OIL in a Nutshell

D. Fensel[1], I. Horrocks[2], F. Van Harmelen[1,3], S. Decker[4], M. Erdmann[4], and M. Klein[1]

Abstract

Currently computers are changing from single isolated devices into entry points into a worldwide network of information exchange and business transactions. Support in data, information, and knowledge exchange is becoming the key issue in current computer technology. Ontologies will play a major role in supporting information exchange processes in various areas. A prerequisite for such a role is the development of a joint standard for specifying and exchanging ontologies. The purpose of the paper is precisely concerned with this necessity. We will present *OIL*, which is a proposal for such a standard. It is based on existing proposals such as OKBC, XOL and RDF, enriching them with necessary features for expressing ontologies. The paper sketches the main ideas of OIL.

1. Introduction

Currently, we are on the brink of the second Web generation. The Web started with mainly handwritten HTML pages; then the step was made to machine generated and often active HTML pages. This first generation of the Web was designed for direct human processing (reading, browsing, form-filling, etc.). The second generation Web, that we could call the "Knowledgeable Web", aims at the machine processable interpretation of information. This coincides with the vision that Tim Berners-Lee calls the Semantic Web in his recent book "Weaving the Web", and for which he uses the slogan "Bringing the Web to its full potential". The Knowledgeable Web will enable intelligent services such as information brokers, search agents, information filters etc. Ontologies will play a crucial role in enabling the processing and sharing of knowledge between programs on the Web.

Ontologies are a popular research topic in various communities, such as knowledge engineering, natural language processing, cooperative information systems, intelligent information integration, and knowledge management. They provide a shared and common understanding of a domain that can be communicated between people and application systems. They have been developed in Artificial Intelligence to facilitate knowledge sharing and reuse. Ontologies are generally defined as a "representation of a shared conceptualisation of a particular domain". Recent articles covering various aspects of ontologies can be found in [Uschold & Grüninger, 1996], [van Heijst et al., 1997], [Gomez Perez & Benjamins, 1999], [Fensel, to appear].

The IST Key Action On-To-Knowledge[1] will develop methods and tools to employ the full power of the ontological approach to facilitate Web-based knowledge use, knowledge access and knowledge management. The On-To-Knowledge tools will help knowledge workers who are not IT specialists to access company-wide information repositories in an efficient, natural and intuitive way. The technical backbone of On-To-Knowledge is the use of ontologies for the various tasks of information integration and mediation. The first major spin-off from the On-To-Knowledge project is OIL (the *Ontology Inference Layer*)[2]. OIL is a Web-based representation and inference layer for ontologies, which combines the widely used modeling primitives from frame-based languages with the formal semantics and reasoning services provided by description logics. Furthermore, OIL is the first ontology representation language that is properly grounded in W3C standards such as RDF/RDF-schema and XML/XML-schema.

It is envisaged that this core language will be extended in the future with sets of additional primitives. A more detailed discussion of OIL, including formal semantics and syntax definitions in RDF and XML, is provided in [Horrocks et al., to appear].

The content of this paper is organized as follows. Section 2 provides the underlying rationales of OIL. Section 3 provides the language primitives of OIL and discusses tool support. We also sketch possible directions in extending OIL. Section 4 compares OIL with other ontology languages and web standards. Finally, a short summary is provided in Section 5.

[1] Vrije Universiteit Amsterdam, Holland,
{dieter, frankh, mcaklein}@cs.vu.nl
[2] Department of Computer Science, University of Manchester, UK, horrocks@cs.man.ac.uk
[3] AIdministrator, Amersfoort, Nederland
4 AIFB, University of Karlsruhe,
{sde, mer}@aifb.uni-karlsruhe.de

1. www.ontoknowledge.org
2. www.ontoknowledge.org/oil

2. OIL = *O*ur *I*deas of a *L*anguage

In this Section, we will first explain the three roots upon which OIL was based. Then we will show why the existing proposal for an ontology exchange language (Ontolingua, [Gruber, 1993], [Farquhar et al., 1997]) is not very well-defined. Then the relationships of OIL with OKBC and RDF are sketched out. These are discussed further in Section 4.

2.1 The three roots of OIL

OIL unifies three important aspects provided by different communities (see Figure 1): Formal semantics and efficient reasoning support as provided by Description Logics, epistemological rich modeling primitives as provided by the Frame community, and a standard proposal for syntactical exchange notations as provided by the Web community.

Description Logics (DL). DLs describe knowledge in terms of concepts and role restrictions that are used to automatically derive classification taxonomies. The main effort of research in knowledge representation is in providing theories and systems for expressing structured knowledge, for accessing it and reasoning with it in a principled way. DLs (cf. [Brachman & Schmolze, 1985], [Baader et al., 1991]), also known as terminological logics, form an important and powerful class of logic-based knowledge representation languages.[3] They result from early work on semantic networks and define a formal and operational semantics for them. DLs try to find a fragment of first-order logic with high expressive power which still has a decidable and efficient inference procedure (cf. [Nebel, 1996]). Implemented systems include BACK, CLASSIC, CRACK, FLEX, K-REP, KL-ONE, KRIS, LOOM, and YAK.[4] A distinguishing feature of DLs is that classes (usually called concepts) can be defined intensionally in terms of descriptions that specify the properties that objects must satisfy in order to belong to the concept. These descriptions are expressed using a language that allows the construction of composite descriptions, including restrictions on the binary relationships (usually called roles) connecting objects. Various studies examine extensions of the expressive power for such languages and the trade-off in computational complexity for deriving is-a relationships between concepts in such a logic (and also, although less commonly, the complexity of deriving instance-of relationships between individuals and concepts). In spite of discouraging theoretical complexity results, there are now efficient implementations for DL languages (cf. [Borgida & Patel-Schneider, 1994], [MacGregor, 1994], [Horrocks & Patel-Schneider, 1999]), see for example DLP[5] and the FaCT system.[6] OIL inherits from Description Logic its *formal semantics* and the *efficient reasoning support* developed for these languages. In OIL, *subsumption* is decidable and with FaCT we can provide an efficient reasoner for this. In general, subsumption is only one of several reasoning tasks for working with an ontology. Others are: instance classification, query subsumption and query answering over classes and instances, navigation through ontologies, etc. However, many of them can be reformulated in terms of subsumption checking. Others may lead to different super- and subsets of the current OIL language version. The current version of OIL can be seen as a starting point for exploring the space of possible choices in designing Ontology exchange languages and characterizing them in terms of their pros and cons.

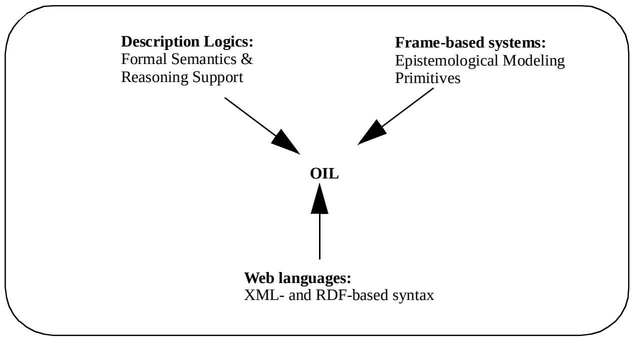

Fig 1. The three roots of OIL.

Frame-based systems. The central modeling primitive of predicate logic are predicates. Frame-based and object-oriented approaches take a different approach. Their central modeling primitive are classes (i.e., frames) with certain properties called attributes. These attributes do not have a global scope but are only applicable to the classes they are defined for (they are typed) and the "same" attribute (i.e., the same attribute name) may be associated with different range and value restrictions when defined for different classes. A frame provides a certain context for modeling one aspect of a domain. Many other additional refinements of these modeling constructs have been developed, and this has contributed to the incredible success of this modeling paradigm. Many frame-based systems and languages have been developed, and under the name object-orientation it has conquered the software engineering community. Therefore, OIL incorporates the *essential modeling primitives* of frame-based systems into its language. OIL is

3. http://dl.kr.org/. Here links to most papers, project, and research events in this area can be found.

4. http://www.research.att.com/sw/tools/classic/imp-systems.html

5. http://www.bell-labs.com/user/pfps/

6. http://www.cs.man.ac.uk/˜horrocks/software.html We will discuss later in the paper the use of FaCT as an inference engine for OIL.

based on the notion of a class and the definition of its superclasses and attributes. Relations can also be defined not as attributes of a class but as an independent entities having a certain domain and range. Like classes, relations can be arranged in a hierarchy. We will explain the difference between OIL and pure Description Logics using their different treatment of attributes. In DLs, roles are not defined for concepts. Actually, concepts are defined as subclasses of role restriction. One could rephrase this in a frame context as follows: a class is a subclass of its attribute definitions (i.e., all instances of the class must fulfil the restrictions defined for the attributes). However, asking which roles could be applied to a class does not make much sense for a DL as nearly all slots can be applied to a class. With frame-based modeling one makes the implicit assumption that only those attributes can be applied to a class that are defined for this class.

Web standards: XML and RDF. Modeling primitives and their semantics are one aspect of an Ontology Exchange Language. In addition, you have to decide about its syntax. Given the current dominance and importance of the WWW, a syntax of an ontology exchange language must be formulated using existing web standards for information representation. As already proven with XOL[7] (cf. [Karp et al., 1999], [McEntire et al., 1999]), XML can be used as a serial syntax definition language for an ontology exchange language. The BioOntology Core Group[8] recommends the use of a frame-based language with an XML syntax for the exchange of ontologies for molecular biology. The proposed language is called XOL. The ontology definitions that XOL is designed to encode include both schema information (meta-data), such as class definitions from object databases, as well as non-schema information (ground facts), such as object definitions from object databases. The syntax of XOL is based on XML, and the modeling primitives and semantics of XOL are based on OKBC-Lite. OIL is closely related to XOL and can be seen as an extension of XOL. For example, XOL allows only necessary but not sufficient class definitions (i.e., a new class is always a sub-class of and not equal to its specification) and only class names, but not class expressions (except for the limited form of expression provided by slots and their facets) can be used in defining classes. The XML syntax of OIL was mainly defined as an extension of XOL, although, as we said above for OKBC, we omit some of the original language primitives. More details on the XML syntax of OIL (defined as a DTD and in XML schema) can be found in[Horrocks et al., to appear] and [Klein et al., 2000].

Other candidates for a web-based syntax for OIL are RDF and RDFS. The Resource Description Framework (RDF)[9] (cf. [Miller, 1998], [Lassila & Swick,1999]) provides a means for adding semantics to a document without making any assumptions about the structure of the document. RDF is an infrastructure that enables the encoding, exchange and reuse of structured metadata. RDF schemes (RDFS) [Brickley & Guha, 2000] provide a basic type schema for RDF. Objects, Classes, and Properties can be described. Predefined properties can be used to model instance of and subclass of relationships as well as domain restrictions and range restrictions of attributes. A speciality of RDFS is that properties are defined globally and are not encapsulated as attributes in class definitions. Therefore, a frame or object-oriented ontology can only be expressed in RDFS by reifying the property names with class name suffixes. In regard to ontologies, RDF provides two important contributions: a standardized syntax for writing ontologies and a standard set of modeling primitives, like instance of and subclass of relationships.

2.2 Why not Ontolingua?

Ontolingua[10] (cf. [Gruber, 1993], [Farquhar et al., 1997]) is an existing proposal for a ontology exchange language. It was designed to support the design and specification of ontologies with a clear logical semantics based on KIF[11]. Ontolingua extends KIF with additional syntax to capture intuitive bundling of axioms into definitional forms with ontological significance; and a Frame Ontology to define object-oriented and frame-language terms.[12] The set of KIF expressions that Ontolingua allows is defined in an ontology, called the Frame Ontology. The Frame Ontology specifies in a declarative form the representation primitives that are often supported with special-purpose syntax and code in object-centered representation systems (e.g., classes, instances, slot constraints, etc.). Ontolingua definitions are Lisp-style forms that associate a symbol with an argument list, a documentation string, and a set of KIF sentences labeled by keywords. An Ontolingua ontology is made up of definitions of classes, relations, functions, objects distinguished, and axioms that relate these terms.

The problem with Ontolingua is its high expressive power

7. http://www.ai.sri.com/pkarp/xol/.

8. http://smi-web.stanford.edu/projects/bio-ontology/

9. http://www.w3c.org/Metadata/

10. http://ontolingua.stanford.edu/

11. The **Knowledge Interchange Format KIF** ([Genesereth, 1991], [Genesereth & Fikes, 1992]) is a language designed for use in the interchange of knowledge among disparate computer systems. KIF is based on predicate logic but provides a Lisp-oriented syntax for it.

12. The Ontolingua Server as described in [Farquhar et al., 1997] has extended the original language by providing explicit support for building ontological modules that can be assembled, extended, and refined in a new ontology.

that is provided without any means to control it. Not surprisingly, no reasoning support has ever been provided for Ontolingua. OIL takes the opposite approach. We start with a very simple and limited core language. The web has proven that restriction of initial complexity and controlled extension when required is a very successful strategy. OIL takes this lesson to heart. We already mentioned that the focus on different reasoning tasks may lead to different extensions. We also showed in [Fensel et al., 2000] serious shortcomings in the expressiveness of OIL. This process may finally lead to one version of OIL with similar expressiveness as Ontolingua. Still we would have had a process of rational reconstruction that makes certain choices with their pros and cons explicitly. Second, we would still have versions with smaller expressive power for cases they can be applied to.

In general there are two strategies to achieve a standard: Defining a “small” set of modeling primitives that are consensus in the community and define a proper semantics for them; or defining a “large” set of modeling primitives that are present in some of the approaches in a community and glue them together. Both may lead to success. The first approach can be illustrated with HTML. Its first version was very simple and limited but therefore allowed the Web to catch on and become a world wide standard. Meanwhile we have HTML version 5, XHTML, and XML. So beginning with a core set and successively refining and extending them has proven to be successful strategy. The second approach has been taken by the UML community by designing a model that is broad enough to cover all modeling concepts of a community. This leads to ambiguity and redundancy in modeling primitives and sometimes a precise semantic definition is lacking. However, UML has been adopted by Software industry as one of the major approaches, and is therefore a success too. Obviously these two opposed approaches to standardization may both work successfully. We have chosen the first approach in developing OIL. This stems from the purpose OIL is designed for. It should provide machine understandable semantics of domain theories. This will be used in the Web context to provide machine processable semantics of information sources helping the make true Tim Berners-Lee’s vision of a semantic web. Therefore clear definitions of semantics and reasoning support is essential.

2.3 OIL and OKBC

A simple and well-defined semantics is of great importance for an ontology exchange language because it is used to transfer knowledge from one context to another. There already exists an ontology exchange standard for frame-based systems, the Open Knowledge Base Connectivity (OKBC)[13] ([Chaudhri et al., 1997], [Chaudhri et al., 1998]). OKBC is an API (application program interface) for accessing frame-based knowledge representation systems. Its knowledge model supports features most commonly found in frame-based knowledge representation systems, object databases, and relational databases. OKBC-Lite extracts most of the essential features of OKBC, while not including some of its more complex aspects. OKBC has also been chosen by FIPA[14] as an exchange standard for ontologies (cf. FIPA 98 Specification, Part 12: Ontology Service [FIPA, 1998]). OIL shares many features with OKBC and defines a clear semantics and XML-oriented syntax for them. A detailed comparison is made later in this document.

2.4 OIL and RDF

In the same way that OIL provides an extension to OKBC (and is therefore downward compatible with OKBC) it also provides an extension to RDF and RDFS. Based on its RDF syntax, ontologies written in OIL are valid RDF documents. OIL extends the schema definition of RDFS by adding additional language primitives not yet present in RDFS. Based on these extensions an ontology in OIL can be expressed in RDFS.

3. The OIL Language

This section provide an informal description of the modeling primitives, an example in OIL, its tool environment, and a discussion of future extensions of OIL. The semantics of OIL is described in [Horrocks et al., to appear].

3.1 An informal description of OIL

In this section we will give an informal description of the OIL language; an example is provided in Section 3.2.

An OIL ontology is a structure made up of several components, some of which may themselves be structures, some of which are optional, and some of which may be repeated. We will write **component**$^?$ to indicate an optional component, **component**$^+$ to indicate a component that may be repeated one or more times (i.e., that must occur at least once) and **component*** to indicate a component that may be repeated zero or more times (i.e., that may be completely omitted).

When describing ontologies in OIL we have to distinguish three different layers:

- The object level where concrete instances of an ontology are described. We do *not* deal with this level in this paper. The exchange of application-specific information on instances is currently beyond the scope of OIL.

13. http://www.ai.sri.com/˜okbc/

14. http://www.fipa.org

- The first metalevel, where the actual ontological definitions are provided. Here we define the terminology that may be populated at the object level. OIL is mainly concerned with this level. It is a means for describing a structured vocabulary with well-defined semantics. The main contribution of OIL is in regard to this level.
- The second metalevel (i.e., the meta-metalevel) is concerned with describing features of such an ontology like author, name, subject, etc. For representing metadata of ontologies we make use of the DublinCore Meta data Element Set (Version 1.1) [Dublin Core] standard. The Dublin Core is a meta-data element set intended to facilitate the discovery of electronic resources. Originally conceived for author-generated descriptions of web resources, it is now widely used and has attracted the attention of resource description communities such as museums, libraries, government agencies, and commercial organizations.

OIL is concerned with the first and second metalevels. The former is called *ontology definition* and the latter is called *ontology container*. We will discuss both elements of an ontology specification in OIL. We start with the ontology container and will then discuss the backbone of OIL, the ontology definition.

Ontology Container: We adopt the components as defined by Dublin Core Meta data Element Set, Version 1.1 for the *ontology container* part of OIL. Although every element is optional and repeatable in the Dublin Core set, in OIL some elements are required or have a predefined value. Required elements are written as element$^+$. Some of the elements can be specialized with a *qualifier*, which refines the meaning of that element. In our shorthand notation we will write element.qualifier. The precise syntax based on RDF is given in [Miller et al., 1999].

Apart from various header fields encapsulated in its container, an OIL ontology consists of a **set of definitions**:

- **import**$^?$ A list of references to other OIL modules that are to be included in this ontology. XML schemas and OIL provide the same (limited) means for composing specifications. You can include specifications and the underlying assumption is that names of different specifications are different (via different prefixes).
- **rule-base**$^?$ A list of rules (sometimes called axioms or global constraints) that apply to the ontology. At present, the structure of these rules is not defined (they could be horn clauses, DL style axioms etcetera), and they have no semantic significance. The rule base consists simply of a **type** (a string) followed by the unstructured rules (a string).
- **class and slot definitions** Zero or more class definitions **(class-def**) and slot definitions **(slot-def**), the structure of which will be described below.

A class definition **(class-def**) associates a class name with a class description. A **class-def** consists of the following components:

- **type**$^?$ The type of definition. This can be either **primitive** or **defined**; if omitted, the type defaults to **primitive**. When a class is **primitive**, its definition (i.e., the combination of the following **subclass-of** and **slot-constraint** components) is taken to be a necessary but not sufficient condition for membership of the class.
- **name** The name of the class (a string).
- **documentation** Some documentation describing the class (a string).
- **subclass-of**$^?$ A list of one or more **class-expressio**ns, the structure of which will be described below. The class being defined in this **class-def** must be a sub-class of each of the class-expressions in the list.
- **slot-constraints** Zero or more **slot-constraints**, the structure of which will be described below. The class being defined in this **class-def** must be a sub-class of each of the slot-constraints in the list.

A **class-expression** can be either a class name, a **slot-constrain**t, or a boolean combination of class expressions using the operators **AND**, **OR** or **NOT**. Note that class expressions are recursively defined, so that arbitrarily complex expressions can be formed.

A **slot-constraint** (a slot may also be called a *role* or an *attribute*) is a list of one or more constraints (restrictions) applied to a **slo**t. A slot is a binary relation (i.e., its instances are pairs of individuals), but a slot-constraint is actually a class definition—its instances are those individuals that satisfy the constraint(s). A **slot-constraint** consists of the following components:

- **name** A slot name (a string). The slot is a binary relation that may or may not be defined in the ontology. If it is not defined it is assumed to be a binary relation with no globally applicable constraints, i.e., any pair of individuals could be an instance of the slot.
- **has-value**$^?$ A list of one or more **class-expressions**. Every instance of the class defined by the slot-constraint must be related via the slot relation to an instance of each **class-expression** in the list. For example, the **value** constraint:

  slot-constraint *eats*
    has-value zebra, wildebeest

  defines the class each instance of which *eats* some instance of the class zebra and some instance of the class wildebeest. Note that this does not mean that instances of the slot-constraint eat *only* zebra and wildebeest: they may also be partial to a little gazelle when they can get it. **Has-value** *expresses the existential quantifier of Predicate logic and a necessary condition*. An instance of a class must have at most one value for this slot that fulfils its range restriction.

- **value-type?** A list of one or more **class-expressions**. If an instance of the class defined by the slot-constraint is related via the slot relation to some individual x, then x must be an instance of each **class-expression** in the list. For example, the **value-type** constraint:

  slot-constraint *eats*
    value-type meat

  defines the class each instance of which *eats* nothing that is not meat. Note that this does not mean that instances of the slot-constraint eat anything at all. **value-type** *expresses the all quantifier of Predicate logic and a sufficient condition*. If an instance of a class has a value for this slot, then it must fulfil its range restriction.
- **max-cardinality?** A non-negative integer *n* followed by a **class-expression**. An instance of the class defined by the slot-constraint can be related to at most *n* distinct instances of the **class-expression** via the slot relation.
- **min-cardinality?** A non-negative integer *n* followed by a **class-expression**. An instance of the class defined by the slot-constraint must be related to at least *n* distinct instances of the **class-expression** via the slot relation.

A slot definition **(slot-def**) associates a slot name with a slot description. A slot description specifies global constraints that apply to the slot relation, for example that it is a transitive relation. A **slot-def** consists of the following components:

- **name** The name of the slot (a string).
- **documentation?** Some documentation describing the slot (a string).
- **subslot-of?** A list of one or more **slo**ts. The slot being defined in this **slot-def** must be a sub-slot of each of the slots in the list. For example,

  slot-def *daughter*
    subslot-of *child*

  defines a slot *daughter* that is a subslot of *child*, i.e., every pair of individuals that is an instance of *daughter* must also be an instance of *child*.
- **domain?** A list of one or more **class-expressions**. If the pair (*x*,*y*) is an instance of the slot relation, then *x* must be an instance of each **class-expression** in the list.
- **range?** A list of one or more **class-expressions**. If the pair (*x*,*y*) is an instance of the slot relation, then *y* must be an instance of each **class-expression** in the list.
- **inverse?** The name of a slot *S* that is the inverse of the slot being defined. If the pair (*x*,*y*) is an instance of the slot S, then (*y*,*x*) must be an instance of the slot being defined. For example,

  slot-def *eats*
    inverse *eaten-by*

  defines the inverse of the slot *eats* to be the slot *eaten-b*y, i.e., if *x eats y* then *y* is *eaten-by x*.
- **properties?** A list of one or more properties of the slot. Valid properties are: **transitive** and **symmetric**.

3.2. An example OIL ontology

The following example of an OIL ontology illustrates some of the key features of the language.[15]

```
ontology-container
   title “African animals”
   creator “Ian Horrocks”
   subject “animal, food, vegetarians”
   description “A didactic example ontology describing
   African animals”
   description.release “1.01”
   publisher “I. Horrocks”
   type “ontology”
   format “pseudo-xml”
   format “pdf”
   identifier “http://www.cs.vu.nl/~dieter/oil/TR/oil.pdf”
   source “http://www.africa.com/nature/animals.html”
   language “OIL”
   language “en-uk”
   relation.hasPart “http://www.ontosRus.com/animals
   /jungle.onto”

ontology-definitions
  slot-def eats
     inverse is-eaten-by
  slot-def has-part
     inverse is-part-of
     properties transitive
  class-def animal
  class-def plant
     subclass-of NOT animal
  class-def tree
     subclass-of plant
  class-def branch
     slot-constraint is-part-of
        has-value tree
```

15. For reasons of space limitations only parts of the language are illustrated.

```
class-def leaf
  slot-constraint is-part-of
    has-value branch
class-def defined carnivore
  subclass-of animal
  slot-constraint eats
    value-type animal
class-def defined herbivore
  subclass-of animal
  slot-constraint eats
    value-type
      plant OR
      (slot-constraint is-part-of has-value plant)
class-def giraffe
  subclass-of animal
  slot-constraint eats
    value-type leaf
class-def lion
  subclass-of animal
  slot-constraint eats
    value-type herbivore
class-def tasty-plant
  subclass-of plant
  slot-constraint eaten-by
    has-value herbivore, carnivore
```

Some points to note in the above ontology are:

- The classes plant and animal are made disjoint by defining plant to be a subclass of **NOT** animal.
- The class carnivore is a defined class, and lion can be recognized as a sub-class of carnivore because of its definition.
- The class herbivore is a defined class, and giraffe can be recognized as a sub-class of herbivore because of its definition. However, in this case the inference is a little more complex and is only valid because *has-part* is transitive and *is-part-of* is the inverse of *has-part*.
- The class tasty-plant is inconsistent. This is because tasty-plant is a kind of plant that is eaten by both herbivores and carnivores, but we have already stated that carnivore eat only animals, and that animal and plant are disjoint.

3.3. Current Limitations of OIL

Our starting point has been to define a decidable core language, with the intention that additional (and possibly important) features be defined as a set of extensions (still with clearly defined semantics). Modelers will be free to use these language extensions, but it will be clear that this may compromise decidability and reasoning support. This seems to us a cleaner solution than trying to define a single “all things to all men” language.

In this section we briefly discuss a number of features which are available in other ontology modeling languages and which are not or not yet included in OIL. For each of these features, we briefly motivate our choice, and mention future prospects where relevant.

Default reasoning: Although OIL does provide a mechanism for inheriting values from super-classes, such values cannot be overwritten. As a result, such values cannot be used for the purpose of modeling default values. Combining defaults with a well defined semantics and reasoning support is known to be problematical.

Rules/Axioms: As discussed above, only a fixed number of algebraic properties of slots can be expressed in OIL. There is no facility for describing arbitrary axioms that must hold for all items in the ontology. Such a powerful feature is undoubtedly useful and may be added to the core language.

Modules: We presented a very simple construction to modularize ontologies in OIL. In fact, this mechanism is identical to the namespace mechanism in XML. It amounts to a textual inclusion of the imported module, where name-clashes are avoided by prefixing every imported symbol with a unique prefix indicating its original location. Future extensions would concern parameterized modules, signature mappings between modules, and restricted export interfaces for modules.

Using instances in class definitions: Results from research in description and modal logics show that the computational complexity of such logics changes dramatically for the worse when reasoning with domain-instances is allowed (cf. [Areces et al., 1999]). For this reason OIL does not currently allow the use of instances in slot-values, or extensional definitions of classes (i.e., class definitions by enumerating the class instances).

Concrete domains: OIL currently does not support concrete domains (e.g., integers, strings, etc.). This would seem to be a serious limitation for a realistic ontology exchange language, and extensions of OIL in this direction are probably necessary. The theory of concrete domains is well understood [Baader & Hanschke, 1991], and it should be possible to add some restricted form of concrete domains without sacrificing reasoning support.

Limited Second-order expressivity: Many existing languages for ontologies (KIF, CycL, Ontolingua) include some form of reification mechanism in the language, which allows us to treat statements of the language as objects in their own right, thereby making it possible to express statements about these statements. A full second order extension would be clearly undesirable (even unification is undecidable in full 2nd order logic). However, much weaker second order constructions already provide much if not all of the required expressivity without causing any computational problem (in effect, they are simply 2nd order syntactic sugar for what are essentially first order constructions).

3.4 Tools

OIL makes use of the **FaCT (Fast Classification of Terminologies)** system in order to provide reasoning support for ontology design, integration and verification. FaCT is a Description Logic classifier that can also be used for consistency checking in modal and other similar logics. FaCT's most interesting features are its expressive logic (in particular the *SHIQ* reasoner), its optimized tableaux implementation (which has now become the standard for DL systems), and its CORBA based client-server architecture. FaCT's optimizations are specifically aimed at improving the system's performance when classifying realistic ontologies, and this results in performance improvements of several orders of magnitude when compared with older DL systems. This performance improvement is often so great that it is impossible to measure precisely as unoptimised systems are virtually non-terminating with ontologies that FaCT is easily able to deal with [Horrocks & Patel-Schneider, 1999]. Taking a large medical terminology ontology developed in the GALEN project [Rector et al., 1993] as an example, FaCT is able to check the consistency of all 2,740 classes and determine the complete class hierarchy in about 60 seconds of (450MHz Pentium III) CPU time.[16] In contrast, the KRIS system [Baader & Hollunder, 1991] had been unable to complete the same task after several weeks of CPU time.

4. Comparing OIL with other approaches

This section compares OIL with other frame-based approaches and with the arising web standards RDF and RDFS.

4.1 OIL and other frame-oriented approaches

The modeling primitives of OIL are based on those of XOL (cf. [Karp et al., 1999]). OIL extends XOL so as to make it more suitable for capturing ontologies defined using a logic-based approach (such as used in DLs) in addition to the frame-based ontologies for which XOL (and OKBC [Chaudhri et al., 1998]) were designed. The extensions are designed so that most valid XOL ontologies should also be valid OIL ontologies. The exceptions are due to the omission of constructs for which reasoning support (e.g., for class consistency and subsumption checking) could not be provided from OIL, either because their semantics are unclear or because their inclusion would lead to the language being undecidable.

How OIL extends XOL

It is the frame structure itself that restricts the way language primitives can be combined to define a class. In XOL, class definitions consist of the specification of zero or more parent classes (from which characteristics are inherited) and zero or more slots—binary relations whose characteristics can be additionally restricted using slot *facets* (e.g., the range of the relation can be restricted using the **value-type** facet). Viewed from a logical perspective, each slot (with its associated facets) defines a class (e.g., a slot *eats* with the **value-type** junk-food defines the class of individuals who eat nothing but junk food), and the frame is implicitly[17] the class formed from the conjunction of all the slots and all the parent classes. Consequently, every class must be defined by a conjunction of slots (which themselves have a very restricted form) and other named classes. In contrast, DLs usually allow language primitives to be combined in arbitrary boolean expressions (i.e., using conjunction, disjunction and negation) and allow class definitions to be used recursively wherever a class name might appear. Moreover, XOL only provides one form of class definition statement. It is not clear whether the resulting class is meant to be primitive or non-primitive: we will assume that it is primitive.[18]

In our view, this very restricted form of class definition makes XOL (and indeed OKBC) unsuitable as an ontology exchange language: it makes it impossible to capture even quite basic DL ontologies and precludes some very simple and intuitive kinds of class definition. For example, it is impossible to define the class of vegetarian as the subclass of person such that everything they eat is neither meat nor fish. On the one hand, the value of the **value-type** facet of the slot *eats* cannot be an expression such as **"not** (meat **or** fish)". On the other hand, because vegetarian must be primitive, there could be individuals of type person who eat neither meat nor fish but who are not classified as vegetarians.[19] Another serious weakness of XOL class definitions (and those of OKBC) is that there is no mechanism for specifying disjointness of classes, a basic modeling primitive that can be captured even by many conceptual modeling formalisms used for database schema design.[20] This makes it impossible to capture the fact that

[16] Adding single classes and checking both their consistency and their position in the class hierarchy is virtually instantaneous.

[17] The OKBC semantics (on which XOL relies) are less than clear on this and on several other important points.

[18] In contrast, OKBC supports the definition of both primitive and non-primitive classes.

[19] This aspect of the definition can be captured in OKBC as non-primitive classes are supported.

the class male is disjoint from the class female. This is easy for a DL, where the class female can simply be made a subclass of **"not** male".

Another weakness of XOL (and OKBC) is that slots (relations) are very much second class citizens when compared to classes. In particular, there is no support for a slot hierarchy and only restricted kinds of properties that can be specified for relations. For example, it is not possible to define the slot *has-parent* as a subslot of the *has-ancesto*r, nor is it possible to specify that *has-ancestor* is a transitive relation. The specification of this kind of slot hierarchy including transitive and non-transitive relations is essential in ontologies dealing with complex physically composed domains such as human anatomy [Rector et al., 1997] and engineering [Sattler, 1995].

How OIL restricts XOL

As mentioned above, OIL also restricts XOL in some respects.

- Initially, only conceptual modeling will be supported, i.e., individuals are not supported. This does not seem too onerous a restriction for an ontology specification language given that an ontology can be viewed as a kind of schema. Moreover, allowing individuals to occur in class definitions is equivalent to having extensionally defined classes, and this soon leads to very hard reasoning problems and even undecidability (cf. [Areces et al., 1999]). This means that slot values in OIL can only be classes. Future extensions of OIL may support the specification of individuals as instances of one or more classes.
- The slot constraints **numeric-minimum** and **numeric-maximum** are not supported. Again, future extensions of OIL may support concrete data types (including numbers and numeric ranges).
- Collection types other than **set** are not supported.
- Slot **inverse** can only be specified in global slot definitions: naming the inverse of a relation only seems to make sense when applied globally.

4.2 OIL and RDF

The **Resource Description Framework (RDF)** [Lassila & Swick,1999] is a recommendation of the World Wide Web Consortium (W3C) for representing meta-data in the Web. RDF data represents resources and attached attribute/value pairs. A resource represent anything representable through a URI. Attributes are named properties of the resources, and their values are either atomic entities (text strings, numbers, etc.) or other resources represented by a URI. The resources, properties, and values build up the RDF data model, that can be seen as a labeled directed graphs.

Besides defining the data model, RDF needs a serialization syntax to make actual data available in the Web. For this purpose XML was chosen. RDF and XML are complementary in so far as RDF represents the abstract model and XML provides the concrete textual representation of the model. There are several ways to represent the same RDF data model in XML.

A third component in the RDF-context has to be introduced: since RDF does not define any particular vocabularies for authoring of data, a schema language with appropriate primitives is needed. For this purpose the RDF-Schema specification was created. RDF-schema is a simple ontology language able to define basic vocabularies which covers the simplest parts a of a knowledge model like OKBC (classes, properties, domain and range restrictions, instance-of, subclass-of and subproperty-of relationships). RDF-Schema is itself defined in RDF and an RDF Schema defining the RDF-Schema language itself is also available [Brickley & Guha, 2000].

The relationship between OIL and RDF/RDFS is very close because RDF/RDFS was meant to capture meaning in the manner of semantic nets. In the same way, as RDF-Schema is used to define itself it can be used to define other ontology languages. We define a syntax for OIL by giving an RDF-Schema for the core of OIL and proposing related RDF-Schemas that could complement this core to cover further aspects. To ensure maximal compatibility with existing RDF/RDFS-applications and vocabularies the integration of OIL with the resources defined in RDF-Schema has been a main focus in designing the RDF-model for OIL (see for a survey).

- The major integration point of RDF/RDFS and OIL is defined by the class oil:ClassExpression, which is a subclass of rdfs:Resource (the most general class in RDFS). In OIL, rdfs:Class is also made a subclass of oil:ClassExpression. Doing this, existing classes from RDFS-vocabularies can be accessed and refined in OIL descriptions. Other subclasses of oil:ClassExpression are the boolean operators oil:AND, oil:OR and oil:NOT and oil:SlotConstraint (which is also a subclass of rdfs:ConstraintResource) with its derivatives. The class oil:ClassExpression thus embraces all ways to define class descriptions in OIL.
- Furthermore, OIL-slots are realized as instances of rdf:Property or of subproperties of the original rdf:Property. The subslot relationship is also expressed by original RDF-means, namely the rdfs:subPropertyOf relationship. rdf:Property is enriched by a qualifier oil:inverseRelationOf, that specifies the inverse role. Properties can also be realized as instances of oil:TransitiveRelation and oil:SymmetricRelation to express this qualities of

[20] For example extended entity relationship (EER) modeling.

properties, which are not available in RDF/RDFS.

- To distinguish between primitive and defined class definitions, two new class are introduced: oil:PrimitiveClass and oil:DefinedClass. Every class definition can be realized as an instance of one of these classes, and thus specified as a primitive or defined class.
- To express constraints on the way properties may be applied to classes, we introduced oil:SlotConstraint, which encapsulates the constraints on the slots of a class. This goes beyond the possibilities in the original RDFS-specification to define classes.

In a nutshell, RDFS relies on RDF and defines a new name space called RDFS. Some of the OIL primitives can directly be expressed in this name space. Others require a refinement of the RDFS primitives in an additional OIL name space.

5. Summary

In this paper, we sketched out both the syntax and semantics of an ontology exchange language called OIL. One of our main motivations while defining this language has been to ensure that it has a clear and well defined semantics—an agreed common syntax is useless without an agreement as to what it all means.

The core we have currently defined can be justified from a pragmatic and a theoretical point of view. From a pragmatic point of view, OIL covers consensual modeling primitives of Frame systems and Description Logics. From a theoretical point of view it seems quite natural to us to limit the expressiveness of this version so that subsumption is decidable. This defines a well-understood subfragment of first-order logic. However, it is important to note that we are open to further discussions that may influence the final design of an ontology exchange language.

We are currently evaluating the use of OIL in the two running IST projects: On-to-knowledge[21] and Ibrow[22]. In On-to-knowledge, OIL will be extended to become a full-fledged environment for knowledge management in large intranets. Unstructured and semi-structured data will be annotated automatically and agent-based user interface techniques and visualization tools will help users to navigate and query the information space. Here On-to-knowledge continues a line of research that was set up with SHOE (cf. [Luke et al., 1996], [Heflin et al., 1999]) and Ontobroker (cf. [Fensel et al., 1998a], [Fensel et al., 1999]): using ontologies to model and annotate the semantics of information in a machine processable manner. In Ibrow, we are currently investigating the usefulness of OIL for software component description, based on its integration with UPML (cf. [Fensel et al., 2000]).

21. *On-To-Knowledge: Content-driven Knowledge-Management Tools through Evolving Ontologies*. http://www.ontoknowledge.com

22. *IBROW* started with a preliminary phase under the 4th European Framework and has been a full-fledged Information Society Technologies (IST) project under the 5th European Framework Program since February 2000. http://www.swi.psy.uva.nl/projects/ibrow/home.html

Acknowledgment. We thank J. Broekstra, Monica Crubezy, W. Grosso, Carole Goble, Peter Karp, Robin McEntire, Enrico Motta, Mark Musen, Steffen Staab, Guus Schreiber, Rudi Studer, and the anonymous reviewers for helpful comments on drafts of this paper, and Jeff Butler for correcting our English.

References

[Areces et al., 1999]C. Areces, P. Blackburn, and M. Marx: A road-map on complexity for hybrid logics. In *Proc. of CSL'99*, number 1683 in LNCS, pages 307–321. Springer-Verlag, 1999.

[Baader et al., 1991] F. Baader, H.-J. Heinsohn, B. Hollunder, J. Muller, B. Nebel, W. Nutt, and H.-J. Profitlich: Terminological knowledge representation: A proposal for a terminological logic. Technical Memo TM-90-04, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH (DFKI), 1991.

[Baader & Hanschke, 1991] F. Baader and P. Hanschke: A Scheme for Integrating Concrete Domains into Concept Languages. In *Proceddings IJCAI91*, 1991: 452–457.

[Baader & Hollunder, 1991] F. Baader and B. Hollunder: KRIS: Knowledge representation and inference system. *SIGART Bulletin*, 2(3):8–14, 1991.

[Bechhofer et al., 1999] S. Bechhofer, I. Horrocks, P. F. Patel-Schneider, and S. Tessaris: A proposal for a description logic interface. In *Proc. of DL'99*, pages 33–36, 1999.

[Borgida & Patel-Schneider, 1994] A. Borgida and P. F. Patel-Schneider: A semantics and complete algorithm for subsumption in the CLASSIC description logic. *J. of Artificial Intelligence Research*, 1:277–308, 1994.

[Brachman & Schmolze, 1985] R. J. Brachman and J. G. Schmolze: An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.

[Brickley & Guha, 2000] D. Brickley and R.V. Guha: Resource Description Framework (RDF) Schema Specification 1.0, W3C Candidate Recommendation 27 March 2000. http://www.w3.org/TR/2000/CR-rdf-schema-20000327.

[Chaudhri et al., 1997] V. K. Chaudhri, A. Farquhar, R.

Fikes, P. D. Karp, and J. P. Rice: Open knowledge base connectivity 2.0. Technical Report KSL-98-06, Knowledge Systems Laboratory, Stanford, 1997.

[Chaudhri et al., 1998] V. K. Chaudhri, A. Farquhar, R. Fikes, P. D. Karp, and J. P. Rice: OKBC: A programmatic foundation for knowledge base interoperability. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98) and of the 10th Conference on Innovative Applications of Artificial Intelligence (IAAI-98*), pages 600–607. AAAI Press, 1998.

[Dublin Core] http://purl.oclc.org/dc/

[Errikson et al., 1999] H. Eriksson, R. W. Fergerson, Y. Shahar, and M. A. Musen: Automated Generation of Ontology Editors. In *Proceedings of the Twelfth Workshop on Knowledge Acquisition, Modeling and Management (KAW99)*, Banff, Alberta, Canada, October 16-21, 1999.

[Farquhar et al., 1997] A. Farquhar, R. Fikes, and J. Rice: The ontolingua server: A tool for collaborative ontology construction. *Journal of Human-Computer Studie*s, 46:707–728, 1997.

[Fensel, to appear] D. Fensel. *Ontologies: Silver Bullet for Knowledge Management and Electronic Commerc*e. Springer-Verlag, to appear.

[Fensel et al., 1998a] D. Fensel, S. Decker, M. Erdmann und R. Studer: Ontobroker: The Very High Idea. In *Proceedings of the 11th International Flairs Conference (FLAIRS-98)*, Sanibal Island, Florida, USA, 131-135, Mai 1998.

[Fensel et al., 1998b] D. Fensel, J. Angele, and R. Studer: The Knowledge Acquisition And Representation Language KARL, *IEEE Transactions on Knowledge and Data Engineering*, 10(4):527-550, 1998.

[Fensel et al., 1999] D. Fensel, J. Angele, S. Decker, M. Erdmann, H.-P. Schnurr, S. Staab, R. Studer, and A. Witt: On2broker: Semantic-Based Access to Information Sources at the WWW. In *Proceedings of the World Conference on the WWW and Internet (WebNet 99),* Honolulu, Hawaii, USA, October 25-30, 1999.

[Fensel et al., 2000] D. Fensel, M. Crubezy, F. van Harmelen, and M. I. Horrocks: OIL & UPML: A Unifying Framework for the Knowledge Web. In *Proceedings of the Workshop on Applications of Ontologies and Problem-solving Methods, 14th European Conference on Artificial Intelligence ECAI'00*, Berlin, Germany August 20-25, 2000.

[FIPA, 1998] Foundation for Intelligent Physical Agents (FIPA): *FIPA 98 Specificatio*n, 1998.

[Genesereth, 1991] M. R. Genesereth: Knowledge interchange format. In J. Allen, R. Fikes, and E. Sandewall, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Second International Conference (KR'91*). Morgan Kaufmann Publishers, San Francisco, California, 1991.

[Genesereth & Fikes, 1992] M.R. Genesereth and R.E. Fikes: Knowledge interchange format, version 3.0, reference manual. Technical Report Logic-92-1, Computer Science Dept., Stanford University, 1992.

[Gomez Perez & Benjamins, 1999] A. Gomez Perez and V. R. Benjamins: Applications of ontologies and problem-solving methods. *AI-Magazin*e, 20(1):119–122, 1999.

[Grosso et al., 1999] W. E. Grosso, H. Eriksson, R. W. Fergerson, H. Gennari, S. W. Tu, and M. A. Musen: Knowledge Modeling at the Millennium (The Design and Evolution of Protégé-2000). In *Proceedings of the Twelfth Workshop on Knowledge Acquisition, Modeling and Management (KAW99)*, Banff, Alberta, Canada, October 16-21, 1999.

[Gruber, 1993] T. R. Gruber: A translation approach to portable ontology specifications. *Knowledge Acquisitio*n, 5(2), 1993.

[Guenther, 1999] R. Guenther: Type Working Group List of Resource Types 1999-08-05. http://purl.org/DC/ documents/wd-typelist.htm

[van Heijst et al., 1997] G. van Heijst, A. Th. Schreiber, and B. J. Wielinga: Using explicit ontologies in KBS development. *International Journal of Human-Computer Studie*s, 46(2/3):183–292, 1997.

[Heflin et al., 1999] J. Heflin, J. Hendler, and S. Luke: SHOE: A Knowledge Representation Language for Internet Applications. Technical Report, CS-TR-4078 (UMIACS TR-99-71), Dept. of Computer Science, University of Maryland at College Park. 1999.

[Horrocks et al., to appear] I. Horrocks, D. Fensel, J. Broekstra, S. Decker, M. Erdmann, C. Goble, F. Van Harmelen, M. Klein, S. Staab, and R. Studer: The Ontology Inference Layer OIL. http:// www.ontoknowledge.org/oil.

[Horrocks & Patel-Schneider, 1999] I. Horrocks and P. F. Patel-Schneider: Optimising description logic subsumption. *Journal of Logic and Computatio*n, 9(3):267–293, 1999.

[Karp et al., 1999]P. D. Karp, V. K. Chaudhri, and J. Thomere: XOL: An XML-based ontology exchange language. Version 0.3, 1999.

[Klein et al., 2000] M. Klein, D. Fensel, F. van Harmelen, and I. Horrocks: The Relation between Ontologies and Schema-Languages: Translating OIL-Specifications to XML-Schema. In *Proceedings of the Workshop on Applications of Ontologies and*

Problem-solving Methods, 14th European Conference on Artificial Intelligence ECAI'00, Berlin, Germany August 20-25, 2000.

[Lassila & Swick,1999] O. Lassila and R. Swick: Resource description framework (RDF). W3C recommendation. http://www.w3c.org/TR/WD-rdf-syntax, 1999.

[Luke et al., 1996] S. Luke, L. Spector, and D. Rager: Ontology-Based Knowledge Discovery on the World-Wide Web. In *Working Notes of the Workshop on Internet-Based Information Systems at the 13th National Conference on Artificial Intelligence (AAAI96)*, 1996.

[McEntire et al., 1999] R. McEntire, P. Karp, N. Abernethy, F. Olken, R. E. Kent, M. DeJongh, P. Tarczy-Hornoch, D. Benton, D. Pathak, G. Helt, S. Lewis, A. Kosky, E. Neumann, D. Hodnett, L. Tolda, and T. Topaloglou: An evaluation of ontology exchange languages for bioinformatics, 1999.

[MacGregor, 1994] R. M. MacGregor: A description classifier for the predicate calculus. In *Proceedings of the Twelfth National Conference on Artificial Intelligenc*e, pages 213–220, Seattle, Washington, USA, 1994.

[Miller, 1998] E. Miller: An introduction to the resource description framework. *D-Lib Magazin*e, 1998.

[Miller et al., 1999] E. Miller, P. Miller, and D. Brickley: Guidance on expressing the Dublin Core within the Resource Description Framework (RDF). http://www.ukoln.ac.uk/metadata/resources/dc/datamodel/WD-dc-rdf.

[Nebel, 1996] B. Nebel: Artificial intelligence: A computational perspective. In G. Brewka, editor, *Principles of Knowledge Representatio*n, Studies in Logic, Language and Information. CSLI publications, Stanford, 1996.

[Patel-Schneider & Swartout, 1993]P. F. Patel-Schneider and B. Swartout: Description logic specification from the KRSS effort, 1993.

[Rector et al., 1993] A. L. Rector, W A Nowlan, and A Glowinski: Goals for concept representation in the GALEN project. In *Proceedings of the 17th Annual Symposium on Computer Applications in Medical Care (SCAMC'93)*, pages 414–418, Washington DC, USA, 1993.

[Rector et al., 1997] A. Rector, S. Bechhofer, C. A. Goble, I. Horrocks, W. A. Nowlan, and W. D. Solomon: The GRAIL concept modelling language for medical terminology. *Artificial Intelligence in Medicin*e, 9:139–171, 1997.

[Sattler, 1995] U. Sattler: A concept language for engineering applications with part–whole relations. In *Proceedings of the International Conference on Description Logics—DL'9*5, pages 119–123, Roma, Italy, 1995.

[Uschold & Grüninger, 1996] M. Uschold and M. Grüninger: Ontologies: Principles, methods and applications. *Knowledge Engineering Revie*w, 11(2), 1996.