The relation between ontologies and XML schemata

Michel Klein[1], **Dieter Fensel**[1], **Frank van Harmelen**[1,2], and **Ian Horrocks**[3]

Abstract. Currently computers are changing from single isolated devices to entry points into a worldwide network of information exchange and business transactions called the World Wide Web (WWW). Therefore support in the exchange of data, information, and knowledge is becoming the key issue in current computer technology. Ontologies provide a shared and common understanding of a domain that can be communicated between people and application systems. Therefore, they may play a major role in supporting information exchange processes in various areas. However, in order to develop their full power, the representation languages for ontologies must be applicable to existing data exchange standards in the World Wide Web. Therefore, we will compare ontology representation languages with document structure techniques (schemata) on the web. We will do this by giving a detailed comparison of OIL, a proposal for expressing ontologies in the Web, with XML Schema, a proposed standard for describing the structure and semantics of Web documents. We will argue that these two refer to different levels of abstraction and illustrate this claim by providing an translation procedure from the first to the latter.

1 INTRODUCTION

For the past few years, information on the the World Wide Web was mainly intended for direct human consumption. However, to facilitate new intelligent applications such as meaning-based search and information brokering, the semantics of the data on the internet should be accessible for machines. Therefore, methods and tools to create such a "semantic web" have generated wide interest. An important basis for many developments in this area is the Resource Description Framework [1], a standard from the W3C for representing metadata on the web, and its accompanying schema language RDFS. RDFS provides some modelling primitives which can be used to define a vocabulary for a specific domain.

However, although the general aim of this paper is also about adding semantics to online resources, we will not look at RDF, but take an orthogonal view and consider the relation between ontologies and the structure and markup of documents. RDF is mainly intended for describing explicit *metadata* about webresources, but does not give semantics to the actual *markup* of a document (i.e. the tags and their stucture). Therefore, RDF does not answer the question how the structure of documents is related to conceptual terms.

The purpose of this paper is to investigate how ontologies are related to document structure prescriptions, i.e. XML schemata. We will do this by a close comparison of the ontology representation language OIL, a recent proposal for expressing ontologies in the Web, with XML Schema, a proposed standard for describing the structure and semantics of Web documents. We will explain why we think that it is a mistake to *directly* relate ontology languages and XML Schema. Instead we will show that the relationship between ontologies and schema definitions is a modern counterpart of the relationship between (Extended) Entity Relationship Models (cf. [2]) and relational schemata.[4] That is, they refer to different abstraction levels on how to describe information and therefore also to different states in the process of developing on-line information sources.

To illustrate this statement, we will provide a translation procedure from an ontology to an XML structure prescription. As a result of this procedure, a document schema is created, which is founded in a domain ontology. This schema in its turn can be used to validate document markup, finally providing us with well-founded semantic annotation of actual data.

This paper is organized as follows. In the next section, we give an abstract introduction to ontologies, schemata and their relationship. In Section 3 we provide a short introduction to OIL and Section 4 does the same for XML Schema. Central to the paper is Section 5, where we compare both approaches and provide the translation procedure. Section 6 contains a discussion and in Section 7 we present our conclusions.

2 ONTOLOGIES AND SCHEMATA

Ontology, which has been a field of philosophy since Aristotle, has become a buzz-word in information and knowledge-based systems research [4]. Various publications in knowledge engineering, natural language processing, cooperative information systems, intelligent information integration, and knowledge management report about the application of ontologies in developing and using systems. In general, ontologies provide a *shared and common* understanding of a domain that can be communicated between people and heterogeneous and distributed application systems. They have been developed in Artificial Intelligence to facilitate knowledge sharing and reuse.

Database schema have been developed in computer science to

[1] Department of Computer Science, Vrije Universiteit Amsterdam, De Boelelaan 1081a, 1081 HV Amsterdam, the Netherlands, {mcaklein|dieter|frankh}@cs.vu.nl
[2] AIdministrator, Amersfoort, the Netherlands
[3] Department of Computer Science, University of Manchester, Oxford Road, Manchester, M13 9PL, UK, horrocks@cs.man.ac.uk

[4] If you are not familiar with database concepts, the distinction between the symbol and the knowledge levels of Newel is a good analogy (cf. [3]).

describe the structure and semantics of data. A well-known example is the relational database schema that has become the basis for most of the currently used databases [2]. A database schema defines a set of relations and certain integrity constraints. A central assumption is the atomicity of the elements that are in certain relationships (i.e., first normal form). In a nutshell, an information source (or, more precisely, a data source) is viewed as a set of tables. However, many new information sources now exist that do not fit into such rigid schemata. In particular, the WWW has made predominantly document-centered information based on natural language text available. Therefore, new schema languages have arisen that better fit the needs of richer data models. Basically, they integrate schemata for describing documents (like HTML or SGML) with schemata designed for describing data. A prominent approach for a new standard for defining schema of rich and semistructured data sources is XML Schema (see [5], [6] and [7]). XML Schema is a means for defining constraints on well formed XML documents. It provides basic vocabulary and predefined structuring mechanisms for providing information in XML. XML seems to be becoming the pre-dominant standard for exchanging information via the WWW, which is currently becoming the most important way for the on-line dissemination of information. In consequence, comparing ontologies languages and XML schema languages is a timely issue, as both approaches aim, to an extent, at the same goal.

And their relationship? Ontologies applied to on-line information source may be seen as explicit conceptualizations (i.e., *meta information*) that describe the semantics of the data. Fensel [8] points out the following differences between ontologies and schema definitions:

- A language for defining ontologies is syntactically and semantically richer than common approaches for databases.
- The information that is described by an ontology consists of semi-structured natural language texts and not tabular information.
- 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.

However, these statements need to be formulated more precisely when comparing ontology languages with XML schema languages and the purpose of ontologies with the purpose of schemata. This will be done in the next sections.

3 OIL

Horrocks et al. [9] defines the *Ontology Interface Layer (OIL)*. In this section we will only give a brief description of the OIL language. More detailed descriptions can be found elsewhere: a comparison of OIL with other ontology languages and a description of its situation between other web languages can be found in [9] and [10]. In [11], OIL is compared to RDF Schema and defined as an extension of it.

A brief example ontology in OIL is provided in Figure 1; the example is based on the country pages of the CIA World Factbook[1], which we will use as an example throughout this paper. The OIL language has been designed so that: (1) it provides most of the modeling primitives commonly used in frame-based and Description Logic (DL) oriented ontologies; (2) it has a simple, clean, and well defined semantics; (3) 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 by sets of additional primitives, with the proviso that full reasoning support may not be available for ontologies using such primitives.

An ontology in OIL is represented by an *ontology container* and an *ontology definition*. 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.

For the **ontology container** part of OIL, we adopt the components as defined by Dublin Core Metadata Element Set, Version 1.1[2].

Apart from the container, an OIL ontology consists of a set of **ontology definitions**:

- **import** A list of references to other OIL modules that are to be included in this ontology. XML Schema and OIL provide the same (limited) means for composing specifications. Specifications can be included and the underlying assumptions is that names of different specifications are different (via different prefixes).
- **rule-base** A list of rules (sometimes called axioms or global constraints) that apply to the ontology. At present, the structure of these rules is not defined (they could be horn clauses, DL style axioms, etc.), and they have no formal semantics in OIL, but this could be added in the future (see [9]). The rule base consists simply of a **type** (a string) followed by the unstructured rules (a string).
- **class and slot definitions** Zero or more class definitions **(class-def**) and slot definitions **(slot-def**), the structure of which will be described below.

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

- **type** The type of definition. This can be either **primitive** or **defined**; if omitted, the type defaults to primitive. When a class is primitive, its definition (i.e., the combination of the following subclass-of and slot-constraint components) is taken to be a necessary but not sufficient condition for membership of the class.
- **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 subclass of each of the class expressions in the list.
- **slot-constraints** Zero or more slot-constraints, the structure of which will be described below. The class being defined in this class-def must be a subclass of each of the slot-constraints in the list (note that a slot-constraint defines a class).

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

[1] http://www.odci.gov/cia/publications/factbook/
[2] http://purl.org/DC/

defined, so that arbitrarily complex expressions can be formed.

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

- **name** A slot name (a string). The slot is a binary relation that may or may not be defined in the ontology. If it is not defined it is assumed to be a binary relation with no globally applicable constraints, i.e., any pair of individuals could be an instance of the slot.
- **has-value** A list of one or more **class-expressions**. Every instance of the class defined by the slot constraint must be related via the slot relation to an instance of each **class-expression** in the list. For example, the **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.
- **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.
- **max-cardinality** A non-negative integer n followed by a class-expression. An instance of the class defined by the slot-constraint can be related to at most n distinct instances of the class-expression via the slot relation.
- **min-cardinality** and, as a shortcut, **cardinality**.

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 main components:

- **subslot-of** A list of one or more slots. The slot being defined in this slot-def must be a subslot of each of the slots in the list. For example,

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

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

ontology-container
title CIA World Fact Book ontology
creator Michel Klein
subject country information, CIA, world factbook
description A didactic example ontology describing country information
description.release 1.02
publisher CIA
type ontology
format pseudo-xml
identifier http://www.ontoknowledge.org/oil/wfb.xml
source http://www.odci.gov/cia/publications/factbook/
language OIL
language en-uk

ontology-definitions
slot-def *capital*
domain Country
range City
inverse *capital_of*

slot-def *has_boundary*
domain Country
range LandBoundary

slot-def *coastline*
domain Geographical_Location
range KilometerLength
range MilesLength

slot-def *relative_area*
domain Geographical_Location
range AreaComparison

class-def Geographical_Location
slot-constraint *name*
value-type STRING
class-def City
subclass-of Geographical_Location
slot-constraint *located_in*
value-type Country
class-def Country
subclass-of Geographical_Location

class-def LandBoundary
slot-constraint *neighbor_country*
cardinality 1 Country
slot-constraint *length*
value-type
(OR KilometerLength, MilesLength)

class-def KilometerLength
slot-constraint *value*
value-type INTEGER
slot-constraint *unit*
has-value km
class-def MilesLength
slot-constraint *value*
value-type INTEGER
slot-constraint *unit*
has-value mile

class-def AreaComparison
slot-constraint *compared_to*
value-type Geographical_Location
slot-constraint *proportion*
value-type STRING

Figure 1. An partial ontology in OIL

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/1999/XMLSchema">

    <complexType name="address">
        <element name="name" minOccurs="1" maxOccurs="1">
            <complexType base="string" content="mixed"/>
        </element>
        <element name="street" minOccurs="1" maxOccurs="2" type="string"/>
        <element name="city" minOccurs="1" maxOccurs="1" type="string"/>
        <element name="state" minOccurs="1" maxOccurs="1" type="string"/>
        <element ref="zip" minOccurs="1" maxOccurs="1"/>
        <element name="country" minOccurs="0" maxOccurs="1" type="string"/>
    </complexType>

    <element name="zip" type="zipCode"/>

    <simpleType name="zipCode" base="string">
        <pattern value="[0-9]{5}(-[0-9]{4})?"/>
    </simpleType>

</schema>
```

Figure 2. An example for a schema definition.

instance of each class-expression in the list.

- **inverse** The name of a slot S that is the inverse of the slot being defined. If the pair $(x; y)$ is an instance of the slot S, then $(y; x)$ must be an instance of the slot being defined.
- **properties** A list of one or more properties of the slot. Valid properties are: **transitive** and **symmetric**.

The syntax of OIL is oriented on XML and RDF. The technical report on OIL [9] defines a DTD, an XML Schema definition, and a definition of OIL in RDFS.

4 XML SCHEMA

XML Schema is a means for defining constraints on the syntax and structure of valid XML documents (cf. [5], [12], [7]). A more easily readable explanation of XML Schema can be found in [13]. XML Schemata have the same purpose as DTDs, but provide several significant improvements:

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

We will discuss these four aspects in more detail.

4.1 XML schema definitions are itself XML documents.

Figure 2 shows an XML Schema definition of an address. The schema definition for the address tag is itself an XML document, whereas DTDs would provide such a definition in an external second language. The clear advantage is that all tools developed for XML (e.g., validation or rendering tools) can be immediately applied to XML schema definitions, too.

4.2 Datatypes

Datatypes are described in [5]. We already saw the use of a datatype (i.e., *string*) in the example. In general, a datatype is defined as a 3-tuple consisting of a set of distinct values, called its *value space*, a set of lexical representations, called its *lexical space*, and a set of *facets* that characterize properties of the value space, individual values, or lexical items.

Value space. The value space of a given datatype can be defined in one of the following ways: enumerated outright (extensional definition), defined axiomatically from fundamental notions (intensional definition)[1], defined as the subset of values from a previously defined datatype with a given set of properties, and defined as a combination of values from some already defined value space(s) by a specific construction procedure (e.g., a list).

Lexical space. A lexical space is a set of valid literals for a datatype. Each value in the datatype's value space is denoted by one or more literals in its lexical space. For example, "100" and "1.0E2" are two different literals from the lexical space of float which both denote the same value.

Facets. A facet is a single defining aspect of a datatype. Facets are of two types: fundamental facets that define the datatype and non-fundamental or constraining facets that constrain the permitted values of a datatype.

- Fundamental facets: equality, order on values, lower and upper bounds for values, cardinality (can be categorized as “finite”, “countably infinite” or “uncountably infinite”), numeric versus nonnumeric
- Constraining or non-fundamental facets are optional

[1] However, XML Schema does not provide any formal language for these intensional definitions. Actually *primitive datatypes* are defined in prose or by reference to another standard. *Derived datatypes* can be constrained along their facets (such as maxInclusive, maxExclusive etc.).

properties that can be applied to a datatype to constrain its value space: length constrains minimum and maximum, pattern can be used to constrain the allowable values using regular expressions, enumeration constrains the value space of the datatype to the specified list, lower and upper bounds for values, precision, encoding, etc. Some of these facets already constrain the possible lexical space for a datatype.

It is useful to categorize the datatypes defined in this specification along various dimensions, forming a set of characterization dichotomies.

- **Atomic vs. list datatypes:** Atomic datatypes are those having values which are intrinsically indivisible. List datatypes are those having values which consist of a sequence of values of an atomic datatype. For example, a single token which matches NMTOKEN from [XML 1.0 Recommendation] could be the value of an atomic datatype *NMTOKEN*, whereas a sequence of such tokens could be the value of a list datatype *NMTOKENS*.
- **Primitive vs. derived datatypes:** Primitive datatypes are those that are not defined in terms of other datatypes; they exist ab initio. Generated datatypes are those that are defined in terms of other datatypes. Every generated datatype is defined in terms of an existing datatype, referred to as the basetype. Basetypes may be either primitive or generated. If type *a* is the basetype of type *b*, then *b* is said to be a subtype of *a*. The value space of a subtype is a subset of the value space of the basetype. For example, *date* is derived from the base type *recurringInstant*.
- **Built-in vs. user-derived datatypes:** Built-in datatypes are those which are defined in the XML schema specification and may be either primitive or generated. User-derived datatypes are those derived datatypes that are defined by individual schema designers by giving values to constraining facets. XML Schema provides a large collection of such built-in datatypes, for example, string, boolean, flot, decimal, timeInstant, binary, etc. In our example, *zipCode* is an user-derived datatype.

4.3 Structures

Structures provide facilities for constraining the contents of elements and the values of attributes and for augmenting the information set of instances, e.g. with defaulted values and type information (see [12]). They make use of the datatypes for this purpose. An example is the element zip that makes use of the datatype zipCode. Another example is the definition of the element type “name”. The value “mixed” of the content-attribute allows to mix strings with (sub-)tags.

Attributes are defined by their *name*, a *datatype* that constraints their values, *default* or *fixed* values, and constraints on their *presence* (*minOccurs* and *maxOccurs*), see for example:

```
<attribute name="key" type="integer" minOccurs="1"
maxOccurs="1"/>
```

Elements can be constrained by reference to a simple datatype. The datatypes can be unconstrained, can be constrained to be empty, or can allow elements in its content (called rich content model).

- In the former case, **element declarations** associate an element name with a type, either by reference (e.g. *zip* in Figure 2) or by incorporation (i.e., by defining the datatype within the element declaration).
- In the latter case, the content model consists of a *simple grammar governing the allowed types of child elements and the order in which they must appear.* If the mixed qualifier is present, text or elements may occur. Child elements are defined via an **element reference** (e.g. <element ref="zip"/>) or directly via an **element declaration**. Elements can be combined in **group**s with a specific **order** (*all*, *sequence* or *choice)*. This combination can be recursive, for example, a sequence of some elements can be a selection from a different sequence or a sequence of different elements (i.e., the “()”, “,” and “| “of a DTD are present). Elements and their groups can be accompanied with **occurrence constraints**, for example, <element name="street" minOccurs="1" maxOccurs="2" type="string"/>.

In the previous subsection we already discussed the possibility of primitive vs. derived datatypes, where the latter further restricts the definition of the former (see [5]). An additional mechanism is provided via **derived type definitions** defined in [6]. Here the following two cases are distinguished:

- Derivation by **extension**. A new complex type can be defined by adding additional particles at the end of its definition and/or by adding attribute declarations. An example for such an extension is provided in Figure 3.
- Derivation by **restriction**. A new type can be defined by decreasing the possibilities made available by an existing type definition: narrowing ranges, removing alternatives, etc.

Important in this context is the following definition of [6]:

“A type T_1 is said to refine a type T_2 if and only if T_1 is declared to refine either T_2 or (recursively) some type that refines T_2. The effective constraints are the union of the explicit and the acquired.”

This implies that all inheritance has to be defined explicitly and cannot be derived from the definition of the types.

4.4 Namespaces

XML Schema provides the following mechanism for assembling a complete component set from several <schema> elements (cf. [12]):

include ::= URI

A <schema> element may contain one or more <include> elements. XML Schema uses the namespace mechanism when several schemata are combined:

import ::= *namespace* URI

In general, only inclusion is provided as means to combine various schemata and module name prefix is used to realize the non-equality of name assumptions (i.e., identifiers of two different schemata are by definition different).

```
<complexType name="personName">
    <element name="title" minOccurs="0"/>
    <element name="forename" minOccurs="0" maxOccurs="unbounded"/>
    <element name="surname"/>
</complexType>
<complexType name="extendedName" base="personName" derivedBy="extension">
    <element name="generation" minOccurs="0"/>
</complexType>
<element name="name" type="extendedName"/>
```

A snippet of a valid XML-file according to this schema is:

```
<name>
    <forename>Albert</forename>
    <forename>Arnold</forename>
    <surname>Gore</surname>
    <generation>Jr</generation>
</name>
```

Figure 3. An example for a derived type definitions via extension (taken from [6]).

5 THE RELATION BETWEEN OIL AND XML SCHEMA

On the one hand, ontologies and XML schemata serve very different purposes. Ontology languages are a means to specify domain theories and XML schemata are a means to provide integrity constraints for information sources (i.e., documents and/ or semistructured data). It is therefore not surprising to encounter differences when comparing XML schema with ontology languages like OIL. On the other hand, XML schema and OIL have one main goal in common: both provide vocabulary and structure for describing information sources that are aimed at exchange. It is therefore legitimate to compare both and investigate their commonalities and differences. In this section, we provide a twofold way to deal with this situation. First we analyze commonalities and differences and second we provide a procedure for translating OIL specifications into an XML Schema definition. As a guiding metaphor we use the relationship between the **relational model** and the **Entity Relationship model (ER model)**, cf. [2]. We realize that this analogy is only partially correct, because ER is a model for analysis, whereas OIL is a language for design. Nevertheless, the metaphor illustrates the relation nicely.

The relational model provides an implementation oriented description of databases. The Entity Relationship model provides a modeling framework for modeling information sources required for an application. In [2], Elmasri and Navathe also provides a procedure that translates models formulated in the Entity Relationship model into the relation model. During system development you start with a high-level ER model. Then you transform this model into a more implementation oriented relational model. As we will see in this section, it is surprising to see how easily the relationship between OIL and XML can be interpreted with this metaphor in mind. We will first compare both approaches and then we will provide a procedure on how to translate OIL specifications into an XML Schema definition. The overall picture is provided in Figure 4.

5.1 Comparing OIL and XML Schema

Both XML Schema and OIL have a XML syntax. This improvement of XML Schema compared to DTDs is also present in OIL. The XML syntax of OIL is useful for supporting the exchange of ontologies specified in OIL. It is defined in [9]. The translation approach for OIL which we will present in the following differs from this syntax because we describe some preprocessing instead of directly expressing OIL ontologies in XML Schema. These two XML schema definitions of OIL have different purposes: *In [9] we describe an XML syntax for writing ontologies in OIL. In this paper, we provide a structure and syntax (= a schema) for writing instances of an OIL ontology in XML.*

XML Schema has rich datatypes and OIL does not. XML Schema improves DTDs by providing a much richer set of basic datatypes than just PCDATA. XML Schema provides a large collection of built-in datatypes as, for example, string, boolean, float, decimal, timeInstant, binary, etc. OIL does not provide these built-in datatypes because reasoning with concrete domains quickly becomes undecidable or at least inefficient. XML Schema does not worry about this aspect because all inheritance needs to be defined explicitly. Undecidable in the derivation of implicit hierarchical relationships is therefore a nonexistent issue. In XML Schema, a datatype is defined by a value space, a lexical space, and a set of facets. Restricting a value space (i.e., the membership of classes) is also present in OIL, however, OIL does not provide a lexical space and facets. These aspects are much more related to the representation of a datatype than to the aspect of modeling a domain. That is, *date* may be an important aspect of a domain, but various different representations of *dates* are not. This is a rather important aspect when talking about how to represent the information. Finally, it should be noted that OIL is extremely precise and powerful in an aspect that is nearly neglected by XML Schema. XML Schema mentions the possibility of defining types intensionally via axioms. However, no language, semantics, nor any actual reasoning service is provided for this purpose. Here lies one of the main strengths of OIL. It is a flexible language for the intensional, i.e. axiomatic, definition of types. In a nutshell, neither OIL nor XML Schema are more expressive. Depending on the point of view, one of the

two approaches has richer expressive power: Built-in datatypes, lexical constraints and facets are not present in OIL. OIL provides an explicit language for the intensional definition of types that is completely lacking in XML Schema.

XML provides structures: elements. XML Schema's main modeling primitives are elements. Elements may be simple, composed or mixed. Simple elements have as their contents datatypes, like string or integer. Composed elements have as contents other (child) elements. Also they define a grammar that defines how they are composed from their child elements. Finally, mixed elements can mix strings with child elements. In addition, elements may have attributes. OIL takes a different point of view. The basic modeling primitives are concepts and slots. Concepts can be roughly identified with elements and child elements are roughly equivalent to slots defined for a concept. However, slots defined independently from concepts have no equivalents in XML Schema. This reconsolidates the relation between the relational model and the Entity Relationship model. The former only provides relations and the latter provides entities (with attributes) and relationships. In [2], a translation procedure is described from the Entity Relationship model, the richer modeling framework, to the relational model. Concepts and relationships are both expressed as relations. A similar reduction step has to be taken when transforming OIL specifications into XML Schema definitions.

XML provides structures: grammar. OIL does not provide any grammar for the application of slots to concepts, i.e., an instance of a concept comprises of a *set* of slots values. XML Schema allows the definition of stronger requirements via a grammar: *sequence* and *choice* of attributes applied to an instance can be defined.

XML provides structures: type-derivation. XML Schema incorporates the notion of type-derivation. However, this can only partially be compared with what is provided with inheritance in ontology languages like OIL. First, in XML Schema all inheritance has to be modeled explicitly. In OIL inheritance can be derived from the definitions of the concepts. Second, XML Schema does *not* provide a direct way to inherit from multiple parents. Types can only be derived from one basetype. OIL (like most ontology languages) provides multiple inheritance. Third, and very important, the is-a relationship has a twofold role in conceptual modeling which is not directly covered by XML Schema:

- Top-down inheritance of attributes from superclasses to subclasses. Assume *employee* as a subclass of a class *person*. Then *employee* inherits all attributes that are defined for *person*.
- Bottom-up inheritance of instances from subclasses to superclasses. Assume *employee* as a subclass of a class *person*. Then *person* inherits all instances (i.e., elements) that are an element of *employee*.

In XML Schema, both aspects can only be modeled in an artificial way. The top-down inheritance of attributes is difficult to model, because type derivations in XML Schema can either extend *or* restrict the base type. A "dummy" intermediate type has to be used to model full top-down inheritance of attributes with both extending and restricting derivations. For example, it is not possible to model a student as a *person* with a student-number and age < 28 in only one step. You first have to model a dummy type "young person", which *restricts* the age of persons to less than 28. After that it is possible to model a student as a "young person" *extended* with a student-number.

Also the bottom-up inheritance of instances to superclasses is not automatically available in XML Schema. However, using an additional attribute, it is possible to use an instance of a subclass wherever a superclass of it is expected. For example, to use a

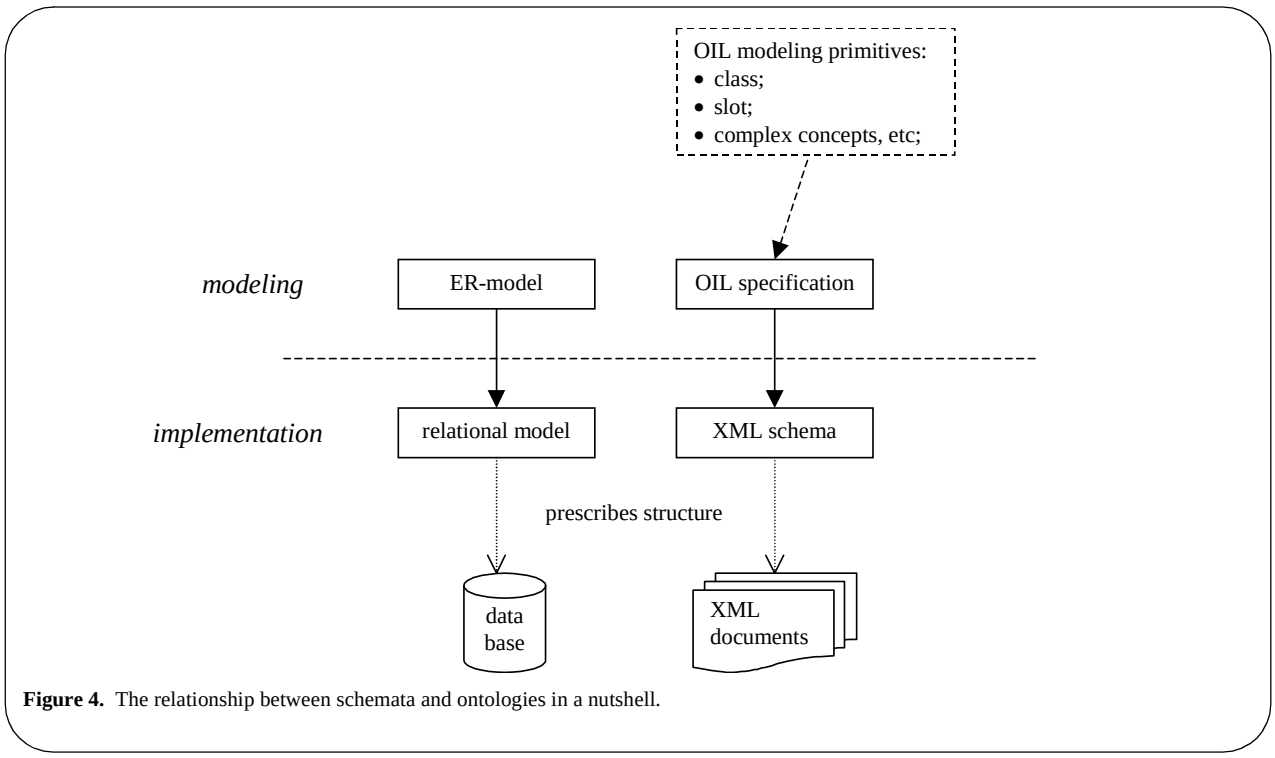

Figure 4. The relationship between schemata and ontologies in a nutshell.

```
Geographical_Location
Country                    <= Geographical_Location
City                       <= Geographical_Location
AreaComparison
KilometerLength-OR-MilesLength
KilometerLength            <= KilometerLength-OR-MilesLength
MilesLength                <= KilometerLength-OR-MilesLength
[,,,]
capital                    <= slot
has_boundary               <= slot
coastline                  <= slot
neighbor_country           <= slot
```

Figure 5. Step 1: materializing the hierarchy.

student as a filler of a "driver" element, which requires type person, we can write:

```
<driver xsi:type="student">
    <name>John</name>
    <studentnumber>0792098</studentnumber>
</driver>
```

We have to provide to type of the derived element explicitly in the instance document. This is done with the *type* attribute, which is part of the XML Schema instance namespace. However, it is still not possible to *query* for all persons and also obtain all subtypes of person.

XML provides namespaces (OIL, too). XML Schema and OIL provide the same (limited) means for composing specifications. Specifications can be included and the underlying assumptions is that names of different specifications are different (via different prefixes).

The message of this section in a nutshell is that OIL relates to XML Schema like the Extended Entity Relationship model relates to the relational model. On the one hand, OIL provides much richer modeling primitives. It distinguish classes and slots, and class (or slot) definitions can be used to derive the hierarchy (and its according inheritance). On the other hand, XML Schema provides richer modeling primitives concerning the variety of built-in datatypes and the grammar for structuring the content of elements. The latter is not of importance when building a domain model but important when defining the structure of documents. Models in OIL can be viewed as a high level description that is further refined when aiming at a document structure model. We will prove this statement by providing a translation procedure close in spirit to that provided in [2].

5.2 Translating OIL specifications into an XML Schema definition

ER models provide entities, attributes, and relationships as their primary modeling primitives. This closely corresponds to OIL where we have concepts (i.e., entities), slot definitions for concepts (i.e., attributes), and global slot definitions (i.e., relationships). Extended ER models also incorporate the notion of inheritance, however, require their explicit definition. On the other hand, the relation model only provides relations and the arguments (called attributes) of relations. Therefore, a translation step is required when translating the high-level modeling approach (Extended) ER into schema definitions of relational databases. [2] provide a procedure for this translation. We will describe a similar procedure that translates a high-level conceptual description of a domain into a specific document definition via XML Schema.

We assume a definition of an ontology in OIL. An example is provided in Figure 1. We will now describe its stepwise translation into an XML schema using the stepwise translation of this example as illustration.

First, materialize the hierarchy. Give all complex class expressions that are used in subclass definitions and slot constraints names. Then, materialize the hierarchy, i.e., make all class- and slot-subsumptions explicit. This is necessary because XML Schema lacks any notion of implicit hierarchy and it is possible because subsumption is decidable in OIL. Actually, the FaCT system can be used for this purpose (via its CORBA interface if desired [14]). In this step, also derive all *implicit slot-constraints*, exploiting the domain and range restrictions of the global slots definitions. Figure 5 provides the materialized hierarchy of our running example. Note that KilometerLength-OR-MilesLength is a new concept, constructed from a complex class expression. In our small example, there are no new class subsumptions derived, because all of them are already stated explicitly. See [9] or [10] for a more complex example which illustrates the definition and derivation of implicit subsumptions.

Second, create a complexType definition for each slot definition in OIL. Add references to (still to be defined) element definitions for every **range** components that is present in the OIL slot-definition. Figure 6 shows some example slot-definitions. If a slot has more than one range, the element references are placed inside a in a <choice> element. This means that every slot shall have of exactly one element.

Third, also create a complexType definition for each class definition in OIL. Add the names of the slots that can be applied to the classes as elements in the type definition. The facets on the slot-constraints are translated in the following way: **has-value** facets give a minOccurs="1" attribute in the element-element, **value-type** facets give minOccurs="0" and **min-cardinality**, **max-cardinality,** and **cardinality** give minOccurs="value", maxOccurs="value" or both as attributes respectively.

For the slots that appear in explict slot-constraints, an anonymous type is defined, which is derived from the appropriate slot-type defined in step two. The extension of the base type consist of the reference to the class which must be the filler of the slot.

For slots that can be applied to the classes (according to their domain) but that are *not* present in explict slot-constraints, the

type of the element is directly the slot-type from step 2, with the attributes minOccurs="0" and maxOccurs="unbounded". Figure 7 gives an example.

Fourth, create an element definition for each slot and class. Each slot and each class definition is translated into an element definition in the XML schema. The type of the elements will obviously be the complexType definitions which are created in the second and third step. See Figure 8.

Fifth, define a grammar for each entity, associate basic datatypes with built-in datatypes if desired, add lexical constraints on datatypes if desired. This step adds an additional level of expressiveness that is not present in OIL. It is purely concerned with document structure and appearance.

Sixth, replace the module concept of OIL with the namespace and inclusion concept of XML Schema. This step is straightforward because the concepts only differ syntactically.

Using the schema for document markup

The resulting schema can be used to create XML instance documents. The structure of these documents must conform to the schema. As an example, we show in Figure 9 an XML document which could constitute a wepage in the World Fact Book. Together with a appropriate stylesheet, this document can be used to produce a page as is shown in Figure 10. Note that we now have a webpage which looks similar to the original HTML version, but which has a markup that is well-founded on an ontology.

```
<complexType name="slotType"/>
<complexType name="capitalType" base="slotType" derivedBy="extension">
    <element ref="City"/>
</complexType>
<complexType name="relative_areaType" base="slotType" derivedBy="extension">
    <element ref="AreaComparison"/>
</complexType>
<complexType name="coastlineType" base="slotType" derivedBy="extension">
    <choice>
        <element ref="KilometerLength"/>
        <element ref="MilesLength"/>
    </choice>
</complexType>
```

Figure 6. Step 2: type definitions for slots.

```
<complexType name="GeographicalLocationType">
    <element name="name" type="string" minOccurs="0" maxOccurs="unbounded"/>
    <element name="coastline" type="coastlineType" minOccurs="0" maxOccurs="unbounded"/>
</complexType>

<complexType name="CountryType" base="GeographicalLocationType" derivedBy="extension">
    <element name="capital" type="capitalType" minOccurs="0" maxOccurs="unbounded"/>
    <element name="has-boundary" type="has_boundaryType" minOccurs="0" maxOccurs="unbounded"/>
    <element name="relative_area" type="relative_areaType" minOccurs="0" maxOccurs="unbounded"/>
</complexType>

<complexType name="AeraComparisonType">
    <element name="proportion" type="string" minOccurs="0" maxOccurs="unbounded"/>
    <element name="compared_to" minOccurs="0" maxOccurs="unbounded">
        <complexType base="compared_toType" derivedBy="extension">
            <element ref="Country"/>
        </complexType>
    </element>
</complexType>
```

Figure 7. Step 3: example of some type definitions for a classes.

```
<element name="capital" type="capitalType"/>
<element name="GeographicalLocation" type="GeographicalLocationType"/>
<element name="Country" type="CountryType"/>
<element name="City" type="CityType"/>
<element name="AeraComparison" type="AeraComparisonType"/>
```

Figure 8. Step 4: element definitions for classes and slots

```
<?xml version="1.0" encoding="UTF-8"?>
<Country     xmlns="worldfactbook.xsd"
             xmlns:xsi="http://www.w3.org/1999/XMLSchema/instance">
    <name>The Netherlands</name>
    <capital><City><name>Amsterdam</name></City></capital>
    <relative_area>
        <AeraComparison>
            <proportion>slightly less than twice the size of</proportion>
            <compared_to><Country><name>New Jersey</name></Country></compared_to>
        </AeraComparison>
    </relative_area>
    <has_boundary>
        <LandBoundary>
            <neighbor_country><Country><name>Belgium</name></Country></neighbor_country>
            <length xsi:type="KilometerLength"><value>450</value><unit>km</unit></length>
        </LandBoundary>
        <LandBoundary>
            <neighbor_country><Country><name>Germany</name></Country></neighbor_country>
            <length xsi:type="KilometerLength"><value>577</value><unit>km</unit></length>
        </LandBoundary>
    </has_boundary>
    <coastline><KilometerLength><value>451</value><unit>km</unit></KilometerLength></coastline>
```

Figure 9. Example of XML document conforming the generated schema.

6 DISCUSSION

In the previous section, we compared OIL to XML Schema, as two specific examples of an ontology language and a XML document schema. Main results of the comparison are:

- It is correct to say that OIL has more expressive power than XML Schema but this is also true the other way around in the sense that XML Schema is much richer in defining structures and grammars for information elements and in the large variety of basic data types they provide.
- It is true that ontologies can be used for describing semi-structured natural language texts, but the same holds for XML Schema.
- It is true that an ontology must be a shared and consensual terminology. However, there are serious efforts to achieve the same for XML. This stems from the fact that XML has, from its inception, been a language for providing information via the WWW, not for designing "private" databases.
- It is true that an ontology provides a domain theory and not the structure of a data container. This helped us to explain most of the differences between ontologies (i.e., ontology languages) and XML schemata (i.e., the XML Schema definition language).

In the following, we will discuss some of the points that arose when investigating the relation between ontologies and XML schemata.

First, multiple inheritance forms a problem in the translation procedure. As we already discussed in Section 5.1, in XML Schema there is no explicit way to define multiple inheritance. Conform the XML Schema specification, it is also not possible to define a type multiple times, thus inheriting attributes from more than one supertype. Multiple inheritance, which is an important aspect in most ontology languages, is therefore not expressable in XML Schema.

Second, the question may arise as to whether the translation process can be automated and whether it is reversible. Concerning the first question, we can state that most of the steps can be completely automatic. The fifth step can be partially automated, for example by using sequence as standard grammar for applying slots to classes. In addition, by matching class names with the names of built-in datatypes of XML Schema, some datatypes could automatically be applied. Final tuning via human interference may be necessary. The reverse direction is possible but more difficult, a high degree of automatization should be achievable, however.

The relation that is described in this paper, is in the same strain as earlier approaches on relating ontology languages and XML (cf. [17], [18], [19], and [16]). However, this approaches did not deal with XML Schema but with its predecessor, i.e.with DTDs, and focussed on proper translation of attribute inheritance in tag nesting. This mapping is less interesting because DTDs provide very limited expressiveness compared to ontologies and XML Schema. In this paper, we went a step further and used the XML Schema type hierarchy to express the conceptual knowledge. Although there are still some questions about the best way to create this type hierarchy, with the help of some artifices it is possible to capture the central is-a relationship of an ontology in an XML Schema definition. However, as we showed in this paper, XML Schema is not suitable as a ontology language. This is not meant as a criticism, because XML Schema is not designed for ontological modeling, it is designed for describing valid structures of documents.

Further, we want to stress the difference between the XML Schema definition of OIL that can be found in [9] and the kind of schema that is presented in this paper. Their purposes are completely different. In [9], an XML Schema definition is provided to write down an ontology in a plain XML document. In this article, we try to generate a schema that captures the underlying semantics of an ontology, which can be used for representing instances of an OIL ontology in XML.

Finally, we would like to mention that the procedure described in

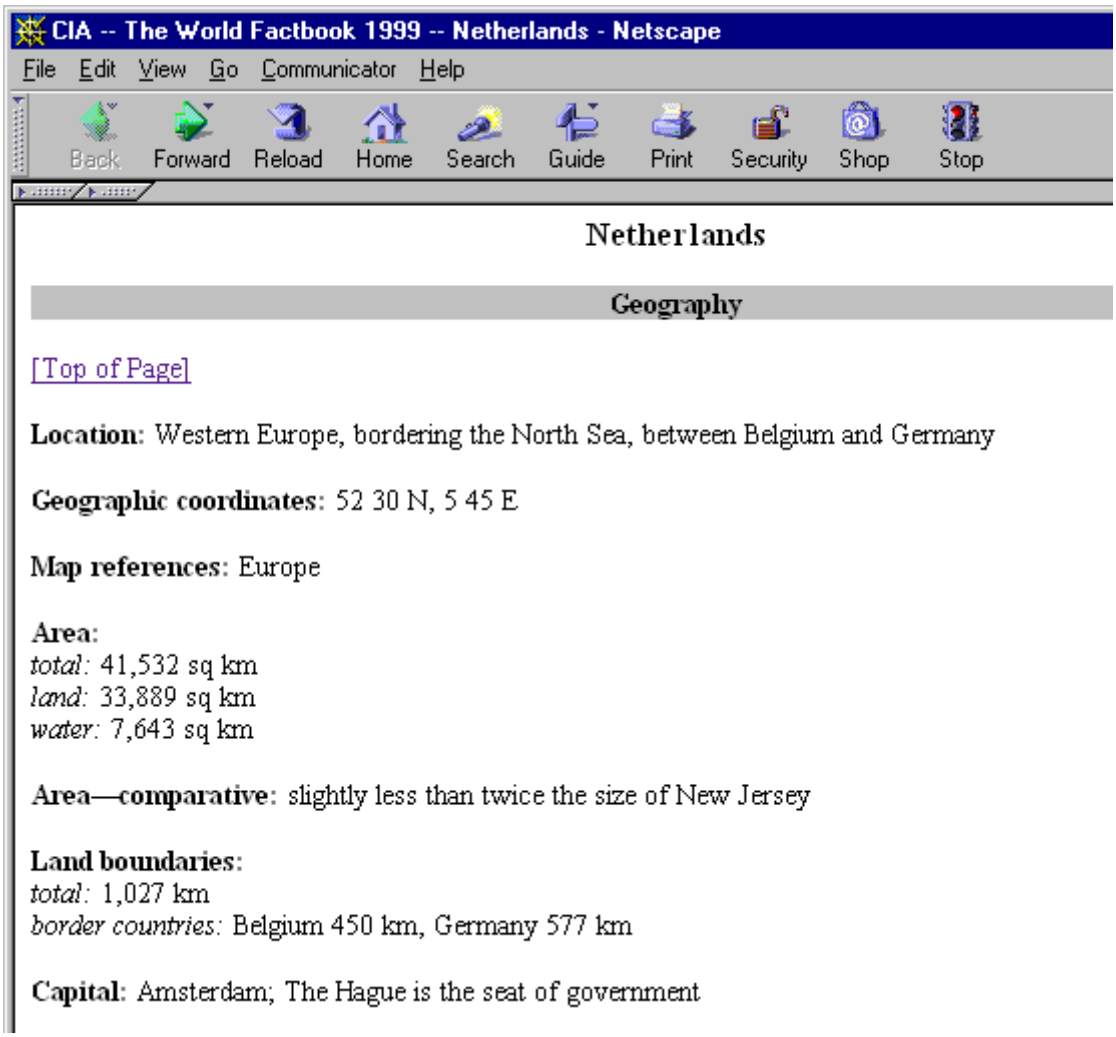

Figure 10. An possible view on the data from Figure 9.

this paper is certainly not the only way to relate ontologies to data on the web [16]. We do not want to advocate XML Schema as the most appropriate way to go. As already mentioned, another, and complementary way to annotate data with ontological information is the use of RDF and RDF Schema. In this case, OIL should be defined as an extension to RDF Schema (as is already done in [11]) and the ontology should be written in RDF Schema. The instance documents could then be marked-up with RDF. This way of annotating documents has the advantage that the interpretation of the markup is already in RDF: it is clear from the syntax which constructs are classes, slots, and so on.

7 CONCLUSION

When comparing ontologies and XML schemata directly we run the risk of trying to compare two incomparable things. Ontologies are domain models and XML schemata define document structures. Still, when applying ontologies to on-line information sources their relationship becomes closer. Then, ontologies provide a structure and vocabulary to describe the semantics of information contained in these documents. The purpose of XML schemata is prescribing the structure and valid content of documents, but, as a side effect, they also provide a shared vocabulary for the users of a specific XML application.

Therefore, we compared the Ontology Inference Layer OIL with the proposed XML Schema standard in this paper. We did not compare OIL to other ontology languages, because the goal of this paper is to investigate the relation between ontology languages and document schemata. Such comparisons can be found in [9].

Our main conclusion is that the two refer to different levels of abstraction and therefore also to two different phases in describing the semantics of on-line information sources. OIL provides much richer primitives: concepts, slots, complex concept and slot definitions, and the implicit definition of concept and slot hierarchies. You have to first materialize this hierarchy and translate concepts and slots into elements in order to express these aspects via an XML Schema definition. On the other hand, then you also have richer modeling primitives to express constraints over the content of information sources. You can make use of the variety of built-in datatypes of XML Schema and you can define grammars for the structure of elements. Therefore, the two approaches do not conflict but rather are concerned with different phases in the development process of on-line

information sources. Finally, OIL may be a candidate for providing intensional definitions of datatypes, an aspect which is offered but undefined in XML Schema.

The procedure described in this paper considers an ontology as a conceptual layer on top of a set of structured (XML) documents. As a result, the actual data in XML documents is annotated with a well founded semantic markup. This semantically grounded markup complements the kind of semantic annotation that is provided by RDF based approaches. We think that, together with the RDF based approaches, the foundation of document markup in ontologies is an important building stone of the semantic web.

ACKNOWLEDGMENT

Credits to Uwe Krohn who stimulated the work on this paper. Thanks to Peter Fankhauser for explaining tricky aspects of XML Schema, to Borys Omelayenko for his case study on the World Fact Book and to Frank van Harmelen for his helpful comments and the review of early drafts.

REFERENCES

[1] O. Lassila and R. R. Swick: Resource Description Framework (RDF): Model and Syntax Specification, W3C Recommendation, 22 February 1999. http://www.w3.org/TR/REC-rdf-syntax/

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

[3] A. Newell: The Knowledge Level, *Artificial Intelligence*, 18:87—127, 1982.

[4] N. Guarino and C. Welty: A Formal Ontology of Properties, In, Dieng, R., and Corby, O., eds, *Proceedings of EKAW-2000: The 12th International Conference on Knowledge Engineering and Knowledge Management.* AAAI Press, Menlo Park. October, 2000.

[5] P. V. Biron and A. Malhotra: XML Schema Part 2: Datatypes, W3C Working Draft 7 April 2000 `http://www.w3.org/TR/2000/WD-xmlschema-2-20000407/`

[6] H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn: *XML Schema Part 1: Structures*, W3C Working Draft, 17 December 1999. `http://www.w3.org/TR/1999/WD-xmlschema-1-19991217/.`

[7] N. Walsh: Schemas for XML, July 1, 1999. `http://www.xml.com/pub/1999/07/schemas/index.html`

[8] D. Fensel: Ontologies: *Silver Bullet for Knowledge Management and Electronic Commerce*, Springer-Verlag, to appear 2000. `http://www.cs.vu.nl/~dieter/ftp/spool/silverbullet.pdf`

[9] I. Horrocks, D. Fensel, J. Broekstra, S. Decker, M. Erdmann, C. Goble, F. Van Harmelen, M. Klein, S. Staab, and R. Studer: OIL: The Ontology Inference Layer, Technical Report, Vrije Universiteit Amsterdam, July 2000. `http://www.ontoknowledge.com/oil.`

[10] D. Fensel, I. Horrocks, F. Van Harmelen, S. Decker, M. Erdmann, and M. Klein: OIL in a Nutshell. In, Dieng, R., and Corby, O., eds, *Proceedings of EKAW-2000: The 12th International Conference on Knowledge Engineering and Knowledge Management.* AAAI Press, Menlo Park. October, 2000.

[11] J. Broekstra, M. Klein, D. Fensel, S. Decker, and I. Horrocks: Adding formal semantics to the Web: building on top of RDF Schema, submitted `http://www.ontoknowledge.org/oil/extending-rdfs.pdf`

[12] H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn: *XML Schema Part 1: Structures*, W3C Working Draft, 7 April 2000. `http://www.w3.org/TR/2000/WD-xmlschema-1-20000407/ .`

[13] David C. Fallside: *XML-Schema Part 0: Primer*, W3C Working Draft, 7 April 2000. `http://www.w3.org/TR/2000/WD-xmlschema-0-20000407/`

[14] S. Bechhofer, I. Horrocks, P. F. Patel-Schneider, and S. Tessaris: *A proposal for a description logic interface*. In P. Lambrix, A. Borgida, M. Lenzerini, R. Möller, and P. Patel-Schneider, editors, *Proceedings of the International Workshop on Description Logics (DL'99)*, pages 33-36, 1999.

[15] R. A. Meersman: The use of lexicons and other computer-linguistic tools in semantics, design and cooperation of database systems. In *CODAS Conference Proceedings, Lecture Notes in Artificial Intelligence (LNAI)*, ed. Y. Zhang. Springer Verlag, Berlin, pp. 1-14, 1999.

[16] D. Fensel: Relating Ontology Languages and Web Standards. In J. Ebert et al. (eds.), *Modelle und Modellierungssprachen in Informatik und Wirtschaftsinformatik*, Modellierung 2000, St. Goar, April 5-7, 2000, Foelbach Verlag, Koblenz, 2000.

[17] A. Rabarijoana, R. Dieng, and O. Corby: Exploitation of XML for Corporative Knowledge Management. In D. Fensel and R. Studer (eds.), *Knowledge Acquisition, Modeling, and Management, Proceedings of the European Knowledge Acquisition Workshop (EKAW-99)*, Lecture Notes in Artificial Intelligence, LNAI 1621, Springer-Verlag, 1999.

[18] M. Erdmann and R. Studer: *Ontologies as Conceptual Models for XML Documents*, research report, Institute AIFB, University of Karlsruhe, 1999.

[19] C. Welty and N. Ide: Using the Right Tools: Enhancing retrieval From Marked-Up Documents, *Computers and the Humanities*, 33:1-2, Special Issue on the Tenth Anniversary of the Text Encoding Initiative, 1999. `http://www.cs.vassar.edu/faculty/welty/papers/.`