoning with rules, does the language allow to perform **forward and backward chaining**?

## 3 TRADITIONAL ONTOLOGY SPECIFICATION LANGUAGES

In this section, we make an analysis of languages which can be considered as standards for the ontology community (Ontolingua, OKBC, OCML, FLogic and LOOM). They will serve as a reference for the comparative study presented in section 5.

### 3.1. Ontolingua

Ontolingua [1] is a language based on KIF [17] and on the Frame Ontology [15], and it is the ontology-building language used by the Ontolingua Server [1].

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 the first-order predicate calculus, with a prefix notation. It also provides for the representation of meta-knowledge and allows for the representation of non-monotonic reasoning rules.

As KIF is an interchange format, it is tedious to use for specification of ontologies per se. However, the Frame Ontology [15], built on top of KIF, allows an ontology to be specified following the paradigm of frames (it is a knowledge representation

ontology for modeling ontologies under a frame-based approach). Terms like: class, instance, subclass-of, instance-of, etc are included in this ontology.

Since the Frame Ontology is less expressive than KIF, that is, not all of the knowledge that can be expressed in KIF can be expressed using the *Frame-Ontology*, Ontolingua allows to include KIF expressions inside of definitions based on the *Frame-Ontology*. So, the Ontolingua language allows to build ontologies in any of the following three manners: (1) using exclusively the Frame Ontology vocabulary (it is not possible to represent axioms); (2) using KIF expressions; (3) using both languages simultaneously, depending on ontology developer preferences.

Currently, an inference engine is being developed for Ontolingua. However, in case we want to develop a customized one, we must build it using the OKBC API (which will be defined later on this section).

## 3.2. OKBC

OKBC [18] is an acronym for *Open Knowledge Base Connectivity*, previously known as *Generic Frame Protocol*. It specifies a protocol for accessing knowledge bases stored in frame knowledge representation systems, and it is considered 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.

## 3.3. OCML

OCML [4] stands for Operational Conceptual Modeling Language. It was originally developed at the Knowledge Media Institute (UK) in the context of the VITAL project to provide operational modeling capabilities for the VITAL workbench. The current version of the language is v6.3.

It provides mechanism 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.

## 3.4. FLogic

FLogic [5] is an acronym for *Frame Logic*. FLogic is a language which 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. These features include 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. A small number of fundamental concepts that come from object-oriented programming have direct representation in FLogic; other, secondary aspects of this paradigm are easily modeled as well.

## 3.5. LOOM

LOOM [2] 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 of languages, characterized for their efficient automatic classifiers. LOOM achieves a tight integration between rule-based and frame-based paradigms.

It 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). All of these capabilities reside in a framework of query-based assertion and retrieval.

# 4 WEB LANGUAGES FOR BUILDING ONTOLOGIES

This section provides an analysis of new languages created in the context of Internet (XML, RDF, XOL, SHOE and OIL), which are the motivation of this study. First, a state of the art in web standards is given. Second, we describe these web languages which are used for building ontologies.

## 4.1. Web standards

### *4.1.1. XML*

XML [6] metalanguage derives from SGML (Standard General Markup Language). It is being developed by the XML Working Group of the World Wide Web Consortium (W3C), for ease of implementation and interoperability with both SGML and HTML.

As a language for the World Wide Web, its main advantages are the following: 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, as XML is widely used. XML 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 (XML-based ontology languages), 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 (although it is not intended to be used for the direct coding of ontologies, information of the ontology coded in an XML-based ontology language can be easily read and understood).
- 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 may influence on ontologies specified in it:

- The standard is defined in order to allow the lack of structure of information inside XML tags, which makes it difficult to find the components of an ontology inside the same 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. Therefore, XML will be used for two purposes: for providing the syntax of a set of languages, such as XOL or OIL, so that the definition of these languages just consists of describing the semantics of new tags created and used in it; and for covering ontology exchange needs, exploiting the communication facilities of the World Wide Web.

These are the reasons why XML is not included in the comparison performed in section 5.

#### *4.1.2. RDF(S)*[3]

RDF [7] 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 and RDF schemas in ontological engineering may be analyzed in [19] 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 broad 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 consists of three object types: resources (subjects), which are 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.

The RDF data model does not provide itself mechanisms for defining the relationships between properties (attributes) and resources. This is the role of RDFS, the acronym for RDF Schema Specification language [11], which is a declarative language used for the definition of RDF schemas[4]. It is based on some ideas from knowledge representation (semantic nets, frames and predicate logic), but it is much simpler to implement (and also less expressive) than full predicate calculus languages such as CycL and KIF. 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.

A conclusion is that an ontology defined in RDF(S) will lack from functions and axioms, but concepts, relations and instances (as well as claims) can be easily defined.

## 4.2. Web standards and ontologies

#### *4.2.1. XOL*

XOL [9] stands for XML-Based Ontology Exchange Language. XOL 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.

However, since XOL files are textual, a text editor or XML editor may be used to author XOL files. It is expected that many XML tools will soon be available so that XOL documents will be easily generated with them.

#### *4.2.2. SHOE*

SHOE [8] stands for Simple HTML Ontology Extension. It is being developed at the University of Maryland.

SHOE was first an extension of HTML, with the aim of incorporating machine-readable semantic knowledge in HTML or other World Wide Web 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 objetcs 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 (also called categories), plus a set of atomic relations between them, and a set of 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.

#### *4.2.3. OIL*

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

---

[3] RDF(S) is the acronym commonly used to refer to the combination of RDF and RDFS.

[4] An RDF schema consists of the declaration of attributes and their corresponding semantics in the context of RDF

| | Ontol[5] | OKBC | OCML | LOOM | FLogic | XOL | SHOE | RDF(S) | OIL |
|---|---|---|---|---|---|---|---|---|---|
| **Concepts** | + | + | + | + | + | + | + | + | + |
| **Relations** | + | +/- | + | + | +/- | - | + | + | + |
| **Functions** | + | +/- | + | + | +/- | - | - | - | + |
| **Procedures** | + | + | + | + | - | - | - | - | - |
| **Instances** | + | + | + | + | + | + | + | + | - |
| **Axioms** | + | +/- | + | + | + | - | - | - | + |
| **Production rules** | - | - | + | + | - | - | - | - | +/- |

**Table 1.** Definition of the main elements of domain knowledge.

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

**Table 2.** Definition of concepts.

level or ontology container (which contains information about features of the ontology, such as its author).

Concepts, relations, functions and axioms can be defined using OIL's ontology definitions.

## 5 RESULTS AND COMPARISON OF LANGUAGES

The results of applying the evaluation framework described in section 2 are described 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 logics and object-centered).

Information in tables of sections 5.1 and 5.2 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 languages and may change because of the evolution of them.

### 5.1. Domain knowledge

The information contained in Table 1 shows at first glance the main characteristics of the ontology specification languages selected for this study. It can also be used to compare the types of information that can be represented when using them.

Concepts, 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.

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.

#### *5.1.1. Concepts*

Table 2 summarizes the most important features that a language must provide when describing concepts in an ontology. It is divided in three sections: metaclasses, definition of attributes and definitions of properties of attributes (facets).

Note that not all languages allow the definition of meta-classes, which restricts the expressiveness that can be achieved with a language which does not support them.

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.

5 'Ontol' will be used to refer to Ontolingua.

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

**Table 3.** Definition of taxonomies of concepts.

| RELATIONS/FUNCTIONS | Ontol | OKBC | OCML | LOOM | FLogic | XOL | SHOE | RDF(S) | OIL |
|---|---|---|---|---|---|---|---|---|---|
| **Functions as relations** | + | + | - | + | + | - | - | - | + |
| **Concepts as unary relations** | + | + | + | + | - | - | + | - | + |
| **Slots as binary relations** | + | + | + | + | - | - | + | + | + |
| **n-ary relations/functions** | + | +/- | + | + | +/- | - | + | + | +/- |
| **Type constraints** | + | + | + | + | + | - | + | + | + |
| **Integrity constraints** | + | + | + | + | + | - | - | - | - |
| **Operational definitions** | - | - | + | + | + | - | - | - | - |

**Table 4.** Definition of relations and functions.

| INSTANCES | Ontol | OKBC | OCML | LOOM | FLogic | XOL | SHOE | RDF(S) | OIL |
|---|---|---|---|---|---|---|---|---|---|
| **Instances of concepts** | + | + | + | + | + | + | + | + | - |
| **Facts** | + | + | + | + | + | + | + | + | - |
| **Claims** | - | - | - | - | - | - | + | + | - |

**Table 5.** Definition of instances.

| AXIOMS | Ontol | OKBC | OCML | LOOM | FLogic | XOL | SHOE | RDF(S) | OIL |
|---|---|---|---|---|---|---|---|---|---|
| **First-order logic** | + | +/- | + | + | + | - | +/- | +/- | +/- |
| **Second-order logic** | + | +/- | - | - | - | - | - | - | - |
| **Named axioms** | + | + | + | - | - | - | - | - | - |

**Table 6.** Definition of axioms.

### *5.1.2. 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 primitve *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. Again, traditional ontology languages are more expressive.

### *5.1.3. Relations and functions*

We will see how ontology languages allow to define relations and functions in ontologies. Relations are very important elements 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.

Many languages represent concepts as unary relations, so that they can be used in the ontology as if they were relations; the rest of languages clearly distinguish concepts and relations (they are different components). 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 ontology languages lays on the definition of integrity constraints (the last ones don't allow to define these kinds of constraints for relations).

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 [4]. 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 does have 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*).

| PRODUCTION RULES | Ontol | OKBC | OCML | LOOM | FLogic | XOL | SHOE | RDF(S) | OIL |
|---|---|---|---|---|---|---|---|---|---|
| **PREMISES** | | | | | | | | | |
| **Conjunctive** | - | - | + | + | - | - | - | - | **N.D.** |
| **Disjunctive** | - | - | + | + | - | - | - | - | **N.D.** |
| **CONSEQUENT** | | | | | | | | | |
| **Truth values** | - | - | - | - | - | - | - | - | **N.D.** |
| **Execution of procedures** | - | - | +/- | + | - | - | - | - | **N.D.** |
| **Updating of the KB** | - | - | + | + | - | - | - | - | **N.D.** |

**Table 7.** Definition of rules.

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

**Table 8.** Reasoning mechanisms of the language.

#### *5.1.4. Instances*

Instances of concepts and of relations (facts) are supported by all the languages. Claims, however, are just allowed by some of the web 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 specialized ways to treat this information.

#### *5.1.5. Axioms*

This is a good measure of expressiveness. The richest the axioms can be defined, the more expressive the language is. This is the case of Ontolingua, which 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.

#### *5.1.6. Production rules*

Production rules are components of an ontology in OCML, LOOM and OIL.

LOOM makes a distinction 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.

### 5.2. 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 parametrized 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.

## 5.3. Conclusion

Once studied the main components of a given ontology language and knowing the KR and reasoning mechanisms needed for a given application, this framework will avoid blind decisions on the selection of ontology languages. We claim that different needs in KR exist nowadays for applications, and some languages are more suitable than others. So:

- For interchanging ontologies on the web, we strongly recommend web based languages.
- For representing – modeling – ontologies with high expressiveness needs, we recommend traditional ontology languages. However, if ontologies are considered just as taxonomies, the use of XML-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.

An additional analysis of the existing tools to build ontologies could be also useful in the task of determining which one is more suitable for our needs. A good analysis can be found in [21].

This evaluation framework is being used in the context of the MKBEEM [6] project, IST project number 1999-10589[7], which aims to create a multilingual electronic marketplace for companies in Europe. Each user of the ontologies to be developed for this project has filled the information in the tables presented in section 5. The union of all the characteristics expressed in them will determine the ideal KR and reasoning needs of the languages which will be used to specify these ontologies.

6 Multilingual Knowledge Based European Electronic Marketplace

7 The full IST -MKBEEM consortium comprises: France Telecom-R&D, SEMA Group Sae, UPM, National Technical Univ. of Athens, Univ. of Montpellier, Tradezone International Ltd, VTT, Ellos Postimyynti, SNCF, FIDAL-France. Started 1st February 2000 - Ending August 2002. See also http://www.linglink.lu/hlt/projects/mkbeem/


## ACKNOWLEDGEMENTS

This paper would not be possible without the comments and feedback of developers and users of the mentioned languages that 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] Farquhar, A., Fikes, R., Rice, J. *The Ontolingua Server: A Tool for Collaborative Ontology Construction*. Proceedings of KAW96. Banff, Canada, 1996.

[2] MacGregor, R. *Inside the LOOM clasifier*. SIGART bulletin. #2(3):70-76. June, 1991.

[3] Lenat, D.B., Guha, R.V. *Building Large Knowledge-based systems. Representation and Inference in the Cyc Project*. Addison-Wesley. *Reading*. Massachusetts. 1990.

[4] Motta, E. *Reusable Components for Knowledge Modelling*. IOS Press. Amsterdam. 1999.

[5] Kifer, M., Lausen, G., Wu, J. *Logical Foundations of Object-Oriented and Frame-Based Languages*. Journal of the ACM. 1995.

[6] Bray, T., Paoli, J., Sperberg, C. *Extensible Markup Language (XML) 1.0*. W3C Recommendation. Feb 1998. http://www.w3.org/TR/REC-xml.

[7] Lassila, O., Swick, R. *Resource Description Framework (RDF) Model and Syntax Specification*. W3C Proposed Recommendation. January, 99. http://www.w3.org/TR/PR-rdf-syntax.

[8] Luke S., Heflin J. *SHOE 1.01. Proposed Specification*. SHOE Project. February, 2000. http://www.cs.umd.edu/projects/plus/SHOE/spec1.01.htm

[9] Karp, R., Chaudhri, V., Thomere, J. *XOL: An XML-Based Ontology Exchange Language*. July, 1999.

[10] Horrocks, I., Fensel, D., Harmelen, F., Decker, S., Erdmann, M, Klein, M. *OIL in a Nutshell*. 2000.

[11] Brickley, D., Guha, R.V. *Resource Description Framework (RDF) Schema Specification*. W3C Proposed Recommendation. March, 1999. http://www.w3.org/TR/PR-rdf-schema.

[12] Thompson, H., Beech, D., Maloney, M., Mendelsohn, N. *XML Schema Part 1: Structures*. 1999. http://www.w3.org/TR/xmlschema-1/.

[13] Van Harmelen, F., Fensel, D. *Surveying notations for machine-processable semantics of Web sources*. Proceedings of the IJCAI'99 Workshop on Ontologies & PSMs. 1999.

[14] Schreiber, G., Akkermans, H., Anjewierden, A., Hoog, R., Shadbolt, N., Van de Velde, W., Wielinga, B. *Knowledge engineering and management. The CommonKADS Methodology*. MIT press, Massachussets. 1999.

[15] Gruber, R. *A translation approach to portable ontology specification*. Knowledge Acquisition. #5: 199-220. 1993.

[16] Studer, R., Benjamins, R., Fensel, D. *Knowledge Engineering: Principles and Methods*. DKE 25(1-2).pp:161-197. 1998

[17] Genesereth, M., *Fikes*, R. *Knowledge Interchange Format*. Technical Report. Computer Science Department. Stanford University. Logic-92-1. 1992.

[18] Chaudhri, V., Farquhar, A, Fikes, R., Karp, P., Rice, J. *The Generic Frame Protocol 2.0*. July, 1997.

[19] Amann, B., Fundulaki, I. Integrating Ontologies and Thesauri to Build RDF Schemas. 1999.

[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

[21] Duineveld, A., Studer, R., Weiden, M, Kenepa, B., Benjamis, R. *WonderTools? A comparative study of ontological engineering tools*. Proceedings of KAW99. Banff, Canada. 1999.