12. August 2000

The Ontology Inference Layer OIL

I. Horrocks[1], D. Fensel[2], J. Broekstra[3], S. Decker[4], M. Erdmann[4], C. Goble[1], F. van Harmelen[2,3], M. Klein[2], S. Staab[4], R. Studer[4], and E. Motta[5]

[1] Department of Computer Science
University of Manchester, UK
{horrocks, carole}@cs.man.ac.uk

[2] Vrije Universiteit Amsterdam, Holland
{dieter, frankh, mcaklein}@cs.vu.nl

[3] AIdministrator Nederland B.V.
jeen.broekstra@aidministrator.nl
frank.van.harmelen@aidministrator.nl

[4] AIFB, University of Karlsruhe, D-76128, Germany
{sde, mer, sst, rst}@aifb.uni-karlsruhe.de

[5] Knowledge Media Institute, The Open University, UK
e.motta@open.ac.uk

Homepage: http://www.ontoknowledge.org/oil

Abstract Currently computers are changing from single isolated devices to entry points into a worldwide network of information exchange and business transactions. Therefore, support in the exchange of data, information, and knowledge is becoming the key issue in computer technology today. *Ontologies* provide a shared and common understanding of a domain that can be communicated between people and across application systems. 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. This paper deals with precisely 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, and enriches them with necessary features for expressing rich ontologies. The paper presents the motivation, underlying rationale, modeling primitives, syntax, semantics, and tool environment of OIL. With OIL, we want to make a proposal that initiates a discussion leading to a useful and well defined consensus amongst a large community which could use such an approach.

1 Introduction and Motivation

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 across application systems. They have been developed in Artificial Intelligence to facilitate knowledge sharing and reuse. Recent articles covering various aspects of ontologies can be found in [Uschold & Grüninger, 1996], [van Heijst et al., 1997], [Studer et al., 1998], [Benjamins et al., 1999 (a)], [Gomez Perez & Benjamins, 1999], [Fensel, 2000]. An ontology provides an explicit conceptualization (i.e., meta information) that describes the semantics of the data. They have a function similar to a database schema. Some differences are:[1]

- A language for defining ontologies is syntactically and semantically richer than common approaches for databases.
- An ontology must be a shared and consensual terminology because it is used for information sharing and exchange.
- An ontology provides a domain theory and not the structure of a data container.

Currently computers are shifting from being single isolated devices to becoming entry points into a worldwide network of information exchange and business transactions. Given the exponential growth of on-line information available, an automatic processing of this information becomes necessary for keeping things maintainable and accessible. Automatic processing of information requires a machine-understandable representation of its semantics. Providing shared and common domain structures becomes essential and ontologies will therefore become a key asset in information exchange -- being used to describe the structure and semantics of information exchange. Such technologies will play a key role in areas such as knowledge management and electronic commerce, which are market areas with incredible growth potential.

In the area of information systems and intelligent information integration, we can distinguish different integration tasks that have to be solved in order to achieve a completely integrated access to information [Stuckenschmidt, submitted]:

- **Technical Integration:** This task is concerned with network technology and communication protocols, ensuring that the different information sources can communicate at the physical level. In the last decade, the Internet and the World Wide Web have established a stable infrastructure for exchanging large amounts of information from all over the world. A widely shared and stable set of protocols (TCP/IP, HTTP, FTP etc.) now make it possible that information from web-pages, web-connected data-bases, and web-enabled programs can, in principle, be readily accessed.
- **Syntactic Integration:** Once information sources can exchange information “in principle”, they must agree on a common syntax for exchanging such information. In the WWW, HTML has been serving this purpose rather well up to now. XML is rapidly gaining importance in this area.
- **Semantic integration:** A problem that goes beyond syntactic integration is the mapping of the semantics of terms from different information sources, even when these terms have been expressed using the same syntactic structures: e.g. even when two applications use XML as their interchange format, how can we be sure that they use the same vocabulary, and that the words in

1. See [Klein et al., 2000], for an elaborated comparison of database schemes and ontologies.

this vocabulary mean the same thing?[2]

As we already pointed out, ontologies are good candidates for providing the shared and common domain structures which are required for a truly semantic integration of information sources. The question then becomes: *how will we describe such ontologies?* A prerequisite for such a widespread use of ontologies for information integration and exchange is the achievement of a joint standard for describing ontologies. Take the area of databases as an example. The huge success of the relational model would never have been possible without the SQL standard that provided an implementation-independent way for storing and accessing data. Any approach that tries to achieve such a standard for the areas of ontologies has to answer these questions: What are the appropriate modeling primitives for representing ontologies? How can we define their semantics? and What is the appropriate syntax for representing ontologies? In the US, research funding agencies have already recognized the importance of such issues in setting up the DAML program[3], which aims at a machine processable semantics of information sources which are accessible to agents.

In this paper we present a proposal for such a standard way of expressing ontologies based on using new web standards like XML schemas and RDF schemas: the *Ontology Inference Layer*. It is important to note that we intend OIL to be extensible, and that only the *core* language is described in this paper. This language has been designed so that:

- It provides most of the modeling primitives commonly used in frame-based Ontologies.
- It has a simple, clean, and well defined semantics based on Description Logic.
- Automated reasoning support (e.g., class consistency and subsumption checking) can be provided.

It is envisaged that this core language will be extended in the future with sets of additional primitives covering areas such as concrete data types (e.g., numbers and strings) and extensional class definitions, with the proviso that full reasoning support may not be available for ontologies using such primitives. A further level of extension could include modelling primitives such as defaults, fuzzy/ probabilistic definitions, additional collection types (e.g., bags and lists), and a more expressive axiomatic language. In a nutshell, we do not want to present the final version of OIL in this paper. We want to make a proposal opening the discussion process that may finally lead to a useful and well defined consensus amongst a large community making use of such an approach.

This paper is organized as follows. Section 2 provides the general background for the discussion on OIL (i.e., our position). Section 3 provides the language primitives of OIL and discusses technical support for OIL. We also sketch possible directions for extending OIL. Section 4 compares OIL with other ontology languages and web standards such as XML and RDF. Finally, a short summary is provided in Section 5. The appendix provides syntax definitions of OIL in XML and RDF plus a formal semantics of OIL.

---

2. More precisely, ontologies are used for the semantic integration of information sources. The language we provide for ontology interchange is not for semantic integration of ontologies but for ontology interchange via reuse (i.e., reusing an ontology written in another language). We do not deal with the integration of heterogeneous ontologies in this paper.

3. http://www.darpa.mil/iso/ABC/BAA0007PIP.htm.

2 The Three Roots of OIL

In this Section, we will first explain the three roots upon which OIL is based. Then we will show why we believe that the existing proposal for an ontology interchange language (Ontolingua, [Gruber, 1993], [Farquhar et al., 1997]) is not appropriate as a standard ontology language for the web. Then the relationships of OIL with OKBC and RDF are sketched out. These will be 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, epistemologically 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 the research in knowledge representation is in providing theories and systems for expressing structured knowledge and for accessing 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.[1] They result from early work on semantic networks, and defined a formal semantics for them. DLs attempt 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,

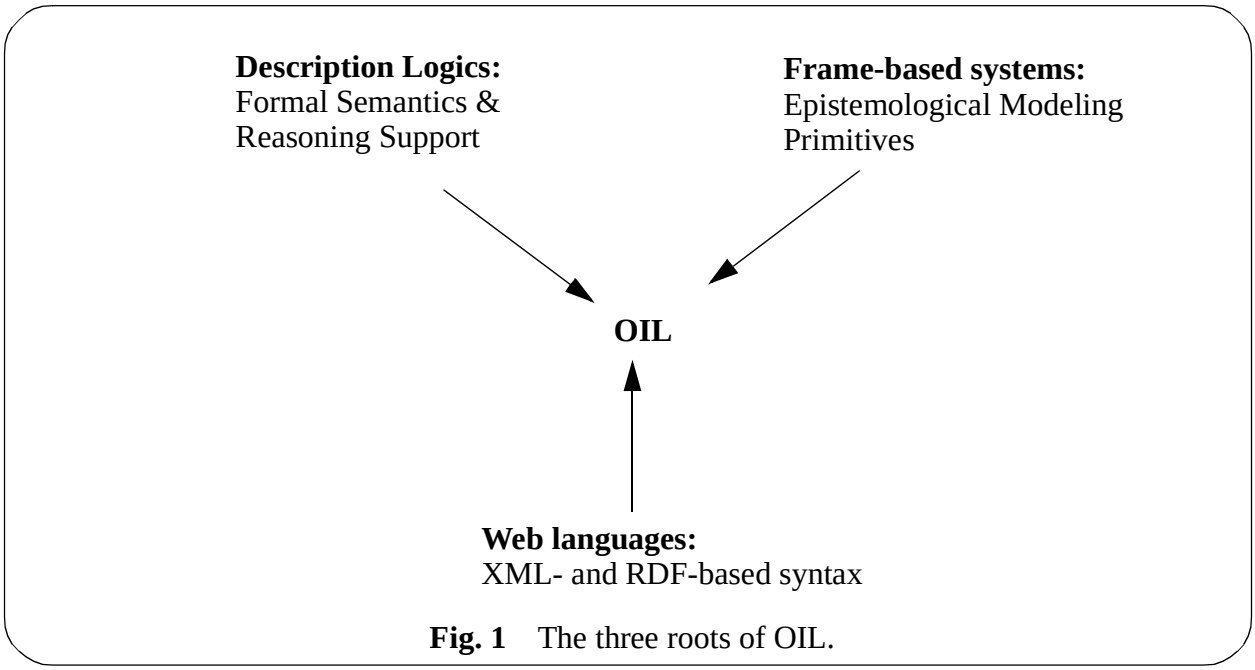

Fig. 1 The three roots of OIL.

1. http://dl.kr.org/. Links to most papers, project, and research events in this area can be found here.

and YAK.[2] 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 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 have examined 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). Despite the discouraging theoretical complexity which results, there are now efficient implementations for DL languages (cf. [Borgida & Patel-Schneider, 1994], [MacGregor, 1994], [Horrocks & Patel-Schneider, 1999]), see for example DLP[3] and the FaCT system.[4] 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. Other reasoning tasks are, for example, 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 languages and characterizing them in terms of their pros and cons.

Frame-based systems. The central modeling primitives of predicate logic are predicates. Frame-based and object-oriented approaches take a different point of view. Their central modeling primitives 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 value restrictions when defined for different classes. A frame provides a certain context for modeling one aspect of a domain. Many additional refinements of these modeling constructs have been developed and have led to the incredible success of this modeling paradigm. Many frame-based systems and languages have been developed, and under the name object-orientation the paradigm has also conquered the software engineering community. Therefore, OIL incorporates the *essential modeling primitives* of frame-based systems into its language. OIL is based on the notion of a concept and the definition of its superclasses and attributes. Relations can also be defined not as attributes of a class, but as 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 treatments 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 we make 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 we 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 shown with XOL[5]

---

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

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

4. 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.

(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[6] 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 exactly 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 primarily defined as an extension of XOL, although, as we said above for OKBC, we omit some of the original language primitives. Further candidates for a web-based syntax for OIL are RDF and RDFS. The Resource Description Framework (RDF)[7] (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 meta data. RDF schema (RDFS) [Brickley & Guha, 2000] provides 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. In relation 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[8] (cf. [Gruber, 1993], [Farquhar et al., 1997]) is an existing proposal for a Ontology Interchange Language. It was designed to support the design and specification of ontologies with a clear logical semantics based on KIF[9]. Ontolingua extends KIF with additional syntax to capture the intuitive bundling of axioms into definitional forms with ontological significance and a Frame Ontology to define object-oriented and frame-language terms.[10] 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

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

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

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

8. http://ontolingua.stanford.edu/

9. 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. Semantically, there are four categories of constants in KIF: object constants, function constants, relation constants, and logical constants. Object constants are used to denote individual objects. Function constants denote functions on those objects. Relation constants denote relations. Logical constants express conditions about the world and are either true or false. KIF is unusual among logical languages in that there is no syntactic distinction among these four types of constants; any constant can be used where any other constant can be used. This feature allows the reification of formulas as terms used in other formulas, making it possible to make statements over statements. This introduces second-order features in KIF, which provides an important extension of first-order logic.

10. 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.

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, which is provided without any means to control it. Not surprisingly, no reasoning support is provided with Ontolingua.[11] 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 have already shown in [Klein et al., 2000] that the current expressiveness of OIL is not sufficient for some purposes (see also Section 3.4). This may lead to a family of controlled extensions to the language. This will give us versions with different expressive power which can be applied in different cases as required. We believe that this approach is preferable to the definition of one single, large and unmanageable language**.**

In general there are two strategies for achieving a standard: Defining a "small" set of modeling primitives that are common across the community, and defining 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 worldwide standard. Meanwhile we have HTML version 5, XHTML, and XML. So, beginning with a core set, and successively refining and extending it, has proven to be a successful strategy. The second approach has been taken by the UML community by designing a model that is broad enough to cover all its modeling concepts. This leads to ambiguity and redundancy in modeling primitives and sometimes a precise semantic definition is lacking. However, UML has been adopted by the software industry as one of the major approaches in the meantime and is therefore also a success. Obviously, these two opposite approaches to standardization may both work successfully. We have chosen the first approach in developing OIL. This stems from the purpose for which OIL is designed. 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 to make Tim Berners-Lee's vision of a semantic web come true. Therefore, clear definitions of semantics and reasoning support are essential.

2.3 OIL and OKBC

A simple and well-defined semantics is of great importance for an ontology interchange language because it is used to transfer knowledge from one context to another. An ontology exchange standard already exists for frame-based systems: the Open Knowledge Base Connectivity (OKBC)[12] ([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[13] as an exchange standard

11. It may be possible to provide reasoning support for Ontolingua using ATP (see http://www.ksl.Stanford.EDU/software/ATP/), but neither the system nor any proof of its correctness is available.

12. http://www.ai.sri.com/~okbc/

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 in Section 4 of this paper.

2.4 OIL and RDF

In the same way as OIL provides an extension of OKBC (and is therefore downwards compatible with OKBC), OIL provides an extension of RDF and RDFS. Based on its RDF syntax, ontologies written in OIL are valid RDF documents. OIL extends the schema definition of RDFS with additional language primitives not yet present in RDFS. Based on these extensions, an ontology in OIL can be expressed in RDFS. A detailed comparison is made in Section 4 of this paper.

13. http://www.fipa.org

3 The OIL Language

This section provides an informal description of the modeling primitives, an example in OIL, its tool environment, and a discussion of future extensions of OIL.

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 and a formal specification and semantics (both of the language and of the common inference problems) will be given in Appendix C. To improve readability we will use a more compact pseudo XML syntax where opening tags are indicated by **bold faced** text, grouping of sub-content is indicated by indentation and closing tags are omitted.

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 about instances is currently beyond the scope of OIL. We describe an XML syntax based on an XML schema derived from OIL in [Klein et al., 2000].
- The first meta level, where the actual ontological definitions are provided. Here we define the terminology that may be instantiated at the object level. OIL is mainly concerned with this level. It is a means for describing structured vocabulary with well-defined semantics. The main contribution of OIL is with regard to this level.
- The second meta-level (i.e., the meta-meta level) is concerned with describing features of such an ontology, like author, name, subject, etc. For representing metadata of ontologies, we make use of the Dublin Core Metadata 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. It consists of 15 metadata elements (Title, Creator, Subject, Description, Publisher, Contributor, Date, Type, Format, Identifier, Source, Language, Relation, Coverage, and Rights) and is the result of international efforts consisting of an ongoing series of workshops. 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 meta-levels. The former is called the *ontology definition* and the latter is called the *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.

3.1.1 Ontology Container

We adopt the components as defined by the Dublin Core Metadata Element Set, Version 1.1 for the *ontology container* part of OIL. Although every element in the Dublin Core set is optional and repeatable, 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], and in the appendix. Here we provide our pseudo-XML syntax explained above.

title^{+} The name of the ontology, e.g., “African animals”.

creator^{+} The name of an agent (i.e., a person, a group of persons, or a software agent) that created the ontology.

subject* Keywords or classification code describing the subject the ontology deals with.

description Natural language text describing the content of the ontology, e.g., “A didactic example ontology describing African animals”. Besides this description, there is one special description element required, which has the **release** qualifier:

description.release The version of the ontology (a number), e.g, 1.01.

publisher* Defining the entity that is responsible for making the resource available.

contributor* The name of an agent (i.e., a person, a group of persons, or a software agent) that helped to create the ontology.

date* The date the ontology has been created, modified, or made available (see ISO 8601 for format instructions).

type^{+} The nature of the resource. A predefined and required value is *ontology*, although this value is not yet in the Working Draft of the resource types [Guenther, 1999].

format* The digital manifestation of the resource, recommended as a value is the MIME type of the resource, i.e. “`text/xml`”.

identifier^{+} The **URI** of the ontology.

source* Optional references (**URI**) to sources from which the ontology is derived. E.g., a reference to a plain text description of the domain on which the ontology is based.

language^{+} The language of the ontology. Obviously, one predefined and required value is “`OIL`”. Other elements can contain the language of the content of the ontology, according to RFC 1766.

relation* A list of references to other OIL ontologies. It is recommended to list all ontologies that are imported in the definition section with a **hasPart** qualifier. Other possible and meaningful qualifiers are **replaces**, **isReplacedBy**, **requires** and **isRequiredBy**. For example, to list an

imported ontology, we write: **relation.hasPart** "`http://www.ontosRus.com/animals/jungle.onto`".

rights* Information about rights held in and over the ontology.

3.1.2 An Ontology specified in OIL

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. Each reference consists of a **URI** specifying where the module is to be imported from, e.g., "`http://www.ontosRus.com/animals/jungle.onto`". XML schemas and OIL provide the same (limited) means for composing specifications. Specifications can be included and the underlying assumption is that names of different specifications are different (via different prefixes).[1]

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 can be Horn clauses, DL style axioms etc.), and they have no semantic significance. The rule base consists simply of a **type** (a string) followed by the unstructured rules (a string).[2]

definition* 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 in the class. For example, if the primitive class elephant is defined to be a sub-class of animal with a slot constraint stating that *skin-color* must be grey, then all instances of elephant must necessarily be animals with grey skin, but there may be grey-skinned animals that are not instances of elephant. When a class is **defined**, its definition is taken to be a necessary *and* sufficient condition for membership of the class. For example, if the defined class carnivore is defined to be a sub-class of animal with a slot constraint stating that it *eats* meat, then as all instances of carnivore are necessarily meat eating animals, and every meat eating animal is also an instance of carnivore.

name The name of the class (a string).

documentation? Some documentation describing the class (a string).

---

1. This definition is embryonal. See Section 3.4 for more details.
2. This definition is embryonal. See Section 3.4 for more details.

subclass-of? A list of one or more **class-expressions**, 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-constraint* Zero or more slot-constraints, a special kind of class-expression, the structure of which will be described below (note that a slot-constraint defines a class). The class being defined in this **class-def** must be a subclass of each slot-constraint.

A **class-expression** can be either a class name, a **slot-constraint**, or a boolean combination of class expressions using the operators **AND**, **OR** or **NOT**. The structure of these boolean combinations is as follows:

AND: A list of two or more class expressions that is to be treated as a conjunction. For example:

Meat **AND** Fish

defines the class whose instances are all those individuals that are instances of both the class Meat and the class Fish.

OR: A list of two or more class expressions that is to be treated as a disjunction. For example:

Meat **OR** Fish

defines the class whose instances are all those individuals that are instances of either the class Meat or the class Fish.

NOT: An expression taking as a parameter a single class expression that is to be negated. For example,

NOT Meat

defines the class whose instances are all those individuals that are not instances of the class Meat.

Note that class expressions are recursively defined, so that arbitrarily complex expressions can be formed. For example

NOT (Meat **OR** Fish)

defines the class whose instances are all those individuals that are not an instances of either the class Meat or the class Fish.

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). For example, if the pair (Leo, Willie) is an instance of the slot *eats*, Leo is an instance of the class lion and Willie is an instance of the class wildebeest, then Leo is also an instance of the **value** constraint wildebeest applied to the slot *eats*. 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, then 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 **has-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*: for each instance of the class, there exists at least one value for this slot that fulfils the 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 not mean that instances of the slot-constraint eat anything at all. **value-type** *expresses the universal (for-all) quantifier of Predicate logic*: for each instance of the class, every value for this slot must fulfill the 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. The class expression can be omitted, in which case an instance of the class defined by the slot-constraint can be related to at most *n* distinct individuals (regardless of their class) via the slot relation. For example, the **max-cardinality** constraint:

slot-constraint *friend*
max-cardinality 2 antelope

defines the class, each instance of which has at most 2 *friends* that are antelopes.

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. The class expression can be omitted, in which case an instance of the class defined by the slot-constraint must be related to at least *n* distinct individuals (regardless of their class) via the slot relation. For example, the **min-cardinality** constraint:

slot-constraint *friend*
min-cardinality 3 wildebeest

defines the class, each instance of which has at least 3 friends that are wildebeests. Note that conflicting cardinality constraints is one way in which logical inconsistencies can arise in an ontology. For example, a class to which both the above min-cardinality and max-cardinality constraints applied would be logically inconsistent (could have no instances) if the ontology correctly represented the fact that a wildebeest is a kind of antelope.

cardinality? A non-negative integer *n* followed (optionally) by a **class-expression**. This is simply shorthand for a pair of **min-cardinality** and **min-cardinality** constraints, both with the same *n* and **class-expression**. For example,

slot-constraint *friend*
cardinality 1 *zebra*

is equivalent to

slot-constraint *friend*
max-cardinality 1 *zebra*

min-cardinality 1 *zebra*

and defines the class, each instance of which has exactly 1 *friend* that is a *zebra*.

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 **slots**. 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-of*
subslot-of *child-of*

defines a slot *daughter-of* that is a subslot of *child-of*, i.e., every pair (*x,y*) that is an instance of *daughter-of* must also be an instance of *child-of*.

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. For example,

slot-def *eats*
domain *animal*

defines a slot *eats* such that any individual that *eats* another individual must be an instance of animal.

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. For example,

slot-def *friend*
range *animal*

defines a slot *friend* such that any individual that is a *friend* of another individual must be an instance of animal. Note that this is shorthand for adding a **value-type** slot-restriction to the list of classes in the **domain** of the slot.

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-by*, 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 The slot is transitive, i.e., if both (*x,y*) and (*y,z*) are instances of the slot, then (*x,z*) must also be an instance of the slot. For example,

slot-def *bigger-than*
properties *transitive*

defines the slot *bigger-than* to be transitive, so if *Jumbo* the *elephant* is *bigger-than Robbie* the *rhino*, and *Robbie* the *rhino* is *bigger-than Walter* the *warthog*, then *Jumbo* must be *bigger-than Walter*.

symmetric The slot is symmetric, i.e., if (*x*,*y*) is an instance of the slot, then (*y*,*x*) must also be an instance of the slot. For example,

slot-def *lives-with*
properties *symmetric*

defines the slot *lives-with* to be symmetric, so if *Zoe* the *zebra lives-with* Willie the *wildebeest*, then *Willie* also *lives-with Zoe*.

3.2 An example OIL ontology

The following example of an OIL ontology illustrates some of the key features of the language. The ontology is intended purely for didactic purposes and is not to be taken as an example of good modeling practice.

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
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 herbivore
subclass-of NOT carnivore
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 classes herbivore and carnivore are made disjoint using a second class definition for herbivore (OIL supports multiple class definitions). Note that if "**subclass-of NOT** carnivore" were included in the first definition, then giraffe would not be recognized as a sub-class of herbivore (because it is not declared to be sub-class of **NOT** carnivore).
- 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 Tools

One of the major benefits of OIL is that it comes with a range of tools to support ontology design, exchange, integration and verification. In particular, it is possible to use the FaCT reasoner to check

the consistency of all the class definitions in an ontology, and to discover sub-class/super-class (subsumption) relations that are implied by the definitions in the ontology but not explicitly stated. FaCT **(Fa**st **C**lassification of **T**erminologies) is a Description Logic (DL) classifier that can also be used for consistency checking in modal and other similar logics. The FaCT system includes two reasoners, one for the logic *SHF* and the other for the logic *SHIQ*, both of which use optimized implementations of sound and complete tableaux algorithms. 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.

3.3.1 Background

The logic implemented in FaCT is based on ALC_{R^+}, an extension of *ALC* to include transitive roles [Sattler, 1996]. For compactness, this logic has be called *S* (due to its relationship with the proposition multi-modal logic $\mathbf{S4_{(m)}}$ [Schild, 1991a]). *SHF* extends *S* with a hierarchy of roles and functional roles (attributes), while *SHIQ* adds inverse roles and fully qualified number restrictions.

The *SHIQ* reasoner is of particular interest, both from a theoretical and a practical viewpoint. Adding inverse roles to *SHF* (resulting in *SHIF*) leads to the loss of the finite model property, and this has necessitated the development of a more sophisticated *double dynamic* blocking strategy that allows the algorithm to find finite representations of infinite models while still guaranteeing termination [Horrocks & Sattler, 1999]. Moreover, when *SHIF* is generalized to *SHIQ*, it is necessary to restrict the use of transitive roles in number restrictions in order to maintain decidability [Horrocks et al., 1999]. *SHIQ* is also of great practical interest as it is powerful enough to encode the logic *DLR*, and can thus be used for reasoning about a wide range of conceptual data models, e.g., Extended Entity-Relationship (EER) schemas [Calvanese et al., 1998a].

3.3.2 Implementation

FaCT is implemented in Common Lisp, and has been run successfully with several commercial and free Lisps, including Allegro, Liquid (formerly Lucid), Lisp works and GNU. Binaries (executable code) are now available (in addition to the source code) for Linux and Windows systems, allowing FaCT to be used without a locally available Lisp. In order to make the FaCT system usable in realistic applications, a wide range of optimization techniques are used to implement the satisfiability testing algorithms. These include axiom absorption, lexical normalization, semantic branching search, simplification, dependency directed backtracking, heuristic guided search and caching [Horrocks & Patel-Schneider, 1999]. The use of these (and other) optimization techniques has now become standard in tableaux-based DL implementations [Patel-Schneider, 1998], [Haarslev et al., 1998]. Work is underway on the development of Abox reasoning for the FaCT system (reasoning with individuals): a *SHF* Abox has recently been released [Tessaris & Gough, 1999] and a full *SHIQ* Abox is being developed [Horrocks et al., submitted].

3.3.3 CORBA Interface

In addition to the standard KRSS functional interface [Patel-Schneider & Swartout, 1993], FaCT can

also be configured as a classification and reasoning server using the Object Management Group's Common Object Request Broker Architecture (CORBA) [Bechhofer et al., 1999]. This approach has several advantages: it facilitates the use of FaCT by non-Lisp client applications; the API is defined using CORBA's Interface Definition Language (IDL), which can be mapped to various target languages; a mechanism is provided for applications to communicate with the DL system, either locally or remotely; and server components can be added/substituted without client applications even being aware of the change. This has allowed, for example, the successful use of FaCT's reasoning services in a (Java based) prototype EER schema integration tool developed as part of the DWQ project [Calvanese et al., 1999].

3.3.4 Performance

FaCT's optimizations are aimed specifically at improving the system's performance when classifying realistic ontologies and this results in a performance improvement 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 effectively 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.[3] In contrast, the KRIS system [Baader & Hollunder, 1991] was unable to complete the same task after several weeks of CPU time.

3.4 Current Limitations of OIL

Our starting point was to define a 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 must be clear that this may compromise reasoning support. This seems to us a better solution than trying to define a single "all things to all men" language like Ontolingua. 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 explain why we chose them, 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. If an attempt is made at "overwriting" an inherited attribute value, this will simply result in inconsistent class definitions which have an empty extension. For example, if we define the class "CS professor" with attribute "gender" and value "male", and we subsequently define a subclass for which we define the gender attribute as "female", this subclass will be inconsistent and have an empty extension (assuming that "male" and "female" are disjoint).

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 the

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

items in the ontology. Such a powerful feature is undoubtedly useful. The use of OIL as an exchange language further justifies a more powerful axiom-language as you might need such axioms to enforce the correct interpretation of the source ontology when mapping into OIL. The lack of such an axiom-language is somewhat mitigated in OIL by the fact that we have a powerful concept and slot definition language. The main limitation is that we do not have *composite* definitions of relations. However, there is currently no broad support for any particular choice in this matter. The main problems in this area are first, that it is difficult to identify a common set of rule/axiom expressions that can be standardized, and second, that you have to define properly how these axioms can be integrated with the other modeling primitives of OIL.

Further algebraic properties: The lack of an axiom language can also be compensated for somewhat by extending the set of properties that can be specified for relations in OIL. Currently this set contains *inverse*, *transitivity* and *symmetry*. Other reasonable candidates are *reflexivity*, *irreflexivity*, *antisymmetry*, *asymmetry*, *linearity* (aRb bRa for any pair a,b), *connectivity* (aRb or a=b or bRa for any pair a,b), *partial order* and *total order*. (Notice that some of these can be defined in terms of each other), cf. [Staab & Mädche, 2000].

Modules: Section 3.1 presented a very simple construction to modularize ontologies in OIL. In fact, this mechanism is identical to the namespace mechanism in XML and XML schema. 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. However, much more elaborate mechanisms would be required for the structured representation of large ontologies. Means of renaming, restructuring, and redefining imported ontologies must be available. Future extensions will cover parameterized modules, signature mappings between modules, and restricted export interfaces for modules. We will use the generic adapter concept of UPML (cf. [Fensel et al., 1999a]) specialized to the fixed set of language primitives of OIL as [Gennari et al., 1994], [Park et al., 1997] have developed for the fixed set of language primitives of Protégé.

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 domain-instances are allowed in class definitions [Schaerf, 1994], [Blackburn & Seligman, 1998], [Areces et al., 1999]. For this reason, OIL currently does not allow the use of instances in slot-values, or extensional definitions of classes (i.e., class definitions by enumerating the class instances). It is not clear how serious a restriction this is for an ontology language, as ontologies should, in general, be independent of specific instantiations—it may be that in many cases, “individuals” can more correctly be replaced with a primitive class or classes.

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 language, and extensions of OIL in this direction are probably required. The theory of concrete domains is well understood [Baader & Hanschke, 1991], and it should be possible to add some restricted form of concrete domains (but still with greater expressive power than XOL's *numeric-minimum* and *numeric-maximum* slot constraints) to OIL's core language without compromising its decidability (but a corresponding extension to the FaCT system would also be required if reasoning support is to be provided).

Limited Second-order expressivity: Many existing languages for ontologies (KIF, CycL [Lenat & Guha, 1990], Ontolingua) include some form of reification mechanism, which allows the treatment of statements of the language as objects in their own right, thereby making it possible to express statements over these statements. A full second order extension would be clearly undesirable (even unification is un-decidable 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 problems (in effect, they are simply 2nd order syntactic sugar for what are essentially first order constructions). A precise characterization of such expressivity is required in a future extension of OIL. OIL is currently very restricted. Only classes are provided, not meta-classes or individuals.

4 Comparing OIL with other approaches

This section compares OIL with other frame-based approaches and with emerging web standards such as XML and RDF.

4.1 OIL and other frame-oriented approaches

As discussed in Section 2, the modeling primitives of OIL are based on those of XOL. 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) 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 in OIL for which reasoning support (e.g., for class consistency and subsumption checking) could not be provided, either because their semantics are unclear or because their inclusion would lead to the undecidability of the language. However, it is envisaged that this *core* OIL will be extended in the future with sets of additional primitives covering areas such as concrete data types (e.g., numbers and strings) and extensional class definitions, with the proviso that full reasoning support may not be available for ontologies using such primitives.

4.1.1 Frame-based versus logic-based

OIL is fundamentally frame based, partly for reasons of upward compatibility with XOL (as discussed before), and partly because frame-based modeling is very intuitive for many users. Moreover, DL approaches can be seen as an extension and generalization of the frame idea, with frames being closely related to DL concepts and slots being very closely related to DL roles. The main differences stem from the fact that frames generally provide quite a rich set of primitives, but impose very restrictive syntactic constraints on how primitives can be combined and on how they can be used to define a class. DLs on the other hand generally have a more restricted set of primitives (they are constrained by requirements for clear semantics, decidability and the provision of practical reasoning procedures), but allow primitives to be combined in arbitrary boolean expression and used to define different kinds of class (in particular *primitive* classes, where the definition is taken to be a necessary condition for membership of the class, and *non-primitive* classes, where the definition is taken to be both a necessary and sufficient condition for membership of the class).

A central difference between frame-based approaches and approaches based on Description Logics are that the former rely solely on explicit statements of class-subsumption, whereas the latter are able to efficiently compute the subsumption relationship between classes on the basis of the intensional definition of these classes. Other relations between classes such as disjointness, consistency etc., can all be expressed in terms of the same subsumption relationship. The ability to automatically compute these relations is important for verification of ontologies. This may be less important with small local ontologies that are probably designed by one expert person. However, our intention is to exchange, share, reuse and merge ontologies. In such a case, reasoning support can be very valuable tool. This has been demonstrated even for database schema integration, which should be much easier than integrating ontologies.

4.1.2 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[1] the class formed from the conjunction of all the slots and all the parent classes. Consequently, each 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), as well as allowing 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.[2]

In our opinion, this very restricted form of class definition makes XOL (and indeed OKBC) unsuitable as a standard ontology 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.[3] 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.[4] This makes it impossible to capture the fact that 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-ancestor*, 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].

Finally, the semantics of OKBC (on which XOL relies) are relatively informally specified, and have idiosyncrasies that are difficult to either formalize or justify.

In OIL we propose to solve these problems by providing the language with a well defined semantics and by extending XOL in the following ways.

- Arbitrary boolean expressions (called *class expression*s) are allowed wherever a class name can appear.

---

1. The OKBC semantics (on which XOL relies) are less than clear on this point, and on several other important points.
2. In contrast, OKBC supports the definition of both primitive and non-primitive classes.
3. This aspect of the definition can be captured in OKBC as non-primitive classes are supported.
4. For example extended entity relationship (EER) modeling [Calvanese et al., 1998b].

- A slot definition can be treated as a class and can be used in class expressions.
- Class definitions have an (optional) additional field that specifies whether the class is primitive or non-primitive (the default is primitive).
- A class can be used as a slot **has-value** and is taken to specify that the slot must have at least one filler that is an instance of the given class.
- Global slot definitions are extended to allow the specification of parent slots and of relation properties such as **transitive**, and **symmetrical**.
- The additional rules governing XOL documents (see [Karp et al., 1999], Appendix 2]) are not required in OIL (e.g., there is no restriction on the ordering of class and slot definitions).

4.1.3 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 directly within OIL. 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 [Schaerf, 1994], [Blackburn & Seligman, 1998], [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. Currently, we provide an XML schema definition for capturing instances of an ontology in OIL (cf. [Klein et al., 2000]).
- 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 Web Standards

When discussing the relationship between OIL and web standards mainly two interesting candidates come into mind: XML and RDF. In this section, we will discuss possible ways to relate OIL with them.

4.2.1 XML

XML can be used as a serial syntax for OIL. Such a syntax is very useful because it puts OIL in the mainstream of tools that are currently being developed for supporting XML-based documents. Validation and rendering techniques developed for XML can directly be used for ontologies specified in OIL. Therefore, the appendix of this paper provides the definition of a DTD that defines constraints on valid documents in OIL.

Meanwhile a successor of DTDs called XML schemas have been published as a proposal by the W3C (cf. [Biron & Malhotra, 1999], [Thompson et al., 1999], [Walsh, 1999]). The main improvements of XML schemas compared to DTDs are:

- XML schemas definitions are themselves XML documents.
- XML schemas provide a rich set of datatypes that can be used to define the values of elementary tags.
- XML schemas provide much richer means for defining nested tags (i.e., tags with subtags).
- XML schemas provide the namespace mechanism to combine XML documents with heterogeneous vocabulary.

Therefore, it was natural to also define the XML syntax of OIL by using the XML schema mechanism (see the appendix).[5] However, a more significant question is whether XML schemas also allow the capturing of some of the semantics of ontologies specified in OIL. Central to an ontology is the is-a relationship, and XML schemas incorporate the notion of inheritance. In addition to the direct XML schema syntax of OIL we provide in the appendix, we discuss in [Klein et al., 2000] a more complex translation procedure that leads to XML documents capturing more aspects of the semantics of an ontology in OIL. This includes the use of type refinement as present in XML schemas to model the subsumption relationship between concepts in OIL.

4.2.2 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 represents 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. XML was chosen for this purpose. RDF and XML are complementary 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 data, a schema language with appropriate primitives is needed. The RDF-schema specification was created for this purpose. RDF-schema is a simple ontology language able to define basic vocabularies. This language covers the simplest parts 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 much closer than that between OIL and XML Schemas. This is not surprising, since XML-schema was meant to generalize the way of defining the structure of valid XML-documents and RDF/RDFS was meant to capture meaning in the manner of

5. We also provide the definition via a DTD because XML schemas are still a proposal and may change in the near future and currently do not provide much tool support.

semantic nets. In the same way as RDF-Schema is used to define itself it can also be used to define other ontology languages. We have therefore defined a syntax for OIL by giving an RDF-schema for the core of OIL, and proposing related RDF-schemas that could complement this core by covering 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.

- The major integration points of RDF/RDFS and OIL are defined by the abstract OIL class ClassExpression which is a subclass of rdfs:Resource (the most general class in RDFS) and by the abstract OIL class OntologyConstraint which is a subclass of rdfs:ConstraintResource.
- Since rdfs:Class is a specific case of a class expression, its definition is extended to make it a subclass of ClassExpression.
- 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 number of classes that specify inverse and transitive roles and cardinality constraints, not originally possible in RDF/ RDFS.
- The way class expressions are defined in RDF is nearly literally equivalent to the abstract syntax defined in Section 3.

The "Appendix B: OIL Syntax in RDF" provides the RDF-Schema specification of OIL. In [Broekstra et al., to appear] the relation between OIL and RDF is examined in more detail.

5 Summary

In this paper, we have proposed both a syntax and a semantics of an ontology inference layer for the WWW based on XML and RDF schemas called OIL. One of our main motivations has been to try to ensure that such a proposed standard had a clear and well defined semantics—a common syntax is useless without an agreement as to what it all means.

The core we have currently defined can be justified from both 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 appears quite natural to us to limit the expressiveness of this version so as to make subsumption decidable. This defines a well-understood subfragment of first-order logic. However, it is important to note that we are open for further discussions that may influence the final design of the language. Clearly future versions will provide variants with more expressive power which lack this reasoning support. Connecting OIL with Horn logic is probably the most challenging question and we will see how far we can get there.

We are currently evaluating the use of OIL in the two running IST projects, On-to-knowledge[1] and Ibrow[2]. In On-to-knowledge OIL will be extended to 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 user in 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., 1998], [Fensel et al., 1999b]): using ontologies to model and annotate the semantics of information in a machine processable manner.

Acknowledgement. We thank Monica Crubezy, Enrico Franconi, W. Grosso, Peter Karp, Robin McEntire, Mark Musen, and Guus Schreiber for helpful comments on drafts of the paper; and Jeff Butler for correcting the English.

1. *On-To-Knowledge: Content-driven Knowledge-Management Tools through Evolving Ontologies* started in January 2000. Project partners are the Vreije Universiteit Amsterdam (VU); the Institute AIFB, University of Karlsruhe, Germany; AIdministrator, the Netherlands; British Telecom Laboratories, UK; Swiss Life, Switzerland; CognIT, Norway; and Enersearch, Sweden. http://www.ontoknowledge.org

2. *IBROW* started with a pre-phase under the 4th European Framework and has become a full-fledged Information Society Technologies (IST) project under the 5th European Framework Program since February 2000. Results of its initial phase are described in [Benjamins et al., 1999 (b)], [Fensel & Benjamins, 1998], and [Fensel et al., 1999a]. Project partners are the University of Amsterdam; the Open University, UK; the Spanish Council of Scientific Research (IIIA) in Barcelona, Spain; the Institute AIFB, University of Karlsruhe, Germany: Stanford University, US: Intelligent Software Components S. A., Spain; and the Vrije Universiteit Amsterdam. http://www.swi.psy.uva.nl/projects/ibrow/home.html

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 Bulleti*n, 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.

[Benjamins et al., 1999 (a)]
V. R. Benjamins, D. Fensel, S. Decker, and A. Gomez Perez: $(KA)^2$: Building ontologies for the internet: a mid term report. *International Journal of Human-Computer Studie*s, 51:687–712, 1999.

[Benjamins et al., 1999 (b)]
V. R. Benjamins, B. Wielinga, J. Wielemaker, and D. Fensel : Brokering Problem-Solving Knowledge at the Internet. In Knowledge Acquisition, Modeling, and Management, Proceedings of the European Knowledge Acquisition Workshop (EKAW-99), D. Fensel et al. (eds.), Lecture Notes in Artificial Intelligence, LNAI 1621, Springer-Verlag, May 1999.

[Biron & Malhotra, 1999]
P. V. Biron and A. Malhotra: XML schema part 2: Datatypes. http://www.w3.org/TR/1999/WD-xmlschema-2-19991217/, 1999. W3C Working draft.

[Blackburn & Seligman, 1998]
P. Blackburn and J. Seligman: What are hybrid languages? In M. Kracht, M. de Rijke, H. Wansing, and M. Zakharyaschev, editors, *Advances in Modal Logi*c, volume 1, pages 41–62. CSLI Publications, Stanford University, 1998.

[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 Researc*h, 1:277–308, 1994.

[Brachman & Schmolze, 1985]
R. J. Brachman and J. G. Schmolze: An overview of the KL-ONE knowledge representation system. *Cognitive Scienc*e, 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.

[Broekstra et al., to appear]
J. Broekstra, M. Klein, D. Fensel, S. Decker and I. Horrocks. OIL: a case study in extending RDF Schema. To appear.

[Calvanese et al., 1998a]
D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati: Source integration in data warehousing. In *Proc. of DEXA-9*8, pages 192–197, 1998.

[Calvanese et al., 1998b]
D. Calvanese, M. Lenzerini, and D. Nardi: Description logics for conceptual data modeling. In Jan

Chomicki and G¨ unter Saake, editors, *Logics for Databases and Information Systems*, pages 229–263. Kluwer Academic Publisher, 1998.

[Calvanese et al., 1999]
D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati: Use of the data reconciliation tool at telecom italia. DWQ deliverable D.4.3, Foundations of Data Warehouse Quality (DWQ), 1999.

[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.

[Decker et al., 1999]
S. Decker, M. Erdmann, D. Fensel, and R.Studer: Ontobroker: Ontology Based Access to Distributed and Semi-Structured Information. In: R. Meersman et al. (eds.), *Database Semantics: Semantic Issues in Multimedia Systems, Proceedings TC2/WG 2.6 8th Working Conference on Database Semantics (DS-8)*, Rotorua, New Zealand, Kluwer Academic Publishers, Boston, 1999.

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

[Elmasri & Navathe, 2000]
R. Elmasri and S. B. Navathe: *Fundamentals of Database Systems*. Addison Wesley, 3rd edition, 2000.

[Errikson et al., 1994]
H. Eriksson, A. R. Puerta, and M. A. Musen: Generation of knowledge-acquisition tools from domain ontologies, *International Journal of Human Computer Studies (IJHCS)*, 41:425-453, 1994.

[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 Studies*, 46:707–728, 1997.

[Fensel, 2000]
D. Fensel. *Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce*. Springer-Verlag, Berlin, 2000.

[Fensel & Benjamins, 1998]
D. Fensel and V. R. Benjamins: Key Issues for Problem-Solving Methods Reuse. In *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI-98)*, Brighton, UK, August 1998, 63-67.

[Fensel et al., 1998]
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., 1999a]
Dieter Fensel, V. Richard Benjamins, Enrico Motta, and Bob Wielinga: UPML: A Framework for knowledge system reuse. In *Proceedings of the International Joint Conference on AI (IJCAI-99)*, Stockholm, Sweden, July 31 - August 5, 1999.

[Fensel et al., 1999b]
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 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.

[Van Gelder et al., 1991]
A. Van Gelder, K. Ross, and J. Schlipf: The well-founded semantics for general logic programs. *Journal of the AC*M, 38(3):620–650, 1991.

[Gennari et al., 1994]
J. H. Gennari, S. W. Tu, T. E. Rothenfluh, and M. A. Musen: Mapping Domains to Methods in Support of Reuse, *International Journal of Human-Computer Studies (IJHCS)*, 41:399–424, 1994.

[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, J. 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

[Haarslev et al., 1998]
V. Haarslev, R. Möller, and A.-Y. Turha:. Implementing an *ALLRP(D)* abox reasoner – progress report. In *Proc. of DL'9*8, pages 82–86, 1998.

[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, 1997]
I. Horrocks: *Optimising Tableaux Decision Procedures for Description Logic*s. PhD thesis, University of Manchester, 1997.

[Horrocks et al., submitted]
I. Horrocks, U. Sattler, and S. Tobies: Abox reasoning for *ALLRP(D)*, submitted.

[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.

[Horrocks & Sattler, 1999]
I. Horrocks and U. Sattler: A description logic with transitive and inverse roles and role hierarchies.

Journal of Logic and Computation, 9(3):385–410, 1999.

[Horrocks et al., 1999]
I. Horrocks, U. Sattler, and S. Tobies: Practical reasoning for expressive description logics. In H. Ganzinger, D. McAllester, and A. Voronkov, editors, *Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99*), number 1705 in Lecture Notes in Artificial Intelligence, pages 161–180. Springer-Verlag, 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.

[Kifer et al., 1995]
M. Kifer, G. Lausen, and J. Wu: Logical foundations of object-oriented and frame-based languages. *Journal of the AC*M, 42, 1995.

[Klein et al., 2000]
M. Klein, D. Fensel, F. van Harmelen, and I. Horrocks: The relation between ontologies and schema-languages: Translating OIL-specifications in 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.

[Kushmerick, 1997]
N. Kushmerick: *Wrapper Induction for Information Extraction*. Ph.D. Dissertation, Department of Computer Science & Engineering, University of Washington, 1997. Available as Technical Report UW-CSE-97-11-04.

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

[Lenat & Guha, 1990]
D. B. Lenat and R. V. Guha: *Building Large Knowledge-Based Systems: Representation and Inference in the* CYC *Project.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1990.

[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.

[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.

[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.

[Meersman, 1999]
R. A. Meersman: The use of lexicons and other computer-linguistic tools in semantics, design and cooperation of database systems. In Yanchun Zhang, editor, *Proceedings of the Second International Symposium on Cooperative Database Systems for Advanced Applications CODAS'9*9, Wollongong, Australia, 1999. Springer Verlag.

[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.

[OMG, 1997]
Object Management Group (OMG): Meta object facility (MOF) specification, 1997.

[OMG, 1998]
Object Management Group (OMG): Stream-based model interchange, 1998.

[Park et al., 1997]
J. Y. Park, J. H. Gennari, and M. A. Musen: Mappings for Reuse in Knowledge-based Systems. SMI Technical Report 97-0697, 1997.

[Patel-Schneider, 1998]
P. F. Patel-Schneide: DLP system description. In *Proc. of DL'9*8, pages 87–89, 1998.

[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.

[Sattler, 1996]
U. Sattler: A concept language extended with different kinds of transitive roles. In G. Görz and S. Hölldobler, editors, *20. Deutsche Jahrestagung für Künstliche Intelligenz*, number 1137 in Lecture Notes in Artificial Intelligence, pages 333–345. Springer-Verlag, 1996.

[Schaerf, 1994]
A. Schaerf: Reasoning with individuals in concept languages. *Data and Knowledge Engineerin*g, 13(2):141–176, 1994.

[Schild, 1991a]
K. Schild: A correspondence theory for terminological logics: Preliminary report. In *Proc. of IJCAI-9*1, pages 466–471, 1991.

[Schild, 1991b]
Klaus Schild: From terminological logics to modal logics. In *Proceedings of the International Workshop on Terminological Logic*s, pages 101–104, Dagstuhl, 1991.

[Staab & Mädche, 2000]
S. Staab and A. Mädche: Axioms are objects, too - ontology engineering beyond the modeling of concepts and relations. Technical Report 399, AIFB, Karlsruhe University, 2000.

[Stuckenschmidt, submitted]
H. Stuckenschmidt: Problem-solving methods for semantic integration. Submitted.

[Studer et al., 1998]
R. Studer, V. R. Benjamins, and D. Fensel: Knowledge engineering: Principles and methods. *Data and Knowledge Engineering (DKE*), 25(1–2):161–197, 1998.

[Tessaris & Gough, 1999]
S. Tessaris and G. Gough: Abox reasoning with transitive roles and axioms. In *Proc. of DL'9*9, 1999.

[Thompson et al., 1999]
H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn: XML schema part 1: Structures. http://www.w3.org/TR/1999/WD-xmlschema-1-19991217/, 1999. W3C Working draft.

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

[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.

[Walsh, 1999]
N. Walsh: Schemas for XML, 1999. http://www.xml.com/pub/1999/07/schemas/index.html

Appendix A: OIL Syntax in XML

This appendix provides an XML-based syntax definition of OIL. First we define a DTD for OIL. Then we use XML schemas to define the syntax of OIL. Finally we provide the example of Section 3 in XML syntax.

Appendix A.1 A DTD for OIL

The XML syntax of OIL ontologies is defined by the following DTD.

```
<!-- DTD for Ontology Integration Language OIL -->

<!ELEMENT ontology (ontology-container, ontology-definitions)>

<!-- Ontology container -->
<!ELEMENT ontology-container(rdf:RDF)>
<!-- This part contains meta-data about the ontology.
    It is formatted according [Miller et al., 1999] -->
    <!ELEMENT rdf:RDF (rdf:Description)>
    <!ATTLIST rdf:RDF
        xmlns:rdf CDATA #FIXED "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:dc  CDATA #FIXED "http://purl.oclc.org/dc#"
        xmlns:dcq CDATA #FIXED "http://purl.org/dc/qualifiers/1.0/"
     >
    <!ELEMENT rdf:Description ((dc:Title+, dc:Creator+, dc:Subject*, dc:Description+,
                    dc:Publisher*, dc:Contributor*, dc:Date*, dc:Type+,
                dc:Format*, dc:Identifier+, dc:Source*,
                dc:Language+, dc:Relation*, dc:Rights*) |
                (dcq:descriptionType, rdf:value) |
                (dcq:relationType, rdf:value) )>
    <!ATTLIST rdf:Description about CDATA #IMPLIED >
    <!ELEMENT dc:Title (#PCDATA)>
    <!ELEMENT dc:Creator (#PCDATA)>
    <!ELEMENT dc:Subject (#PCDATA)>
    <!ELEMENT dc:Description (#PCDATA | rdf:Description)*>
    <!ELEMENT dc:Publisher (#PCDATA)>
    <!ELEMENT dc:Contributor (#PCDATA)>
    <!ELEMENT dc:Date (#PCDATA)>
    <!ELEMENT dc:Type (#PCDATA)>
    <!ELEMENT dc:Format (#PCDATA)>
    <!ELEMENT dc:Identifier (#PCDATA)>
    <!ELEMENT dc:Source (#PCDATA)>
    <!ELEMENT dc:Language (#PCDATA)>
    <!ELEMENT dc:Relation (#PCDATA | rdf:Description)*>
    <!ELEMENT dc:Rights (#PCDATA)>
    <!ELEMENT dcq:descriptionType (#PCDATA)>
    <!ELEMENT dcq:relationType (#PCDATA)>
    <!ELEMENT rdf:value (#PCDATA)>

<!-- Ontology-definitions -->
<!ELEMENT ontology-definitions (imports?, rule-base?,
                    (class-def | slot-def)* )>
    <!-- Import-section with URI's to other ontology-files -->
    <!ELEMENT imports(URI)+>
```

```
    <!ELEMENT URI (#PCDATA)>
    <!-- Rules with URL to definition -->
    <!ELEMENT rule-base (#PCDATA)>
    <!ATTLIST rule-base type CDATA #REQUIRED>

<!-- Class-expressions -->
<!ENTITY % class-expr "( class | slot-constraint | AND | OR | NOT)">
    <!ELEMENT AND ((%class-expr;), (%class-expr;)+)>
    <!ELEMENT OR ((%class-expr;), (%class-expr;)+)>
    <!ELEMENT NOT (%class-expr;)>

<!-- Class-definition -->
<!ELEMENT class-def (class, documentation?,
            subclass-of?, slot-constraint*)>
    <!ATTLIST class-def type ( primitive | defined ) "primitive">
    <!-- Class-name -->
    <!ELEMENT class EMPTY>
     <!ATTLIST classname CDATA #REQUIRED>
    <!ELEMENT documentation (#PCDATA)>
    <!ELEMENT subclass-of(%class-expr;)+>

<!-- Slot-definition -->
<!ELEMENT slot-def (slot, documentation?, subslot-of?,
          domain?, range?, inverse?, properties?)>
    <!-- Slot-name -->
    <!ELEMENT slot EMPTY>
    <!ATTLIST slot name CDATA #REQUIRED>
    <!ELEMENT subslot-of (slot)+>
    <!ELEMENT domain (%class-expr;)+>
    <!ELEMENT range (%class-expr;)+>
    <!ELEMENT inverse (slot)>
    <!-- Slot-properties -->
    <!ELEMENT properties( transitive | symmetric | other )*>
        <!ELEMENT transitive EMPTY>
        <!ELEMENT symmetric EMPTY>
        <!ELEMENT other (#PCDATA)>

<!-- Slot-constraint -->
<!ELEMENT slot-constraint (slot, (has-value | value-type | cardinality |
                  max-cardinality | min-cardinality )+ )>
    <!ELEMENT has-value (%class-expr;)+>
    <!ELEMENT value-type (%class-expr;)+>
    <!ELEMENT cardinality (number, %class-expr;)+>
    <!ELEMENT max-cardinality (number, %class-expr;)+>
    <!ELEMENT min-cardinality(number, %class-expr;)+>
    <!ELEMENT number (#PCDATA)>
```

According to http://www.w3.org/DesignIssues/Syntax and http://www-db.stanford.edu/~melnik/rdf/syntax.html a more comprehensible RDF serialization in XML will be coming. Our oil-container would become much more simple and intuitive:

```
<?xml version="1.0" encoding="UTF-8"?>
<ontology>
  <ontology-container rdf:for="http://www.cs.vu.nl/~dieter/oil/"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
```

```
      xmlns:dc="http://purl.oclc.org/dc#"
      xmlns:dcq="http://purl.org/dc/qualifiers/1.0/">
   <dc:Title>African animals</dc:Title>
   <dc:Creator>Ian Horrocks</dc:Creator>
   <dc:Subject>animal, food, vegetarians</dc:Subject>
   <dc:Description>A didactic example ontology describing African animals</dc:Description>
   <dc:Description dcq:descriptionType="Release">1.01</dc:Description>
   <dc:Publisher>I. Horrocks</dc:Publisher>
   <dc:Type>ontology</dc:Type>
   <dc:Format>pdf</dc:Format>
   <dc:Identifier>http://www.cs.vu.nl/~dieter/oil/TR/oil.pdf</dc:Identifier>
   <dc:Source>http://www.africa.com/nature/animals.html</dc:Source>
   <dc:Language>OIL</dc:Language>
   <dc:Language>en-uk</dc:Language>
   <dc:Relation dcq:relationType="hasPart">
      http://www.ontosRus.com/animals/jungle.onto
   </dc:Relation>
  </ontology-container>
</ontology>
```

Appendix A.2 An XML schema definition of OIL

This section provides the OIL-syntax defined by an XML-Schema definition. It is equivalent to the DTD definition of OIL.

```
<!DOCTYPE schema PUBLIC "-//W3C//DTD XMLSCHEMA 19991216//EN"
        "WD-xmlschema-1-19991217/structures.dtd" [
        <!ATTLIST schema xmlns:x CDATA #IMPLIED> <!-- keep this schema XML1.0 valid -->
]>
<schema targetNamespace="oil.dtd">
    <type name="ontology" content="elementOnly">
        <group order="seq">
            <element name="ontology-container"/>
            <element name="ontology-definitions"/>
        </group>
    </type>
    <type name="ontology-container" content="elementOnly">
        <element name="rdf:RDF"/>
    </type>
    <type name="rdf:RDF" content="elementOnly">
        <element name="rdf:Description"/>
        <attribute name="xmlns:dcq" type="string" minOccurs="1" fixed="http://purl.org/dc/qualifiers/1.0/"/>
        <attribute name="xmlns:rdf" type="string" minOccurs="1"
            fixed="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
        <attribute name="xmlns:dc" type="string" minOccurs="1" fixed="http://purl.oclc.org/dc#"/>
    </type>
    <type name="rdf:Description" content="elementOnly">
        <group order="choice">
            <group order="seq">
                <group maxOccurs="*">
                    <element name="dc:Title"/>
                </group>
                <group maxOccurs="*">
                    <element name="dc:Creator"/>
                </group>
                <group maxOccurs="*" minOccurs="0">
                    <element name="dc:Subject"/>
```

```
            </group>
            <group maxOccurs="*">
                <element name="dc:Description"/>
            </group>
            <group maxOccurs="*" minOccurs="0">
                <element name="dc:Publisher"/>
            </group>
            <group maxOccurs="*" minOccurs="0">
                <element name="dc:Contributor"/>
            </group>
            <group maxOccurs="*" minOccurs="0">
                <element name="dc:Date"/>
            </group>
            <group maxOccurs="*">
                <element name="dc:Type"/>
            </group>
            <group maxOccurs="*" minOccurs="0">
                <element name="dc:Format"/>
            </group>
            <group maxOccurs="*">
                <element name="dc:Identifier"/>
            </group>
            <group maxOccurs="*" minOccurs="0">
                <element name="dc:Source"/>
            </group>
            <group maxOccurs="*">
                <element name="dc:Language"/>
            </group>
            <group maxOccurs="*" minOccurs="0">
                <element name="dc:Relation"/>
            </group>
            <group maxOccurs="*" minOccurs="0">
                <element name="dc:Rights"/>
            </group>
        </group>
        <group order="seq">
            <element name="dcq:descriptionType"/>
            <element name="rdf:value"/>
        </group>
        <group order="seq">
            <element name="dcq:relationType"/>
            <element name="rdf:value"/>
        </group>
    </group>
    <attribute name="about" type="string" minOccurs="0"/>
</type>
<type name="dc:Title" content="textOnly"/>
<type name="dc:Creator" content="textOnly"/>
<type name="dc:Subject" content="textOnly"/>
<type name="dc:Description" content="mixed">
    <group maxOccurs="*" minOccurs="0">
        <group order="choice">
            <element name="rdf:Description"/>
        </group>
    </group>
</type>
<type name="dc:Publisher" content="textOnly"/>
<type name="dc:Contributor" content="textOnly"/>
<type name="dc:Date" content="textOnly"/>
<type name="dc:Type" content="textOnly"/>
<type name="dc:Format" content="textOnly"/>
```

```
<type name="dc:Identifier" content="textOnly"/>
<type name="dc:Source" content="textOnly"/>
<type name="dc:Language" content="textOnly"/>
<type name="dc:Relation" content="mixed">
    <group maxOccurs="*" minOccurs="0">
        <group order="choice">
            <element name="rdf:Description"/>
        </group>
    </group>
</type>
<type name="dc:Rights" content="textOnly"/>
<type name="dcq:descriptionType" content="textOnly"/>
<type name="dcq:relationType" content="textOnly"/>
<type name="rdf:value" content="textOnly"/>
<type name="ontology-definitions" content="elementOnly">
    <group order="seq">
        <group minOccurs="0" maxOccurs="1">
            <element name="imports"/>
        </group>
        <group minOccurs="0" maxOccurs="1">
            <element name="rule-base"/>
        </group>
        <group maxOccurs="*" minOccurs="0">
            <group order="choice">
                <element name="class-def"/>
                <element name="slot-def"/>
            </group>
        </group>
    </group>
</type>
<type name="imports" content="elementOnly">
    <group maxOccurs="*">
        <element name="URI"/>
    </group>
</type>
<type name="URI" content="textOnly"/>
<type name="rule-base" content="textOnly">
    <attribute name="type" type="string" minOccurs="1"/>
</type>
<type name="AND" content="elementOnly">
    <group order="seq">
        <group order="choice">
            <element name="class"/>
            <element name="slot-constraint"/>
            <element name="AND"/>
            <element name="OR"/>
            <element name="NOT"/>
        </group>
        <group maxOccurs="*">
            <group order="choice">
                <element name="class"/>
                <element name="slot-constraint"/>
                <element name="AND"/>
                <element name="OR"/>
                <element name="NOT"/>
            </group>
        </group>
    </group>
</type>
<type name="OR" content="elementOnly">
    <group order="seq">
```

```
            <group order="choice">
                <element name="class"/>
                <element name="slot-constraint"/>
                <element name="AND"/>
                <element name="OR"/>
                <element name="NOT"/>
            </group>
            <group maxOccurs="*">
                <group order="choice">
                    <element name="class"/>
                    <element name="slot-constraint"/>
                    <element name="AND"/>
                    <element name="OR"/>
                    <element name="NOT"/>
                </group>
            </group>
        </group>
    </type>
    <type name="NOT" content="elementOnly">
        <group order="choice">
            <element name="class"/>
            <element name="slot-constraint"/>
            <element name="AND"/>
            <element name="OR"/>
            <element name="NOT"/>
        </group>
    </type>
    <type name="class-def" content="elementOnly">
        <group order="seq">
            <element name="class"/>
            <group minOccurs="0" maxOccurs="1">
                <element name="documentation"/>
            </group>
            <group minOccurs="0" maxOccurs="1">
                <element name="subclass-of"/>
            </group>
            <group maxOccurs="*" minOccurs="0">
                <element name="slot-constraint"/>
            </group>
        </group>
        <attribute name="type" type="NMTOKEN" minOccurs="0" default="primitive">
            <datatype source="string">
                <enumeration value="primitive|defined"/>
            </datatype>
        </attribute>
    </type>
    <type name="class" content="empty">
        <attribute name="name" type="string" minOccurs="1"/>
    </type>
    <type name="documentation" content="textOnly"/>
    <type name="subclass-of" content="elementOnly">
        <group maxOccurs="*">
            <group order="choice">
                <element name="class"/>
                <element name="slot-constraint"/>
                <element name="AND"/>
                <element name="OR"/>
                <element name="NOT"/>
            </group>
        </group>
    </type>
```

```
<type name="slot-def" content="elementOnly">
    <group order="seq">
        <element name="slot"/>
        <group minOccurs="0" maxOccurs="1">
            <element name="documentation"/>
        </group>
        <group minOccurs="0" maxOccurs="1">
            <element name="subslot-of"/>
        </group>
        <group minOccurs="0" maxOccurs="1">
            <element name="domain"/>
        </group>
        <group minOccurs="0" maxOccurs="1">
            <element name="range"/>
        </group>
        <group minOccurs="0" maxOccurs="1">
            <element name="inverse"/>
        </group>
        <group minOccurs="0" maxOccurs="1">
            <element name="properties"/>
        </group>
    </group>
</type>
<type name="slot" content="empty">
    <attribute name="name" type="string" minOccurs="1"/>
</type>
<type name="subslot-of" content="elementOnly">
    <group maxOccurs="*">
        <element name="slot"/>
    </group>
</type>
<type name="domain" content="elementOnly">
    <group maxOccurs="*">
        <group order="choice">
            <element name="class"/>
            <element name="slot-constraint"/>
            <element name="AND"/>
            <element name="OR"/>
            <element name="NOT"/>
        </group>
    </group>
</type>
<type name="range" content="elementOnly">
    <group maxOccurs="*">
        <group order="choice">
            <element name="class"/>
            <element name="slot-constraint"/>
            <element name="AND"/>
            <element name="OR"/>
            <element name="NOT"/>
        </group>
    </group>
</type>
<type name="inverse" content="elementOnly">
    <element name="slot"/>
</type>
<type name="properties" content="elementOnly">
    <group maxOccurs="*" minOccurs="0">
        <group order="choice">
            <element name="transitive"/>
            <element name="symmetric"/>
```

```
            <element name="other"/>
        </group>
    </group>
</type>
<type name="transitive" content="empty"/>
<type name="symmetric" content="empty"/>
<type name="other" content="textOnly"/>
<type name="slot-constraint" content="elementOnly">
    <group order="seq">
        <element name="slot"/>
        <group maxOccurs="*">
            <group order="choice">
                <element name="has-value"/>
                <element name="value-type"/>
                <element name="cardinality"/>
                <element name="max-cardinality"/>
                <element name="min-cardinality"/>
            </group>
        </group>
    </group>
</type>
<type name="-has-value" content="elementOnly">
    <group maxOccurs="*">
        <group order="choice">
            <element name="class"/>
            <element name="slot-constraint"/>
            <element name="AND"/>
            <element name="OR"/>
            <element name="NOT"/>
        </group>
    </group>
</type>
<type name="value-type" content="elementOnly">
    <group maxOccurs="*">
        <group order="choice">
            <element name="class"/>
            <element name="slot-constraint"/>
            <element name="AND"/>
            <element name="OR"/>
            <element name="NOT"/>
        </group>
    </group>
</type>
<type name="cardinality" content="elementOnly">
    <group maxOccurs="*">
        <group order="seq">
            <element name="number"/>
            <group order="choice">
                <element name="class"/>
                <element name="slot-constraint"/>
                <element name="AND"/>
                <element name="OR"/>
                <element name="NOT"/>
            </group>
        </group>
    </group>
</type>
<type name="max-cardinality" content="elementOnly">
    <group maxOccurs="*">
        <group order="seq">
            <element name="number"/>
```

```
                <group order="choice">
                    <element name="class"/>
                    <element name="slot-constraint"/>
                    <element name="AND"/>
                    <element name="OR"/>
                    <element name="NOT"/>
                </group>
            </group>
        </group>
    </type>
    <type name="min-cardinality" content="elementOnly">
        <group maxOccurs="*">
            <group order="seq">
                <element name="number"/>
                <group order="choice">
                    <element name="class"/>
                    <element name="slot-constraint"/>
                    <element name="AND"/>
                    <element name="OR"/>
                    <element name="NOT"/>
                </group>
            </group>
        </group>
    </type>
    <type name="number" content="textOnly"/>
</schema>
<?xml version="1.0" encoding="UTF-8"?>
```

Appendix A.3 The Example in XML syntax

The following is an example of an OIL ontology that conforms to the above DTD. It is the same ontology that was presented in Section 3.2.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ontology SYSTEM "oil.dtd">
<ontology>
    <ontology-container>
        <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:dc="http://purl.oclc.org/dc#"
            xmlns:dcq = "http://purl.org/dc/qualifiers/1.0/">
            <rdf:Description about = "">
                <dc:Title>African animals</dc:Title>
                <dc:Creator>Ian Horrocks</dc:Creator>
                <dc:Subject>animal, food, vegetarians</dc:Subject>
                <dc:Description>A didactic example ontology describing African animals</dc:Description>
                <dc:Description>
                    <rdf:Description>
                        <dcq:descriptionType>Release</dcq:descriptionType>
                        <rdf:value>1.01</rdf:value>
                    </rdf:Description>
                </dc:Description>
                <dc:Publisher>I. Horrocks</dc:Publisher>
                <dc:Type>ontology</dc:Type>
                <dc:Format>pdf</dc:Format>
                <dc:Indentifier>http://www.cs.vu.nl/~dieter/oil/TR/oil.pdf</dc:Indentifier>
```

```
            <dc:Source>http://www.africa.com/nature/animals.html</dc:Source>
            <dc:Language>OIL</dc:Language>
            <dc:Language>en-uk</dc:Language>
            <dc:Relation>
                <rdf:Description>
                    <dcq:descriptionType>hasPart</dcq:descriptionType>
                     <rdf:value>http://www.ontosRus.com/animals/jungle.onto</rdf:value>
                </rdf:Description>
            </dc:Relation>
        </rdf:Description>
    </rdf:RDF>
</ontology-container>
<ontology-definitions>
     <imports>
         <URI>http://www.ontosRus.com/animals/jungle.onto</URI>
    </imports>
    <slot-def>
        <slot name="eats"/>
        <inverse>
            <slot name="is-eaten-by"/>
        </inverse>
    </slot-def>
    <slot-def>
        <slot name="has-part"/>
        <inverse>
            <slot name="is-part-of"/>
        </inverse>
        <properties>
            <transitive/>
        </properties>
    </slot-def>
    <class-def>
        <class name="animal"/>
    </class-def>
    <class-def>
        <class name="plant"/>
        <subclass-of>
            <NOT>
                <class name="animal"/>
            </NOT>
        </subclass-of>
    </class-def>
    <class-def>
        <class name="tree"/>
        <subclass-of>
            <class name="plant"/>
        </subclass-of>
    </class-def>
    <class-def>
        <class name="branch"/>
        <slot-constraint>
```

```
            <slot name="is-part-of"/>
            <has-value>
                <class name="tree"/>
            </has-value>
        </slot-constraint>
    </class-def>
    <class-def>
    <class name="leaf"/>
    <slot-constraint>
        <slot name="is-part-of"/>
        <has-value>
            <class name="branch"/>
        </has-value>
    </slot-constraint>
    </class-def>
    <class-def type="defined">
        <class name="carnivore"/>
        <subclass-of>
            <class name="animal"/>
        </subclass-of>
        <slot-constraint>
            <slot name="eats"/>
            <value-type>
                <class name="animal"/>
            </value-type>
        </slot-constraint>
    </class-def>
    <class-def type="defined">
        <class name="herbivore"/>
        <subclass-of>
            <class name="animal"/>
            <NOT>
                <class name="carnivore"/>
            </NOT>
        </subclass-of>
        <slot-constraint>
            <slot name="eats"/>
            <value-type>
                <OR>
                    <class name="plant"/>
                    <slot-constraint>
                        <slot name="is-part-of"/>
                        <has-value>
                            <class name="plant"/>
                        </has-value>
                    </slot-constraint>
                </OR>
            </value-type>
        </slot-constraint>
    </class-def>
    <class-def>
```

```
        <class name="giraffe"/>
            <subclass-of>
                <class name="animal"/>
            </subclass-of>
            <slot-constraint>
                <slot name="eats"/>
                <value-type>
                    <class name="leaf"/>
                </value-type>
                </slot-constraint>
        </class-def>
        <class-def>
            <class name="lion"/>
            <subclass-of>
                <class name="animal"/>
            </subclass-of>
            <slot-constraint>
                <slot name="eats"/>
                <value-type>
                    <class name="herbivore"/>
                </value-type>
            </slot-constraint>
        </class-def>
        <class-def>
            <class name="tasty-plant"/>
            <subclass-of>
                <class name="plant"/>
            </subclass-of>
            <slot-constraint>
                <slot name="eaten-by"/>
                    <has-value>
                        <class name="herbivore"/>
                        <class name="carnivore"/>
                    </has-value>
                </slot-constraint>
        </class-def>
    </definitions>
</ontology>
```

Appendix B: OIL Syntax in RDF

This appendix provides an RDF-Schema definition for OIL. RDF relies on Namespaces and Namespace prefixes: RDF-vocabulary is prefixed with a Namespace prefix, which is resolved to a complete URI by an RDF processor. We are using the following namespace prefixes: “oil:” for OIL,“rdf:” and “rdfs:” for RDF and RDF-Schema, “dc:” and “dcq:” for Dublin Core Vocabulary and Qualifiers, respectively. The usual RDF-Dublin Core encoding, which can be obtained at [Dublin Core] is used for Dublin Core. OIL relies heavily on RDF-Schema itself, since OIL is defined as an extension of RDF-Schema and reuses concepts of RDF-Schema as much as possible. This strategy was taken to facilitate the reuse of existing RDF-Schema-based applications and tools. However, certain extensions of RDF-schema were required. For example, OIL allows implicit definitions of classes in the form of boolean operators (AND, OR, NOT) as value of the subclass-of relation, whereas in RDFS the value of the subClassOf statement is always an explicit class. We introduced oil:ClassExpression as a placeholder class for the three boolean operators, which are also modeled as classes, to allow their use as value for the subClassOf statement.

Several extensions in this vein have been made, a full listing can be found in Table 1 and Table 2 and in [Broekstra et al., to appear] a detailed analysis is made of this extension. The resulting RDF Schema for OIL can be found in appendix B.1 and an example ontology in RDFS syntax is presented in appendix B.2.

Table 1: Class-definitions in OIL and the corresponding RDF(S) constructs

OIL primitive	**RDFS syntax**	**type**
class-def	rdfs:Class	class
subclass-of	rdfs:subClassOf	property
class-expression	*oil:ClassExpression* (placeholder only)	class
AND	oil:AND (subclass of ClassExpression)	class
OR	oil:OR (subclass of ClassExpression)	class
NOT	oil:NOT (subclass of ClassExpression)	class
slot-constraint	*oil:SlotConstraint* (placeholder only) oil:hasSlotConstraint (rdf:type of rdfs:ConstraintProperty) *oil:NumberRestriction* (placeholder only) (subclass of oil:SlotConstraint)	class property class

OIL primitive	RDFS syntax	type
has-value	oil:HasValue (subclass of oil:SlotConstraint)	class
value-type	oil:ValueType (subclass of oil:SlotConstraint)	class
max-cardinality	oil:MaxCardinality (subclass of oil:NumberRestriction)	class
min-cardinality	oil:MinCardinality (subclass of oil:NumberRestriction)	class
cardinality	oil:Cardinality (subclass of oil:NumberRestriction)	class

Table 2: Slot definitions in OIL and the corresponding RDFS constructs

OIL primitive	RDFS syntax	type
slot-def	rdf:Property	class
subslot-of	rdfs:subPropertyOf	property
domain	rdfs:domain	property
range	rdfs:range	property
inverse	oil:inverseRelationOf	property
transitive	oil:TransitiveRelation	class
symmetric	oil:SymmetricRelation	class

Appendix B.1 RDF-Schema for OIL

```
<?xml version='1.0'?>

<rdf:RDF   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
           xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

    <rdfs:Class rdf:ID="RuleBase">
        <rdfs:comment>A user-defined rulebase possibly described by an external RDF-Schema</rdfs:comment>
        <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
    </rdfs:Class>

<!-- Begin Class & Properties Expressions Ontology -->

    <rdfs:Class rdf:ID="ClassExpression">
        <rdfs:comment>An ontology class expression</rdfs:comment>
        <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
    </rdfs:Class>
```

```
    <rdf:Description rdf:about="http://www.w3.org/2000/01/rdf-schema#Class">
        <rdfs:comment>An additional statement about rdfs:Class</rdfs:comment>
        <rdfs:subClassOf rdf:resource="#ClassExpression"/>
    </rdf:Description>

    <rdf:Description rdf:about="http://www.w3.org/2000/01/rdf-schema#subClassOf">
        <rdfs:comment>An extension of the range of rdfs:subClassOf.</rdfs:comment>
        <rdfs:range rdf:resource="#ClassExpression"/>
    </rdf:Description>

    <rdfs:Class rdf:ID="SlotConstraint">
        <rdfs:comment>An ontology slot constraint</rdfs:comment>
        <rdfs:subClassOf rdf:resource="#ClassExpression"/>
    </rdfs:Class>

    <rdfs:Class rdf:ID="NumberRestriction">
        <rdfs:comment>A generic number restriction expression.</rdfs:comment>
        <rdfs:subClassOf rdf:resource="#SlotConstraint"/>
    </rdfs:Class>

    <rdfs:Class rdf:ID="ClassType">
        <rdfs:comment> an abstract class of class types </rdfs:comment>
        <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    </rdfs:Class>

    <rdfs:Class rdf:ID="PrimitiveClass">
        <rdfs:comment> The class of primitive classes</rdfs:comment>
        <rdfs:subClassOf rdf:resource="#ClassType"/>
    </rdfs:Class>

    <rdfs:Class rdf:ID="DefinedClass">
        <rdfs:comment>The class of defined classes</rdfs:comment>
        <rdfs:subClassOf rdf:resource="#ClassType"/>
    </rdfs:Class>

<!-- End Class & Properties Expressions Ontology -->

<!-- Begin Helper Properties -->

    <rdf:Property rdf:ID="hasClass">
        <rdfs:comment>A property connection between a slot constraint and a class expressions</rdfs:comment>
        <rdfs:domain rdf:resource="#SlotConstraint"/>
        <rdfs:range rdf:resource="#ClassExpression"/>
    </rdf:Property>

    <rdf:Property rdf:ID="hasSlotConstraint">
        <rdfs:comment>A property connection between a class definition and a slot constraint </rdfs:comment>
        <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#ConstraintProperty"/>
        <rdfs:domain rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
        <rdfs:range rdf:resource="#SlotConstraint"/>
    </rdf:Property>

    <rdf:Property rdf:ID="hasOperand">
        <rdfs:comment>A property connection between an operator class expression and
            an operand class expression</rdfs:comment>
        <rdfs:domain rdf:resource="#AND"/>
        <rdfs:domain rdf:resource="#OR"/>
        <rdfs:domain rdf:resource="#NOT"/>
        <rdfs:range rdf:resource="#ClassExpression"/>
    </rdf:Property>

    <rdf:Property rdf:ID="hasProperty">
        <rdfs:comment>A property connection between a class expression and a slot expression</rdfs:comment>
        <rdfs:domain rdf:resource="#SlotConstraint"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
    </rdf:Property>

    <rdf:Property rdf:ID="number">
        <rdfs:comment>A property connection between a class expression and a cardinality (integer)</rdfs:comment>
        <rdfs:domain rdf:resource="#NumberRestriction"/>
        <rdfs:domain rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
```

```
        <rdfs:range rdf:resource="http://www.w3c.org/xml/xmlschema#integer"/>
    </rdf:Property>

<!-- End Helper Properties -->

<!-- Begin Class Expressions -->

    <rdfs:Class rdf:ID="AND">
        <rdfs:comment>An expression corresponding to the conjunction of (two) class expressions</rdfs:comment>
        <rdfs:subClassOf rdf:resource="#ClassExpression"/>
    </rdfs:Class>

    <rdfs:Class rdf:ID="OR">
        <rdfs:comment>An expression corresponding to the disjunction of (two) class expressions</rdfs:comment>
        <rdfs:subClassOf rdf:resource="#ClassExpression"/>
    </rdfs:Class>

    <rdfs:Class rdf:ID="NOT">
        <rdfs:comment>An expression corresponding to the negation of a class expression</rdfs:comment>
        <rdfs:subClassOf rdf:resource="#ClassExpression"/>
    </rdfs:Class>

    <rdfs:Class rdf:ID="HasValue">
        <rdfs:comment>An expression corresponding to an existential slot constraint</rdfs:comment>
        <rdfs:subClassOf rdf:resource="#SlotConstraint"/>
    </rdfs:Class>

    <rdfs:Class rdf:ID="ValueType">
        <rdfs:comment>An expression corresponding to a universally quantified value restriction</rdfs:comment>
        <rdfs:subClassOf rdf:resource="#SlotConstraint"/>
        </rdfs:Class>

    <rdfs:Class rdf:ID="MaxCardinality">
        <rdfs:comment>An ontology property expression corresponding to a (qualified) number restriction.</rdfs:comment>
        <rdfs:subClassOf rdf:resource="#NumberRestriction"/>
    </rdfs:Class>

    <rdfs:Class rdf:ID="MinCardinality">
        <rdfs:comment>An ontology property expression corresponding to a (qualified) number restriction.</rdfs:comment>
        <rdfs:subClassOf rdf:resource="#NumberRestriction"/>
    </rdfs:Class>

    <rdfs:Class rdf:ID="Cardinality">
        <rdfs:comment>An ontology property expression corresponding to a (qualified) number restriction.</rdfs:comment>
        <rdfs:subClassOf rdf:resource="#NumberRestriction"/>
    </rdfs:Class>

<!-- End Class Expressions -->

<!--Begin Property Qualities-->

    <rdf:Property rdf:ID="inverseRelationOf">
        <rdfs:comment>A property connection between a property and the inverse property</rdfs:comment>
        <rdfs:domain rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
    </rdf:Property>

    <rdfs:Class rdf:ID="TransitiveProperty">
        <rdfs:comment>The class of all transitive relations.</rdfs:comment>
        <rdfs:subClassOf rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
    </rdfs:Class>

    <rdfs:Class rdf:ID="SymmetricProperty">
        <rdfs:comment>The class of all symmetric relations.</rdfs:comment>
        <rdfs:subClassOf rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
    </rdfs:Class>

<!--End Property Qualities-->

</rdf:RDF>
```

Appendix B.2 The Example in RDF syntax

```
<?xml version="1.0"?>

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:rdfs="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#"
      xmlns:oil="http://www.ontoknowledge.org/oil/rdf-schema"
      xmlns:dc="http://purl.org/dc/elements/1.1/"
      xmlns:dcq="http://purl.org/dc/qualifiers/1.1/">



    <rdf:Description about="">
        <dc:Title>African Animals</dc:Title>
        <dc:Creator>Ian Horrocks</dc:Creator>
        <dc:subject>animal, food, vegetarians</dc:subject>
        <dc:Description>A didactic example ontology describing African animals</dc:Description>
        <dcq:description.release>1.04</dcq:description.release>
        <dc:publisher>I. Horrocks</dc:publisher>
        <dc:type>ontology</dc:type>
        <dc:format>rdf</dc:format>
        <dc:identifier>http://www.ontoknowledge.org/oil/animals.rdfs</dc:identifier>
        <dc:source>http://www.africa.com/nature/animals.html</dc:source>
        <dc:language>OIL</dc:language>
        <dc:language>en-uk</dc:language>
    </rdf:Description>



    <rdf:Description xmlns:syllogism="http://old.greece/syllogism/">
        <rdf:type resource="http://www.ontoknowledge.org/oil//rdfschema#RuleBase"/>
        <syllogism:premise>if it rains, you get wet</syllogism:premise>
        <syllogism:fact>it rains</syllogism:fact>
        <syllogism:conclusion>you get wet</syllogism:conclusion>
    </rdf:Description>



    <rdf:Property rdf:ID="eats">
        <oil:inverseRelationOf rdf:resource="#is-eaten-by"/>
    </rdf:Property>

    <rdf:Property rdf:ID="is-eaten-by"/>

    <rdf:Property rdf:ID="has-part">
        <oil:inverseRelationOf rdf:resource="#is-part-of"/>
    </rdf:Property>

    <rdf:Property rdf:ID="is-part-of"/>

    <rdfs:Class rdf:ID="animal"/>

    <rdfs:Class rdf:ID="plant">
        <rdfs:subClassOf>
            <oil:NOT>
                <oil:hasOperand rdf:resource="#animal"/>
            </oil:NOT>
        </rdfs:subClassOf>
    </rdfs:Class>

    <rdfs:Class rdf:ID="tree">
        <rdfs:subClassOf rdf:resource="#plant"/>
    </rdfs:Class>

    <rdfs:Class rdf:ID="branch">
        <oil:hasSlotConstraint>
            <oil:HasValue>
                <oil:hasProperty rdf:resource="#is-part-of"/>
                <oil:hasClass rdf:resource="#tree"/>
            </oil:HasValue>
```

```
    </oil:hasSlotConstraint>
</rdfs:Class>

<rdfs:Class rdf:ID="leaf">
    <oil:hasSlotConstraint>
        <oil:HasValue>
            <oil:hasProperty rdf:resource="#is-part-of"/>
            <oil:hasClass rdf:resource="#branch"/>
        </oil:HasValue>
    </oil:hasSlotConstraint>
</rdfs:Class>

<rdfs:Class rdf:ID="carnivore">
    <rdfs:subClassOf rdf:resource="#animal"/>
    <oil:hasSlotConstraint>
        <oil:ValueType>
            <oil:hasProperty rdf:resource="#eats"/>
            <oil:hasClass rdf:resource="#animal"/>
        </oil:ValueType>
    </oil:hasSlotConstraint>
</rdfs:Class>

<rdfs:Class rdf:ID="herbivore">
    <rdf:type rdf:resource="http://www.ontoknowledge.org/oil/rdf-schema/#DefinedClass"/>
    <rdfs:subClassOf rdf:resource="#animal"/>
    <rdfs:subClassOf>
        <oil:NOT>
            <oil:hasOperand rdf:resource="#carnivore"/>
        </oil:NOT>
    </rdfs:subClassOf>
    <oil:hasSlotConstraint>
        <oil:ValueType>
            <oil:hasProperty rdf:resource="#eats"/>
            <oil:hasClass>
                <oil:OR>
                    <oil:hasOperand rdf:resource="#plant"/>
                    <oil:hasOperand>
                        <oil:HasValue>
                            <oil:hasProperty rdf:resource="#is-part-of"/>
                            <oil:hasClass rdf:resource="#plant"/>
                        </oil:HasValue>
                    </oil:hasOperand>
                </oil:OR>
            </oil:hasClass>
        </oil:ValueType>
    </oil:hasSlotConstraint>
</rdfs:Class>

<rdfs:Class rdf:ID="giraffe">
    <rdfs:subClassOf rdf:resouce="#herbivore"/>
    <oil:hasSlotConstraint>
        <oil:ValueType>
            <oil:hasProperty rdf:resource="#eats"/>
            <oil:hasClass rdf:resource="#leaf"/>
        </oil:ValueType>
    </oil:hasSlotConstraint>
</rdfs:Class>

<rdfs:Class rdf:ID="lion">
    <rdfs:subClassOf rdf:resource="#animal"/>
    <oil:hasSlotConstraint>
        <oil:ValueType>
            <oil:hasProperty rdf:resource="#eats"/>
            <oil:hasClass rdf:resource="#herbivore"/>
        </oil:ValueType>
    </oil:hasSlotConstraint>
</rdfs:Class>

<rdfs:Class rdf:ID="tasty-plant">
    <rdfs:subClassOf rdf:resource="#plant"/>
    <oil:hasSlotConstraint>
        <oil:HasValue>
            <oil:hasProperty rdf:resource="#eaten-by"/>
```

```
                    <oil:hasClass>
                        <oil:AND>
                            <oil:hasOperand rdf:resource="#herbivore"/>
                            <oil:hasOperand rdf:resource="#carnivore"/>
                        </oil:AND>
                    </oil:hasClass>
                </oil:HasValue>
            </oil:hasSlotConstraint>
        </rdfs:Class>

    </rdf:RDF>
```

Appendix C: First order semantics of OIL

In this section we will give a formal specification and semantics for OIL as well as for the common inference problems (class consistency and inferred subclass relations) performed with respect to an OIL ontology. We will only consider the **definitions** part of the ontology and we will ignore fields such as **documentation** that have no semantic significance.

The semantics of OIL rely on a translation into the $\mathcal{SHIQ}$ description logic. $\mathcal{SHIQ}$ has a highly expressive concept language that is able to fully capture the OIL core language, and we will define a satisfiability preserving translation $\sigma(\cdot)$ that maps OIL ontologies into $\mathcal{SHIQ}$ terminologies. This has the added benefit that an existing $\mathcal{SHIQ}$ reasoner implemented in the FaCT system can be used to reason with OIL ontologies.

The translation is quite straightforward and follows directly from the informal specification given in Section 3.1. An OIL ontology $\mathcal{O}$ consists of a list $d_1, \ldots, d_n$, where each d_i is either a class definition or a slot definition. This list of definitions is translated into a $\mathcal{SHIQ}$ terminology $\mathcal{T}$ (a set of axioms) as follows:

$$\sigma(d_1, \ldots, d_n) = \bigcup_{i=1,\ldots,n} \sigma(d_i)$$

A class definition is either a pair $\langle \mathsf{CN}, D \rangle$ or a triple $\langle \mathsf{CN}, P, D \rangle$, where CN is a class name, D is a class description and P is either **primitive** or **defined**; $\langle \mathsf{CN}, D \rangle$ is equivalent to $\langle \mathsf{CN}, \textbf{primitive}, D \rangle$. A class definition $\langle \mathsf{CN}, \textbf{primitive}, D \rangle$ is written $\mathsf{CN} \sqsubseteq D$ (it states that CN is a subclass of the class described by D) and a class definition $\langle \mathsf{CN}, \textbf{defined}, D \rangle$ is written $\mathsf{CN} \doteq D$ (it states that CN is equivalent to the class described by D).

A class description D consists of an optional **subclass-of** component, itself a list of one or more **class-expression**s $C_1, \ldots, C_n$, followed by a list of zero or more **slot-constraints** $A_1, \ldots, A_m$. We will write such a class description as

$$[C_1, \ldots, C_n, A_1, \ldots, A_m].$$

A **class-expression** is either a class name CN, a **slot-constraint**, a conjunction of class expressions, written $C_1 \sqcap \ldots \sqcap C_n$, a disjunction of class expressions, written $C_1 \sqcup \ldots \sqcup C_n$ or a negated class expression, written $\neg C$. A **slot-constraint** consists of a slot name SN followed by one or more constraints that apply to the slot, written $\mathsf{SN}[a_1, \ldots, a_n]$. Each constraint can be either:

- A **value** constraint with a list of one or more class-expressions, written $\exists C_1, \ldots, C_n$.
- A **value-type** constraint with a list of one or more class-expressions, written $\forall C_1, \ldots, C_n$.
- A **max-cardinality** constraint with a non-negative integer n followed (optionally) by a class expression C, written $\leqslant n, C$ ($\leqslant n, \top$ if the class expression is omitted).
- A **min-cardinality** constraint with a non-negative integer n followed (optionally) by a class expression C, written $\geqslant n, C$ ($\geqslant n, \top$ if the class expression is omitted).
- A **cardinality** constraint with a non-negative integer n followed (optionally) by a class expression C, written $=n, C$ ($=n, \top$ if the class expression is omitted).

In order to maintain the decidability of the language, cardinality constraints can only be applied to *simple* slots. A simple slot is one that is neither transitive nor has any transitive subslots. However, as the transitivity of a slot can be inferred (e.g., from the fact that the inverse of the slot is a transitive

$$
\begin{array}{rcl}
\sigma(\mathsf{CN} \sqsubseteq D) & = & \{\sigma(\mathsf{CN}) \sqsubseteq \sigma(D)\} \\
\sigma(\mathsf{CN} \doteq D) & = & \{\sigma(\mathsf{CN}) \sqsubseteq \sigma(D), \sigma(D) \sqsubseteq \sigma(\mathsf{CN})\} \\
\sigma([C_1, \ldots, C_n, A_1, \ldots, A_m]) & = & \top \sqcap \sigma(C_1) \sqcap \ldots \sqcap \sigma(C_n) \sqcap \sigma(A_1) \sqcap \ldots \sqcap \sigma(A_m) \\
\sigma(\mathsf{CN}) & = & \mathsf{CN} \\
\sigma(\top) & = & \top \\
\sigma(C_1 \sqcap \ldots \sqcap C_n) & = & \sigma(C_1) \sqcap \ldots \sqcap \sigma(C_n) \\
\sigma(C_1 \sqcup \ldots \sqcup C_n) & = & \sigma(C_1) \sqcup \ldots \sqcup \sigma(C_n) \\
\sigma(\neg C) & = & \neg\sigma(C) \\
\sigma(\mathsf{SN}[a_1, \ldots, a_n]) & = & \sigma(\mathsf{SN}(a_1)) \sqcap \ldots \sqcap \sigma(\mathsf{SN}(a_n)) \\
\sigma(\mathsf{SN}(\exists C_1, \ldots, C_n)) & = & \exists \mathsf{SN}.\sigma(C_1) \sqcap \ldots \sqcap \exists \mathsf{SN}.\sigma(C_n) \\
\sigma(\mathsf{SN}(\forall C_1, \ldots, C_n)) & = & \forall \mathsf{SN}.\sigma(C_1) \sqcap \ldots \sqcap \forall \mathsf{SN}.\sigma(C_n) \\
\sigma(\mathsf{SN}(\leqslant n, C)) & = & \leqslant n \mathsf{SN}.\sigma(C) \\
\sigma(\mathsf{SN}(\geqslant n, C)) & = & \geqslant n \mathsf{SN}.\sigma(C) \\
\sigma(\mathsf{SN}(= n, C)) & = & \leqslant n \mathsf{SN}.\sigma(C) \sqcap \geqslant n \mathsf{SN}.\sigma(C)
\end{array}
$$

Figure C-1: Translation of OIL class definitions into $\mathcal{SHIQ}$

slot), simple slot is defined in terms of the translation into $\mathcal{SHIQ}$: a slot SN in an ontology $\mathcal{O}$ is a simple slot iff $\sigma(\mathsf{SN})$ is a simple role in the $\mathcal{SHIQ}$ terminology $\sigma(\mathcal{O})$.

We can now define how the function $\sigma(\cdot)$ maps an OIL class definition into a set of $\mathcal{SHIQ}$ axioms. The definition is given in Figure C-1, where CN is a class name (or a $\mathcal{SHIQ}$ concept name), SN is a slot name (or $\mathcal{SHIQ}$ role name), D is a class description, C (possibly subscripted) is a class expression, A (possibly subscripted) is a slot constraint, a_i is a constraint (on a slot) and n is a non-negative integer.

A slot definition is a pair $\langle \mathsf{SN}, D \rangle$, where SN is a slot name and D is a slot description. A slot description D consists of an optional **subslot-of** component, itself a list of one or more slot names $\mathsf{RN}_1, \ldots, \mathsf{RN}_n$, followed by a list of zero or more global slot constraints (e.g., **inverse**) $S_1, \ldots, S_m$. We will write such a slot definition as:

$$\mathsf{SN}[\mathsf{RN}_1, \ldots, \mathsf{RN}_n, S_1, \ldots, S_m]$$

Each global constraint S_i on SN can be either:

- A **domain** constraint with a list of one or more class-expressions, written $\downarrow [C_1, \ldots, C_n]$.
- A **range** constraint with a list of one or more class-expressions, written $\uparrow [C_1, \ldots, C_n]$.
- An **inverse** constraint with a slot name RN, written $^{-}\mathsf{RN}$.
- A **properties** constraint with a list of one or more properties, written $[P_1, \ldots, P_n]$. Valid properties are **transitive**, written $+$ and **symmetrical**, written $\leftrightarrow$.

We can now define how the function $\sigma(\cdot)$ maps an OIL slot definition into a set of $\mathcal{SHIQ}$ axioms. The definition is given in Figure C-2, where RN and SN are slot names (or $\mathcal{SHIQ}$ role names), C_i is a class expression, S_i is a global slot constraint and P_i is a property.

The meaning of a $\mathcal{SHIQ}$ terminology, and of the common inference problems, is given in terms of a Tarski style model theoretic semantics using *interpretations*. An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$

$$
\begin{array}{rcl}
\sigma(\mathsf{SN}[\mathsf{RN}_1,\ldots,\mathsf{RN}_n,S_1,\ldots,S_m]) & = & \sigma(\mathsf{SN}[\mathsf{RN}_1,\ldots,\mathsf{RN}_n] \cup \sigma(\mathsf{SN}[S_1,\ldots,S_m]) \\
\sigma(\mathsf{SN}[\mathsf{RN}_1,\ldots,\mathsf{RN}_n] & = & \bigcup_{i=1,\ldots,n} \sigma(\mathsf{SN} \sqsubseteq \mathsf{RN}_i) \\
\sigma(\mathsf{SN}[S_1,\ldots,S_m]) & = & \bigcup_{i=1,\ldots,m} \sigma(\mathsf{SN}(S_i)) \\
\sigma(\mathsf{SN} \sqsubseteq \mathsf{RN}) & = & \{\mathsf{SN} \sqsubseteq \mathsf{RN}\} \\
\sigma(\mathsf{SN}(\downarrow[C_1,\ldots,C_n])) & = & \bigcup_{i=1,\ldots,n} \{\exists \mathsf{SN}.\top \sqsubseteq \sigma(C_i)\} \\
\sigma(\mathsf{SN}(\uparrow[C_1,\ldots,C_n])) & = & \bigcup_{i=1,\ldots,n} \{\top \sqsubseteq \forall \mathsf{SN}.\sigma(C_i)\} \\
\sigma(\mathsf{SN}(^{-}\mathsf{RN})) & = & \{\mathsf{SN}^{-} \sqsubseteq \mathsf{RN}, \mathsf{RN} \sqsubseteq \mathsf{SN}^{-}\} \\
\sigma(\mathsf{SN}([P_1,\ldots,P_n])) & = & \bigcup_{i=1,\ldots,n} \{\sigma(\mathsf{SN}(P_i))\} \\
\sigma(\mathsf{SN}(+)) & = & \{\mathsf{SN} \in \mathbf{S}_+\} \\
\sigma(\mathsf{SN}(\leftrightarrow)) & = & \{\mathsf{SN}^{-} \sqsubseteq \mathsf{SN}, \mathsf{SN} \sqsubseteq \mathsf{SN}^{-}\}
\end{array}
$$

Figure C-2: Translation of OIL slot definitions into $\mathcal{SHIQ}$

consists of a set $\Delta^{\mathcal{I}}$, called the *domain* of $\mathcal{I}$, and a *valuation* $\cdot^{\mathcal{I}}$ which maps every concept to a subset of $\Delta^{\mathcal{I}}$ and every role to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ such that, for all concepts C, D, roles R, S, and non-negative integers n, the following equations are satisfied, where $\sharp M$ denotes the cardinality of a set M:

$$
\begin{array}{rcll}
(R^{-})^{\mathcal{I}} & = & \{\langle x,y\rangle \mid \langle y,x\rangle \in R^{\mathcal{I}}\} & \text{(inverse roles)} \\
(C \sqcap D)^{\mathcal{I}} & = & C^{\mathcal{I}} \cap D^{\mathcal{I}} & \text{(conjunction)} \\
(C \sqcup D)^{\mathcal{I}} & = & C^{\mathcal{I}} \cup D^{\mathcal{I}} & \text{(disjunction)} \\
(\neg C)^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} & \text{(negation)} \\
(\exists R.C)^{\mathcal{I}} & = & \{x \mid \exists y.\langle x,y\rangle \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} & \text{(value constraint)} \\
(\forall R.C)^{\mathcal{I}} & = & \{x \mid \forall y.\langle x,y\rangle \in R^{\mathcal{I}} \text{ implies } y \in C^{\mathcal{I}}\} & \text{(value-type constraint)} \\
(\geqslant nR.C)^{\mathcal{I}} & = & \{x \mid \sharp\{y.\langle x,y\rangle \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \geqslant n\} & \text{(minimum cardinality)} \\
(\leqslant nR.C)^{\mathcal{I}} & = & \{x \mid \sharp\{y.\langle x,y\rangle \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \leqslant n\} & \text{(maximum cardinality)}
\end{array}
$$

In order to avoid considering roles such as R^{--} (i.e., the inverse of an inverse) we will define a function Inv such that $Inv(R)$ is R^{-} and $Inv(R^{-})$ is R. A role R is *directly subsumed* by a role S w.r.t. a terminology $\mathcal{T}$ iff either $\{R \sqsubseteq S\} \subseteq \mathcal{T}$ or $\{Inv(R) \sqsubseteq Inv(S)\} \subseteq \mathcal{T}$. A role R is *subsumed* by a role S w.r.t. $\mathcal{T}$ (written $\mathcal{T} \models R \sqsubseteq S$) iff R is directly subsumed by a S or there is a role S' such that R is directly subsumed by a S' and $\mathcal{T} \models S' \sqsubseteq S$. A role R is *equivalent* to a role S w.r.t. $\mathcal{T}$ (written $\mathcal{T} \models R \doteq S$) iff $\mathcal{T} \models R \sqsubseteq S$ and $\mathcal{T} \models S \sqsubseteq R$. A role R is transitive in $\mathcal{T}$ iff $\{S \in \mathbf{S}_+\} \subseteq \mathcal{T}$ for some role S such that $R \doteq S$ or $Inv(R) \doteq S$ (this defines $\mathbf{S}_+$, the set of transitive role names). A role R is a *simple* role in $\mathcal{T}$ iff there is no role S such that S is transitive in $\mathcal{T}$ and $\mathcal{T} \models S \sqsubseteq R$.

An interpretation $\mathcal{I}$ *satisfies* a $\mathcal{SHIQ}$ terminology $\mathcal{T}$ iff for every axiom $R \sqsubseteq S$ in $\mathcal{T}$, $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$, for every axiom $C \sqsubseteq D$ in $\mathcal{T}$, $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ and for every transitive role S in $\mathcal{T}$, $S^{\mathcal{I}} = (S^{\mathcal{I}})^{+}$. Such an interpretation is called a *model* of $\mathcal{T}$ (written $\mathcal{I} \models \mathcal{T}$).

A concept C is satisfiable with respect to a $\mathcal{SHIQ}$ terminology $\mathcal{T}$ (written $\mathcal{T} \models C \neq \bot$) iff there a model $\mathcal{I}$ of $\mathcal{T}$ with $C^{\mathcal{I}} \neq \emptyset$. A concept C is subsumed by a concept D w.r.t. $\mathcal{T}$ (written $\mathcal{T} \models C \sqsubseteq D$) $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds for each model $\mathcal{I}$ of $\mathcal{T}$.

An OIL ontology $\mathcal{O}$ is called *consistent* iff $\sigma(\mathcal{O}) \models \top \neq \bot$. A class CN in an ontology $\mathcal{O}$ is called *consistent* iff $\sigma(\mathcal{O}) \models \sigma(\mathsf{CN}) \neq \bot$. A class CN is a *subclass* of a class DN in an ontology $\mathcal{O}$ iff $\sigma(\mathcal{O}) \models \sigma(\mathsf{CN}) \sqsubseteq \sigma(\mathsf{DN})$.