Ontologies as Conceptual Models for XML Documents

Michael Erdmann, Rudi Studer

Institut für Angewandte Informatik und Formale Beschreibungsverfahren (AIFB)
University of Karlsruhe (TH)
D-76128 Karlsruhe (Germany)
e-mail: {erdmann, studer}@aifb.uni-karlsruhe.de

Abstract: Access to XML-based documents currently relies on query languages that are closely tied to the document structures, i.e. when looking for information one has to be aware of this structure and cannot easily specify the information needs conceptually. Our approach uses ontologies to access sets of distributed XML documents on a conceptual level. We integrate conceptual modeling, inheritance, and inference mechanisms on the one hand with the popularity, simplicity, and flexibility of XML on the other hand. We present an approach that defines the relationship between a given ontology and a document type definition (DTD) for classes of XML documents. Thus, we are able to supplement syntactical access to XML documents by conceptual, i.e. real semantic access.

1 Introduction

The Extensible Markup Language (XML) [Bray et al. 98] is currently on the way to conquer the web. Dozens, maybe hundreds of applications of this flexible language have been developed and more will surely follow. XML is designed to describe document types for all thinkable domains and purposes, e.g. XML documents may represent multi-media presentations (SMIL, cf. [Hoschka 98]), HTML-pages of arbitrary contents (XHTML, cf. [Pemberton et al. 99]), business transactions [XML/EDI-Group 99], or several other things (cf. [Cover 99] for the most comprehensive list we know of). XML documents are explicitly structured textual documents that can be easily accessed by application programs (via standardized interfaces like SAX [Megginson 98] and DOM [Apparao et al. 98]). This is one main strength of XML that will probably lead to the emergence of XML-based repositories in businesses in the near future, that represent e.g. yellow pages, project and skill descriptions, or publication lists. Thus, XML could play an important role as a basic technology in the context of knowledge management and dissemination and also when it comes to managing large scale web sites. XML supports such techniques as corporate design, style sheets (XSL [Deach 99]), automatic generation of customized views to documents, consistency between documents, superior linking facilities (XLink [Maler, DeRose 98a], XPointer [Maler, DeRose 98b]) etc. All this is based on an individually definable tag set that is tailored to the application needs. The tags have semantic purposes, in contrast to pure layout purposes as in HTML, so that they can be exploited for several tasks such as those mentioned above or as metadata that supports intelligent information retrieval.

In spite of these positive features and prospects of XML it must be clearly stated that XML is solely a description language to specify the structure of documents and thus their syntactic dimension. The document structure can represent some semantic properties but it is not clear how this can be deployed outside of special purpose applications. We will define in this paper how to add true semantics to XML documents by relating the document structure to an ontology. By mapping

ontology concepts and attributes to XML elements via the definition of a DTD, XML documents can be authored that represent facts that are compatible with the designed domain model, i.e. an ontology. An ontology is a "formal specification of a conceptualization" [Gruber 93] and thus, provides a basis for semantics-based processing of XML documents. We will argue that ontologies are the appropriate level for structuring the contents of documents because they speak about concepts and semantic relationships rather than element nesting or sequential order. Of course concepts and relationships have to be expressed and stored in linear form in documents; but this is pure representation, i.e. DTDs and the document structure are not enough to give XML a sound semantics. The representation can be derived automatically from a conceptual description as it can be defined by an ontology. If the ontology is the primary source for structuring documents these documents can be accessed in a more convenient, i.e semantic way. Conceptual terms can be used to retrieve facts. Thus, the ontology is a kind of mediator between the information seeker and the set of XML documents. It unifies the different syntaxes/structures of these documents and can add background knowledge to the process of answering a query. Our approach allows true semantic queries to the contents of XML documents and relieves the information seeker from knowing and accessing the structure of all relevant documents.

The work presented in this paper was developed in the context of the Ontobroker project [Decker et al. 99] that enables intelligent access to the contents of (annotated) HTML pages, that are still the most prevalent sources of information in the web. Semantic metadata is provided by an extension to the HTML language. Ontobroker uses ontologies to realize the intelligent information integration and retrieval. Due to the emergence of XML as *the* future standard representation for semi-structured documents the new version of Ontobroker (cf. [Fensel et al. 99]) is able to process XML documents.

This paper describes how XML is integrated into the framework of the Ontobroker system which is briefly presented in section 2. The next section shows how access to XML documents is currently realized by XML query languages and how Ontobroker+XML can solve some of the problems arising there. Section 4 presents the tool *DTDMaker* that derives an XML document type definition (DTD) from a given ontology, so that XML instances can be linked to an ontology and thus, can be accessed through Ontobroker. Finally, we conclude with a review of some related work and future plans concerning XML and Ontobroker.

2 Defining the Context

2.1 Ontobroker

The presented work is one contribution to the further development of Ontobroker[1] (cf. [Decker et al. 99] and [Fensel et al. 99]). This approach to intelligent information integration and retrieval is briefly presented in this section to provide the framework in which we will combine ontologies and XML documents.

The Ontobroker project [Decker et al. 99] uses techniques from Artificial Intelligence, in particular ontologies and deductive inference systems to provide access to heterogeneous and distributed semi-structured documents. The approach taken in this project (in the beginning) was to annotate HTML-documents with semantic metadata, to collect these data, store it centrally, and to make the populated knowledge base accessible through query facilities. In the knowledge base facts are

1. http://www.aifb.uni-karlsruhe.de/WBS/broker

stored in Frame Logic [Kifer et al. 95], an object oriented and logic-based language from the deductive database community. Underlying all parts of Ontobroker is an ontology [Gruber 93] that defines the vocabulary for annotating documents, for formulating queries, and to structure the knowledge base. The ontology represents a domain model, is formalized in Frame Logic as well, and is a part of the knowledge base. The ontology definition contains an is-a hierarchy of relevant domain concepts, possible relationships between concepts, further properties of concepts (attributes with value ranges), and derivation rules to infer new knowledge.

The principle architecture of Ontobroker (Figure 1) consists of four independent parts. The *query engine* allows information seekers to pose queries to the knowledge provided by Ontobroker. The *data store* contains the knowledge base and can be realized by powerful data-base management systems, thus large amounts of data can be dealt with. An *inference engine* provides services that derive additional knowledge through inference rules on demand or in advance. The *info agent* is responsible to collect the raw data from distributed sources. It is composed of modules that realize data collection for different kinds of documents. The info agent can access (i) HTML documents annotated with HTML-A, (ii) it realizes wrappers to multiple instance documents which have some exploitable structure in common, (iii) it may access RDF metadata [Decker et al. 98], or (iv) it can import the information contained in XML documents. This last appearance of the info agent (i.e. its XML module) will be described in more detail in the remaining sections.

All four parts of Ontobroker are conceptually linked by an *ontology*, the overall structuring principle of the whole system. The ontology is used by the inference engine to infer new knowledge based on inference rules. The knowledge base is organized with respect to the ontology, and the query engine uses it to provide guidance when formulating queries. Lastly the info agent uses ontologies to extract facts, i.e. to translate from the original sources into the conceptual model of the system. This is true regardless of the document type of input, i.e. HTML-A, RDF, or XML.

The small extension of HTML called HTML-A that was developed in Ontobroker project smoothly integrates machine processible metadata into existing document contents, and thus prevents duplication of information, reduces redundancy, and improves maintainability. But these annotations have to be produced manually, which can be a rather time consuming and error-prone endeavour. Although HTML is the major language for presenting information in the web the document structure seldom mirrors the semantic content sufficiently. Problems in annotating web

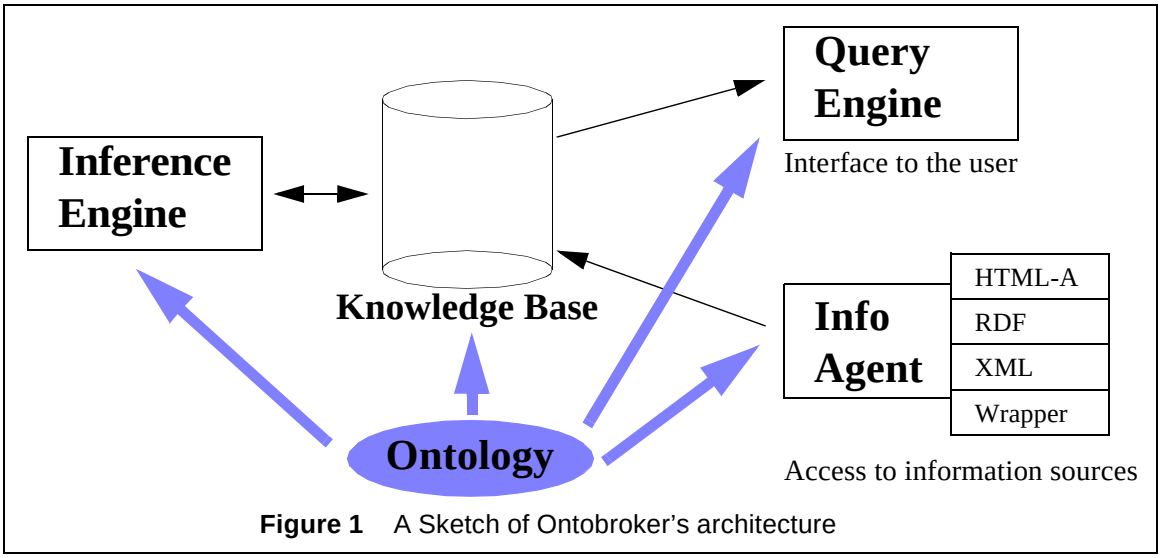

Figure 1 A Sketch of Ontobroker's architecture

pages especially stem from HTML's focus on rendition. XML solves this problem since it keeps content, structure, and representation apart and is a much more adequate means for knowledge representation.

How Ontobroker's info agent maps the structure of XML documents to an ontology (and vice versa) follows in the next subsection and in section 4.

2.2 Role of *DTDMaker* in Ontobroker

Ontobroker can retrieve information stored in XML documents because its info agent provides a mapping between the structure and contents of XML documents and conceptual entities of an ontology. The ontology represents a model of a domain that helps structuring the system internal knowledge base and is used in our approach to also structure the XML documents.

So, the first step to make information retrieval from XML documents possible in Ontobroker is to define a domain ontology. In our example and the Ontobroker project this ontology is formalized in Frame Logic. This Frame Logic representation of the ontology serves as input to a translator that automatically derives a document type definition (DTD) for XML documents (cf. section 4). This component is called *DTDMaker* and is written in Java. The derived DTD defines a rather unrestricted class of documents because the variety of knowledge expressible with the ontology must be representable by valid XML documents as well.

In the next step, contents has to be produced, i.e. the XML instances have to be authored. The info agent collects completed XML documents and imports them into the knowledge base. This import process needs the ontology to retranslate the serialization of instances of ontological concepts and relationships from XML to the knowledge base. Although several documents that may have various structures must be imported, the direct relationship to the ontology makes this task a straightforward one.

At this stage we have a populated knowledge base ready to be queried. Of course, it must be assured, that changes in documents are reflected in the knowledge base and that additional documents can be added. These more administrative tasks are handled by Ontobroker's update mechanisms and are outside the scope of this paper.

An overview of the relationship between the ontology, the DTD, and XML instances is given in figure 2.

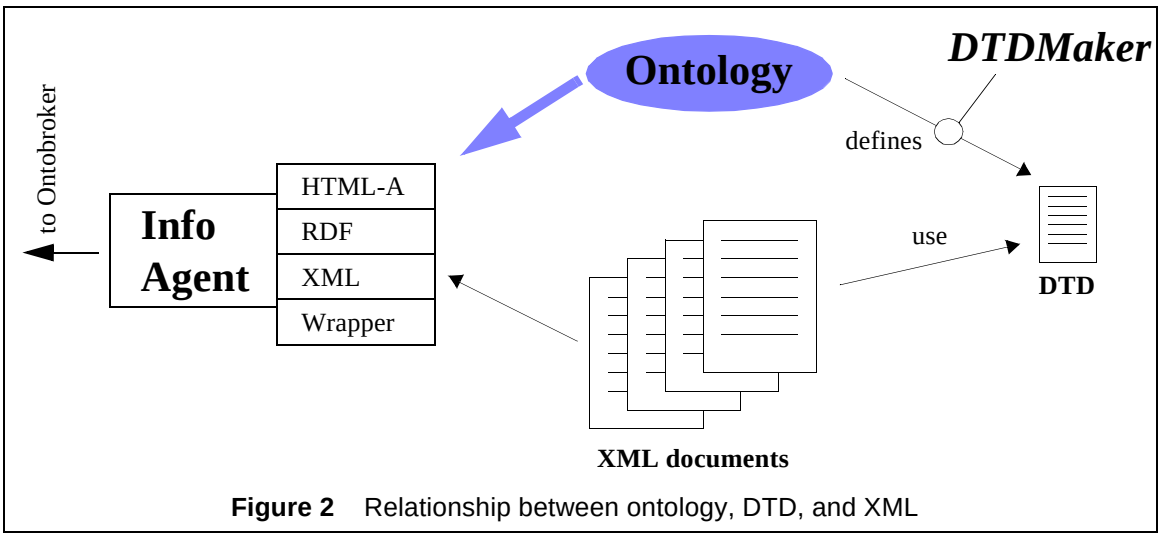

Figure 2 Relationship between ontology, DTD, and XML

3 Query Answering

Although the focus of this paper does not lie in answering queries to XML documents but in connecting these document sets with ontologies, the main goal for combining ontologies and XML is improving the power for accessing the contents of documents such as by query answering. This section will describe current ways of accessing semi-structured data in general and XML documents in particular, show deficits, and how Ontobroker+XML can overcome these deficits.

Access to semi-structured data is currently realized by a number of query languages, e.g. Lorel [Abiteboul et al. 97], Lorel for XML [Goldman et al. 99], XML-QL [Deutsch et al. 98], or XQL [Robie et al. 98][Robie 99]. These query languages access the structure of documents to locate the relevant information (using kinds of path expressions or templates for navigating in the document tree), i.e. they are closely tied to the document structure (i.e. its syntax).

An example in XML-QL syntax illustrates this. We are looking for people with knowledge of the SGML language:

```
WHERE
     <people>
          <person>
               <name> $P </name>
               <know-how> SGML </know-how>
          </person>
     </people> IN "some URL"
CONSTRUCT $P
```

The query delivers the content of the `name` subelement of `person`, iff the person has another subelement `know-how` containing "SGML". When applied to the following XML document we (only) achieve `Peter` and `Hans` as answers.

```
<?xml version="1.0" encoding="utf-16"?>
<skills>
     <people>
          <person>
               <name> Peter </name>
               <know-how> SGML </know-how>
          </person>
          <person>
               <name> Hans </name>
               <know-how> SGML </know-how>
               <know-how> C++ </know-how>
          </person>
     </people>
     <seminars>
          <seminar>
               <topic> SGML </topic>
               <participant>
                    <name> Dieter </name>
                    <name> Gisela </name>
               </participant>
          </seminar>
     </seminars>
</skills>
```

The given XML document also contains facts about seminars and let us assume that participants of these seminars acquire some knowledge of the seminars' topics. When we make this assumption the knowledge that person *X* has know-how of topic *Y* is implicitly encoded in the document. To retrieve (in XML-QL) all persons with the required knowledge a second query has to reflect this explicitly by specifying an alternative pattern to retrieve the participants of seminars with SGML as topic.

```
WHERE
    <seminar>
        <topic> SGML </topic>
        <participant>
            <name> $P </name>
        </participant>
    </seminar> IN "some URL"
CONSTRUCT $P
```

In XML-QL both queries cannot be combined to one since disjunction is not expressible in this language. A combined query achieving the expected results can be formulated by another XML query language, namely XQL [Robie et al. 98]. It looks like this:

```
//person/name[../know-how="SGML"]
    $union$
//seminar[topic="SGML"]/participant/name
```

Again all possible encodings of the relevant knowledge has to be guessed by the information seeker to formulate such a query. Furthermore, XQL has the disadvantage (at least in the ´98 version) that joins are not possible, since no variables can be bound and only information from the direct context is accessible during evaluation of the path expressions.

It is easy to see that an information seeker cannot be expected to formulate (multiple) queries in this way, at least not if the domain contains several (semantic) dependencies between different parts of documents. If the domain is complex and the document structures are varied such a *procedural* way of retrieving knowledge does not seem promising. To retrieve the desired information an information seeker ideally should be able to specify his wishes declaratively using conceptual terms only, e.g. "Which persons have knowledge about SGML?" or in a more formal way:

```
FORALL P <- P:person[know-how="SGML"].
```

This, actually, is a declarative query in Frame Logic [Kifer et al. 95] to a knowledge base (cf. section 2.1). The facts in this central data repository are collected from the same XML document that were addressed by the XML-QL and XQL queries. The repository mediates between the information seeker on the one hand and the different information sources on the other hand. The concrete realization of the XML documents is unimportant for the information seeker since he is interested in semantically meaningful answers irrespective of the documents' syntaxes. The conceptual terms used in the query (`person` and `know-how`) are defined in the ontology and may be used in XML documents. To fully deploy the power of ontologies in this context, a close relationship between the ontology and the structure of XML documents is necessary. In section 4, we will show how this relationship is established, namely via the derivation of a document type definition (DTD) from an ontology.

The ontology has two purposes in supporting queries to the knowledge base: (i) it defines a common vocabulary to enable structure independent queries and (ii) it provides additional

background knowledge to improve the quality of answers. In our example the assumption made above, namely that participants of seminars acquire some knowledge, is part of the ontology's background knowledge. It is expressed in the ontology as an inference rule, such that the query retrieves participants of SGML seminars, as well.

Another area in which ontologies provide direct benefit for query answering comes from the modeled concept taxonomy. Assume Hans is a programmer and this fact is represented by the following snippet of an XML document:

```
<programmer>
    <name> Hans </name>
    <know-how> SGML </know-how>
    <know-how> C++ </know-how>
</programmer>
```

Given this situation the "traditional" languages for querying XML become even more impractible, i.e. Hans would not be retrieved by any of the given queries (although everybody knows that programmers are persons, too ;-). This kind of knowledge is modeled in the concept taxonomy of the ontology and so can help the information seeker to find the conceptually correct answers, i.e. the given Frame Logic query retrieves Hans even from the altered document.

To sum up the state of the art in accessing the contents of XML documents we can say: current query languages are strong when the location of wanted data is known in advance. But, an information seeker has to be aware of the document structure to locate relevant data, i.e. cannot specify his information needs conceptually (even when elaborated path expression and pattern languages using disjunctions, wildcards, conditions etc. are available) but has to stick to the document syntax. Further, implicit knowledge cannot be exploited when the documents are processed directly and thus, this additional knowledge cannot be made available to information seekers. In our approach true semantic queries to the contents of XML documents become possible and the information seeker is relieved from the burden of accessing the structure of documents.

4 Deriving DTDs from Ontologies

This section will present the tool *DTDMaker* that produces an XML Document Type Definition (DTD) based on a given ontology. This is done by mapping ontology concepts and attributes to XML elements, thus that these documents are compatible with the ontology. We do *not* claim that a DTD can be a substitution for a complete formal definition of an ontology but we claim that both are needed, ontology and DTD. DTDs are important since most XML applications do not care about the existence of an ontology. These applications can still read, write, and process documents as usual, and the documents can still be checked for structural validity (due to the DTD). The ontology is important as a semantic basis for applications that are ontology aware and can benefit from the knowledge encoded in the ontology (e.g. for semantic queries). The generation of a DTD from an ontology means that to some extent DTD validity is an indication for semantic validity, as well, i.e. documents complying to the generated DTD are also valid with respect to the ontology.

Thus, our approach has significantly other goals than languages such as RDF Schema (RDFS [Brickley, Guha 99]) or the Ontology Mark-up Language (OML [Kent 99]). Their primary goals are to encode the ontology in XML terms, i.e. the representation language of the ontology becomes XML. Both define XML elements that can be used to express some kind of terminological knowledge that constitutes the ontology. This is sufficient when a representation language is sought

that is expressive enough to encode an ontology but it is not sufficient to constrain the structure of XML documents that will contain real data according to the ontology. RDFS is only loosely coupled to RDF documents that can e.g. define instances of concepts contained in the RDFS-defined ontology. OML does not even provide a formalism to use the ontology defined in the OML document. Our approach on the other hand takes an ontology (represented in a logical language and not in XML) and generates from it a DTD that can be used by nearly all existing XML applications. Thus, applications can profit from the knowledge encoded in the ontology as well as from existing XML technology.

We will now illustrate the mechanisms behind *DTDMaker* by giving examples for all relevant cases.

4.1 General Functionality of *DTDMaker*

The general idea of *DTDMaker* is as follows:

- Each concept from the ontology generates an element type in the DTD.
- For each attribute of these elements the DTD defines a subelement.
- If the attribute represents a relation to another concept the attribute element has as content the respective concept element, otherwise its content model is simply PCDATA.

This is only one possible translation scheme from ontologies to DTDs. Others are imaginable, e.g. instead of "each concept becomes an element type" only some concepts could be translated into element types, all other concepts might be identified by a `type` attribute attached to an element derived from a super concept. E.g., instead of `<Researcher> ... </Researcher>` the DTD could prescribe `<Person type="Researcher"> ... </Person>`. The chosen approach has the advantage, that the DTD models all concepts following a consistent strategy that does not need human intervention for stating which concepts become elements and which are modeled via the `type` attribute.

The following example ontology is a subset of an ontology [Benjamins, Fensel 98] that specifies among other things the domain of researchers and publications.

```
Object[].
Person :: Object.
    Employee :: Person.
        AcademicStaff :: Employee.
            Researcher :: AcademicStaff.
                PhDStudent :: Researcher.
    Student :: Person.
        PhDStudent :: Student.
Publication :: Object.
    Book :: Publication.
    Article :: Publication.
        TechnicalReport :: Article.
        JournalArticle :: Article.
    Journal :: Publication.

Person[name =>> STRING; email =>> STRING; phone =>> STRING;
            publication =>> Publication; editor =>> Book].
```

```
Employee[employeeNo =>> STRING].
AcademicStaff[supervises =>> PhDStudent].
Researcher[cooperatesWith =>> Researcher].
Student[studentID =>> NUMBER].
PhDStudent[supervisor =>> AcademicStaff].
Publication[author =>> Person; title =>> STRING; year =>> NUMBER;
            abstract =>> STRING].
Book[editor =>> Person].
TechnicalReport[series =>> STRING; number =>> NUMBER].
JournalArticle[journal =>> Journal; firstPage =>> NUMBER;
            lastPage =>> NUMBER].
Journal[editor =>> Person; volume =>> NUMBER; number =>> NUMBER;
            containsArticle =>> JournalArticle].

FORALL Pers1, Pers2
        Pers1:Researcher[cooperatesWith ->> Pers2] <->
                Pers2:Researcher[cooperatesWith ->> Pers1].
FORALL Pers1, Publ1
        Publ1:Publication[author ->> Pers1] <->
                Pers1:Person[publication ->> Publ1].
FORALL Pers1, Publ1
        Publ1:Book[editor ->> Pers1] <->
                Pers1:Person[editor ->> Publ1].
FORALL Pers1, Pers2
        Pers1:PhDStudent[supervisor ->> Pers2] <->
                Pers2:AcademicStaff[supervises ->> Pers1].
FORALL Publ1, Publ2
        Publ2:Journal[containsArticle ->> Publ1] <->
                Publ1:JournalArticle[journal ->> Publ2].
```

The ontology comprises a concept hierarchy of persons and of publications and defines several associations between these concepts. The ontology is represented in Frame Logic. The term `X::Y` means `X` is a subconcept of `Y`. Thus, the ontology states that `Person` is a subconcept of `Object`, `Employee` and `Student` are subconcepts of `Person` etc. The concept `PhDStudent` inherits properties from both `Student` and `Researcher`. The inherited properties are defined in the second part of the ontology, namely the associations between concepts. Here, associations are realized by attributes of the appropriate types. Besides pure attributes with the atomic value ranges `STRING` and `NUMBER`, relations to other concepts are specified, e.g. the concept `PhDStudent` has an attribute `supervisor` of type `AcademicStaff` and vice versa `AcademicStaff` may `supervise PhDStudents`. The third part of the ontology contains rules or axioms. These rules will be used to derive new knowledge based on the given facts, e.g. if we know that some `Researcher` *A* `cooperatesWith` *B*, we can infer that this *B* must be a `Researcher` as well and `cooperatesWith` *A*. Thus the ontology enables applications to round out incomplete knowledge.

The presented tool *DTDMaker* takes the described ontology as input and derives a DTD for structuring XML documents. The DTD is too large to present it in its entirety, so only small fragments will be shown to illustrate how the ontology is mapped to a structure for XML documents (see below for a larger portion of the DTD). Since inheritance is a central feature of ontologies that is not supported in XML (XML schemas provide a partial solution to this problem, cf. [Malhotra, Maloney 99], and section 5 for a discussion) this feature has to be brought in by other

means. We use parameter entities of XML for this purpose. Parameter entities define substitution strings that can be used throughout the DTD. Each time a parameter entity is referenced, this reference is replaced with the substitution string. In *DTDMaker* for each concept *C* that has sub-concepts (and subsub-concepts etc.) C_1, C_2, C_3, C_4 a parameter entity is defined:

```
<!ENTITY % C "C | C1 | C2 | C3 | C4" >
```

This definition states that whenever the parameter entity `%C;` is referenced in the DTD it is substituted by the disjunction `"C | C1 | C2 | C3 | C4"`, i.e. wherever in the XML document the concept `C` can be inserted its subconcepts are equally allowed. The ontology states that `Person`s may have `Publication`s. Through inheritance it is perfectly legal to define an `Article` as the value of a `Person`'s `publication`-attribute. This fact must be expressible in a valid XML document as well, and thus, specified in the DTD:

```
<!ENTITY % Publication "Publication | Book | Article | Journal ...">
```

DTDMaker maps concepts from the ontology to element types in the DTD, i.e. for each concept an element type is defined. The contents of these element types consist of elements that represent the concept's attributes. The order in which these attribute elements must occur is not defined by *DTDMaker* since the ontology is a syntax independent specification. For example, the concept `Book` has five attributes including inherited attributes like `author` or `title`.

```
<!ELEMENT Book (#PCDATA| year | abstract | title |
                  author | editor)*>
```

In an XML document this element may look like this:

```
<Book>
    <title> The SGML Handbook. </title>
    <author> Charles F. Goldfarb </author>
    <year> 1990 </year>
</Book>
```

As the example illustrates, the DTD does not dictate the order of elements nor does it require all of them to be present. The DTD even allows pure character data as contents of elements (cf. section 4.4 for a discussion of the strictness of the DTD).

The last piece in the puzzle is the specification of the contents of the attribute elements. This contents specification is governed by the attribute value types as defined in the ontology. The attribute `title` of `Publication` has the atomic value type `STRING` and thus does not specify a relationship to another concept. Consequently the `title` element may have only character data as content.

```
<!ELEMENT title (#PCDATA) >
```

The `author` attribute defines an association between a `Publication` and a `Person` and the cooperation between `Researchers` is expressed with the `cooperatesWith` attribute. The element type definition produced by *DTDMaker* matches these intentions:

```
<!ELEMENT author (#PCDATA | %Person;)* >
<!ELEMENT cooperatesWith (#PCDATA | %Researcher;)* >
```

An `author` element may have one or more `Person` subelements that represent the `authors` of the embracing `Publication`. The element type definition does not refer to the element type `Person` directly, rather it uses the earlier defined parameter entity `%Person;`. By doing so, not only

`Person` is allowed as subelement of `author` but also `Researcher`, `PhDStudent` etc. Replacing the parameter entity with the appropriate substitution string would lead to the following element type definition.

```
<!ELEMENT author (#PCDATA | Person | Employee | AcademicStaff |
                  Researcher | PhDStudent | Student)* >
```

By mapping ontology concepts and attributes to XML elements via the definition of a DTD, XML documents can be authored that represent facts that are compatible with the designed domain model, i.e. the ontology. A part of the DTD derived from the ontology (see above) follows:

```
<!-- entities for ontology concepts to realize is-a hierarchy -->
<!ENTITY % Person "Person | Employee | AcademicStaff |
                   Researcher | PhDStudent | Student" >
<!ENTITY % Researcher "Researcher | PhDStudent" >
<!ENTITY % Publication "Publication | Book | Article |
                   TechnicalReport | JournalArticle |Journal" >
<!ENTITY % Article "Article | TechnicalReport | JournalArticle" >
<!ENTITY % Book "Book" >

<!-- element declarations for ontology concepts -->
<!ELEMENT Person (#PCDATA | name | email | phone | publication |
                   editor)*>
<!ELEMENT Researcher (#PCDATA | name | email | phone |
                   publication | editor | employeeNo |
                   supervises | cooperatesWith)*>
<!ELEMENT Publication (#PCDATA | author | title | year |
                   abstract)*>
<!ELEMENT Article (#PCDATA | author | title | year | abstract)*>
<!ELEMENT Book (#PCDATA | author | title | year | abstract |
                   editor)*>

<!-- element declarations for ontology attributes -->
<!ELEMENT author (#PCDATA | %Person;)* >
<!ELEMENT title (#PCDATA) >
<!ELEMENT year (#PCDATA) >
<!ELEMENT editor (#PCDATA | %Book; | %Person;)* >
<!ELEMENT cooperatesWith (#PCDATA | %Researcher;)* >
```

The presented DTD contains three sections. (i) At the beginning entities are defined that *imitate* the **is-a hierarchy** of ontology concepts (using the substitution string-trick). (ii) The second section declares the content model (i.e. the subelements) of XML elements representing ontology **concepts**. These subelements represent attributes of concepts, e.g. `phone` and `publication` are attributes of `Person`. In the third section the contents of elements generated from ontology **attributes** are defined. The content is either `PCDATA` for atomic attribute value types (e.g. the `title` element) or elements representing the concepts defined as the attribute's value type in the ontology (e.g. the `author` attribute has a value type of `Person`).

4.2 DTDs are broad

The DTD leaves open which element will become the root element of a concrete document. Thus different XML documents can have different root elements while still conforming to the same DTD and with that to the underlying ontology. For example, a document providing information about a researcher would state:

```
<!DOCTYPE Researcher SYSTEM "ka2.dtd">
```

Its root element would become `Researcher`, and its contents would be the researcher's properties expressed by elements, e.g.:

```
<Researcher>
    <name> Joe Doe </name>
    <address> Foostreet 1, Foocity </address>
    <email> doe@foo-bar.com </email>
    <cooperatesWith> Winnie Poo </cooperatesWith>
    <publication>
        <Book>
            <title> ... </title>
            <year> ... </year>
            <editor> ... </editor>
        </Book>
        <Article>
            <title> ...</title>
            <author> 2nd author comes here </author>
            <abstract> ... </abstract>
        </Article>
    </publication>
</Researcher>
```

Another document might describe a single publication, e.g. a book. Here, the document type declaration would consist of

```
<!DOCTYPE Book SYSTEM "ka2.dtd">
```

followed by the document element `Book` and its contents. As can be seen by these examples, each document represents the specification of an instance of an ontological concept, i.e. a `Person` or a `Publication`.

A set of manifold documents can be created and accessed using just one DTD and just one ontology. This is beneficial because all documents follow one common domain model (i.e. an ontology) and thus are all processible by the same (kind of) applications, that "understand" their common DTD. As a consequence the set of these documents establishes a coherent and consistent knowledge base that is connected syntactically via the DTD and semantically via the ontology.

With the tool *DTDMaker* it becomes possible to model a domain conceptually and afterwards to automatically obtain an XML DTD, that defines document structures that are compatible with the ontology. This is especially important when XML documents should act as a communication medium between different partners because during the development process of the ontology the developers can abstract from the final textual representation and agree upon relevant concepts, their properties, and relationships, i.e. semantics. In this way it should be easier to come to a consensus concerning a shared view of the domain.

For this reason, besides Frame Logic other modeling languages would be also possible for modeling the domain model, e.g. the Unified Modeling Language (UML) that provides all modeling primitives (among other things) that are needed by *DTDMaker*. But using pure object oriented modeling languages has the big disadvantage of missing axioms for representing rich background knowledge that goes much further than deploying the is-a relationship.

4.3 Referencing Conceptual Objects in XML

To be able to reason about the knowledge represented in XML documents it is necessary to identify each instance of a concept unambiguously. In an object oriented sense this means, each individual must have a unique (object) identifier. We identify XML elements with conceptual objects, so we have to introduce the concept of object identity for these conceptual elements, i.e. we have to declare object identifiers in XML. XML provides a special type of attribute for elements. This ID type ensures uniqueness of the values of all attributes of this type within a document. Each element that is derived from an ontological *concept* is provided with such an ID attribute, e.g.:

```
<!ATTLIST Publication OID ID #IMPLIED>
<!ATTLIST Researcher OID ID #IMPLIED>
```

A document that conforms to an ontology derived DTD expresses (semantic) relations between objects either through the element nesting as shown in previous sections or by referring to objects/ elements using their object identifier. These references can be made in any element that is derived from an ontological *attribute* that has some concepts as value range. DTDMaker defines these references as follows:

```
<!ATTLIST author OIDREF CDATA #IMPLIED>
<!ATTLIST cooperatesWith OIDREF CDATA #IMPLIED>
```

Within a single document these `OIDREF` attributes can be interpreted as simple attributes of type `IDREF` that refer to an element with a respective value for its `ID` attribute. But we cannot generally specify `OIDREF` as having the type `IDREF` as one might expect because of *Validity Constraint: IDREF* from the XML specification [Bray et al. 98] that states "Values of type `IDREF` must match the `Name` production [...]; each `Name` must match the value of an `ID` attribute on some element in the XML document; i.e. `IDREF` values must match the value of some `ID` attribute." Instead of the simple `ID/IDREF` mechanism to refer to other elements we use XPath expressions [Clark, DeRose 99] that can be appended to an URL to locate elements in other XML documents. XPath unifies two W3C activities concerning pattern languages for XML that were independently started: XPointer [Maler, DeRose 98b] and XSL/XSLT [Clark 99]. XPath expressions are much more powerful and subsume the simple `IDREF` approach. They allow for the formulation of location paths that retrieve document nodes from the document tree by navigating in a sophisticated manner from the root of the document (or any other tree node) to the concept node in question. This is an elegant way to refer to concepts that are encoded in XML and several XML application can process and understand XPath expressions (resp. XSLT, XPointer), thus this approach is viable from an XML point of view. But, it has one big disadvantage: the document structure becomes crucial for referencing objects. This was our main complaint about current XML query languages that access XML documents based on similar patterns or path expressions. When we want to formulate references based on semantical properties we have to consider the ontology somehow, i.e. an alternative has to be found. We propose an additional means to describe references: the use of queries within a special attribute. To not confuse these queries with XPath expressions we introduce another reference attribute called `cOIDREF` that **c**onceptually refers to objects:

```
<!ATTLIST author cOIDREF CDATA #IMPLIED>
<!ATTLIST cooperatesWith cOIDREF CDATA #IMPLIED>
```

This `cOIDREF` attribute contains a query formulated in Frame Logic that returns a single object or a set of objects.

```
<Book>
    <title> The XML Handbook </title>
    <author cOIDREF='FORALL p <-
                p:Researcher[name="Charles F. Goldfarb"] OR
                p:Researcher[name="Paul Prescod"].'>
    </author>
    <year> 1998 </year>
</Book>
```

This example defines the value of the `author` attribute of an instance of the concept `Book` via a Frame Logic query. The `authors` have to be `Researchers` and their names are either Charles F. Goldfarb or Paul Prescod. If there exist multiple Charles F. Goldfarbs this reference incorrectly names all of them as authors of "The XML Handbook". If this should be the case the query has to be refined to disambiguate the references.

This approach is very general, and might be too complicated for average XML authors, thus an easier variant should be desired. The third possibility to refer to objects from XML documents is a mixture of the first two approaches. We are currently enhancing the XML query language XQL [Robie et al. 98] with ontological background knowledge. The enhancement comes from rewriting XQL queries based on the ontology. For example, if we ask in XQL for all `Persons` described in a document, the query

```
//Person
```

only retrieves a subset of all semantically correct answers if the document contains concepts/ elements like `Researcher` or `Student`. This is because the subconcepts of `Person` do not match the pattern specified in the above query. This can be repaired if the query would be reformulated. The reformulation can be done automatically by utilizing the ontology. The rewritten query looks like this

```
//Person $union$ //Employee $union$ //AcademicStaff $union$
//Researcher $union$ //PhDStudent $union$ //Student
```

and will retrieve all intended elements. Thus this mechanism can be used to formulate references from an attribute element to a concept element within XML documents. Currently we are investigating how far rewriting of XQL queries can get us. It is clear that a simple one pass rewriting step will not suffice to create XQL queries with the power of Frame Logic queries. But the expressiveness of rewritten XQL queries seem sufficient to act as references to other objects.

4.4 DTD ≠ Ontology

The expressiveness of XML to define the structure of documents in a DTD is of course not sufficient to reflect all aspects of a formal ontologies. The consequences from this divergence are listed in this section extended by proposals for overcoming the drawbacks.

- Attributes are no longer local to a concept, they are global in a document. So, any possible subelements must be added to the attribute's content model. This leads to a DTD regarding to which more documents are classified as valid as intended. In the example ontology `editor` is an attribute of `Person` as well as of `Book`, thus the element declaration for editor contains the appropriate value types `%Book;` and `%Person;`, respectively. This implies that a valid document may contain a `Book` element with an `editor` subelement that has again a `Book` subelement, what is not intended.

  A partial solution to this problem are XML Schemas that provide a higher expressiveness than DTDs. But currently only one schema aware parser exists and we do not know of any other schema aware applications that could benefit from the schema description.

- If the ontology contains a concept *X* and some (other) concept has an equally named attribute this (currently) leads to a malformed DTD because of violation of the validity constraint "unique element type declaration" [Bray et al. 98]. *DTDMaker* does not yet cope with resolving this conflict.

  To resolve this situation the ontology designers could follow some policies to avoid such conflicts, e.g. all concept names could begin with a capital letter and all attribute names could begin with lower case letters. This policy is followed in the presented example ontology.

- The DTD is not as strict as it could be. This is due to usability reasons of the DTD, esp. when manually producing XML documents. The DTD allows `PCDATA` everywhere where subelements may be inserted. As a consequence every element has a mixed content model which restricts the constraints that can be expressed.

  This is less a consequence of the expressiveness of DTDs but more a design decision made when implementing *DTDMaker*. Making *DTDMaker* more strict would lead to PCDATA only for attributes with atomic value ranges.

- Frame Logic and most other conceptual modeling languages allow the formulation of cardinality constraints for attributes and relationships. These cardinality constraints can be expressed in several ways and in different granularities (e.g. *n:m* notation vs. *min-max* notation). Frame Logic distinguishes single valued and set valued attributes. The example ontology only uses set valued attributes because of its origin in the open web context, in which as few restriction as possible were coded into the ontology. Although XML knows three cardinalities (i) exactly once, (ii) optional, and (iii) zero or many times *DTDMaker* does not exploit this feature (yet). This is due to the fact that all defined content models are mixed contents and thus all subelements have to be of cardinality version (iii). Here again, XML schemas could bring remedy.

5 Conclusion, Related and Future Work

In this paper we showed that ontologies provide a compact, formal, and conceptually adequate way of describing the semantics of XML documents. By deriving DTDs from an ontology the document structure is grounded on a true semantic basis and thus, XML documents become adequate input for semantics-based processing. By providing a conceptual foundation for XML we achieve at the same time a way to access XML documents rather independently of their actual linear representation, i.e. the ontology mediates between the conceptual terms used by an information seeker and the actual mark-up used in XML documents.

Our approach relates to work from the areas of semi-structured data models, query languages, and metadata. We do not claim that semi-structured data models or query languages are not relevant for XML, instead we claim that they need to be complemented by ontology-based approaches like ours. They are powerful tools to retrieve the contents of documents based on the document structure. The data models of all these approaches (among others XML-QL [Deutsch et al. 98], Lorel for XML [Goldman et al. 99], XSL [Deach 99], and XQL [Robie et al. 98]) directly reflect the document structure, i.e. its syntax. Ontobroker+XML abstracts from this structure and refers to the contents as concepts and relationships, instead.

The relationship between our approach and RDF/RDFS [Lassila, Swick 99][Brickley, Guha 99] is manifold. Both define an ontology that is used to structure the contents of XML documents. We use Frame Logic and automatically derive a DTD that constraints the possible document structures. In the RDF/RDFS context the ontology is encoded in XML itself using the primitives provided by RDFS. In both cases the ontology is seen as a semantic complement to the DTD describing syntactic properties only. Both approaches encode the factual knowledge in XML. Differences lie in the expressible ontological primitives. Frame Logic comes from an object oriented tradition where each class has its own local set of attributes, whereas in RDF attributes are global and not uniquely associated with a class. A more important distinction comes from role DTDs play. In our approach DTDs represent the connection to ontology unaware but (DTD) validating XML applications. RDF documents in general do not conform to a DTD and thus are not processible by validating XML applications. This is particularly true if RDF statements are embedded in other resources as additional metadata.

A similar approach of combining ontologies and DTDs from semistructured documents can be found in the Untangle project[2]. In [Ide et al. 97] and [Welty, Ide 99] it is described how an SGML DTD can be translated into the core of an ontology. Their representation language for ontologies is CLASSIC, a language from the KL-ONE (description logics) family. In the core ontology concepts are mainly derived from element types in the DTD. This core ontology has to be extended manually with meaningful semantic relationships, new concepts can be introduced etc. A populated knowledge base can then be queried utilizing this additional background knowledge. One difference to our approach lies in the differing representation languages for the ontology. In our implementation of Frame Logic it is possible to define inference rules using several built-in predicates, such as regular expressions or mathematical operations. But, the most obvious difference lies in the direction of processing. For us, a domain model defined in an ontology is the starting point, which allows for a clear definition of document types. In Untangle the ontology is based on and partially derived from a DTD. This approach has the advantage of being able to process *existing* documents and nevertheless to utilize domain knowledge for query processing. Our approach of generating DTDs from a conceptual model hides syntactical details from the DTD developers. They can focus on the relevant domain terms and thus, the role of ontologies for knowledge sharing is stressed. At the same time, this makes later access to encoded data easier and more powerful, since ontological knowledge can be immediately associated with these data. The Untangle project does not have this modeling aspect of ontologies in mind, that helps to yield document structures with a clear semantics.

XML schemas will go in a similar direction of raising the level of abstraction when desinging the structure of documents. [Malhotra, Maloney 99] define a list of requirements for a future standard for XML schemas that are the designated successor of DTDs. First results produced by the W3C

2. `http://untangle.cs.vassar.edu`

Schema Working Group can be found at `http://www.w3.org/TR/xmlschema-1/` (XML Schema Part 1: Structures) and `http://www.w3.org/TR/xmlschema-2/` (XML Schema Part 2: Datatypes). Schemas are expressed in XML themselves and thus, can be processed by the manifold XML-tools, like editors, parsers, transformators. Schemas will define real data types for XML attribute values or element types (e.g. `integer` or `dateTime`). Schemas will allow the definition of global attributes. Elements will be able to inherit attributes and content models of other elements, such that a schema will become more compact than the corresponding DTD. If the defined requirements for XML schemas are met some aspects of our work, esp. the handling of inheritance, will be realized. That implies that (in the future) transforming ontologies into XML Schemas seems to be more appropriate than to DTDs. But still, XML Schemas primarily cope with document structure and not with semantics. XML Schemas do not provide any inference capabilities and narrowly focus on XML alone. As far as we know no tools exist that associate schema descriptions, XML documents, and processing capabilities that fully exploit the conceptual description of a schema. With Ontobroker+XML we currently have a tool that can make use of schematic knowledge provided by the ontology. Nevertheless, XML Schemas will become more widely known and used in the future, so that we have to position our work relative to schemas and probably will support them when the standardization process will be a little more advanced.

One of the topics which we will tackle in the future is the consideration of multiple or changing ontologies. Since our approach grounds on the existence of an ontology the management of changes to that ontology and the implications for existing documents have to be considered. Another topic will be to tailor the automatically derived DTD to additional constraints, that are not expressed in the ontology, such as element order or cardinality. Since, it cannot be expected that application development always starts with modeling an ontology we must take care of existing XML document structures or XML Schemas, how they can be related to an ontology, or how they can be used to derive an ontology (cf. [Welty, Ide 99]). This reverse direction allows (i) to keep and use the existing XML documents and structures, (ii) to use all existing applications that create, access, manipulate, filter, render, and query these documents, and (iii) at the same time to benefit from the domain knowledge modeled in the ontology by utilizing smarter applications that can complement (or even replace) the existing applications in some areas esp. query answering.

Acknowledgements. The authors would like to thank our colleagues Stefan Decker and Steffen Staab for fruitful discussions as well as the KAW reviewers for their very helpful comments on the submission version of this paper.

6 References

[Abiteboul et al. 97] **S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener**: The Lorel query language for semi-structured data. in: *Journal of Digital Libraries*. Volume 1, No. 1, 1997

[Apparao et al. 98] **V. Apparao et al. (eds.)**: Document Object Model (DOM). *W3C Recommendation*, October 1, 1998. `http://www.w3.org/DOM/`

[Benjamins, Fensel 98] **R. Benjamins and D. Fensel**: The Ontological Engineering Initiative (KA)2. in: *Proceedings of the International Conference on Formal Ontologies in Information Systems (FOIS-98)*, Trento, Italy, **N. Guarino (eds.)**, Frontiers in Artificial Intelligence and Applications, IOS-Press, June 1998.

[Brickley, Guha 99] **D. Brickley and R.V. Guha**: Resource Description Framework (RDF)

Schema Specification. *W3C Proposed Recommendation,* March 3, 1999. http://www.w3.org/TR/PR-rdf-schema

[Bray et al. 98] **T. Bray, J. Paoli, and C.M.Sperberg-McQueen (eds.):** Extensible Markup Language (XML) 1.0. *W3C Recommendation,* February 10, 1998. http://www.w3.org/TR/1998/REC-xml-19980210

[Clark 99] **J. Clark (ed.)**: XSL Transformations (XSLT) Secification 1.0. *W3C Working Draft*, April 21, 1999. http://www.w3.org/TR/1999/WD-xslt-19990421.html

[Clark, DeRose 99] **J. Clark, S. DeRose (eds.)**: XML Path Language (XPath) 1.0. *W3C Working Draft*, August 13, 1999. http://www.w3.org/1999/08/WD-xpath-19990813.html

[Cover 99] **R. Cover**: The SGML/XML Web Page - XML: Proposed Applications and Industry Initiatives. (updated daily), OASIS. http://www.oasis-open.org/cover/xml.html#applications

[Deach 99] **S. Deach (ed.)**: Extensible Stylesheet Language (XSL) Specification. *W3C Working Draft*, April 21, 1999. http://www.w3.org/TR/WD-xsl

[Decker et al. 98] **S. Decker, D. Brickley, J. Saarela, and J. Angele**: A Query and Inference Service for RDF. in: *Proceedings of the W3C Query Language Workshop (QL-98)*, Boston, MA, December 3-4, 1998.

[Decker et al. 99] **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.)**: *Semantic Issues in Multimedia Systems*, Kluwer Academic Publisher, Boston 1999.

[Deutsch et al. 98] **A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu**: XML-QL: A Query Language for XML. *W3C Note*, August 19, 1998. http://www.w3.org/TR/NOTE-xml-ql/

[Fensel et al. 99] **D. Fensel, A. Witt, J. Angele, S. Decker, M. Erdmann, H.-P. Schnurr, S. Staab, and R. Studer:** On2broker in a Nutshell (poster). in: *Proceedings of the 8th World Wide Web Conference (WWW8)*, Toronto, May 11-14, 1999. extended version: *On2broker: Lessons Learned from Applying AI to the Web*. Technical Report no. 383, Inst. AIFB, Univ. Karlsruhe, December 1998. ftp://ftp.aifb.uni-karlsruhe.de/pub/mike/dfe/paper/on2broker.pdf

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

[Goldman et al. 99] **R. Goldman, J. McHugh, and J. Widom**: From Semistructured Data to XML: Migrating the Lore Data Model and Query Language. Stanford University 1999. ftp://db.stanford.edu/pub/papers/xml.ps

[Gruber 93] **T. R. Gruber**: A Translation Approach to Portable Ontology Specifications. in: *Knowledge Acquisition*. vol. 6, no. 2, 1993. pp199-221

[Hoschka 98] **P. Hoschka (ed.)**: Synchronized Multimedia Integration Language (SMIL) 1.0. *W3C Recommendation*. June 15, 1998. http://www.w3.org/TR/1998/REC-smil-19980615

[Ide et al. 97] **N. Ide, T. McGraw, and C. Welty**: Representing TEI Documents in the CLASSIC Knowledge Representation System. *Proceedings of the Tenth workshop of the Text-Encoding Initiative*. November, 1997.

[Kent 99] **R. Kent**: Creating a WAVE (Web Analysis and Visualization Environment). `http://wave.eecs.wsu.edu/`

[Kifer et al. 95] **M. Kifer, G. Lausen, and J. Wu**: Logical Foundations of Object-Oriented and Frame-Based Languages, *Journal of the ACM*, 42, 1995.

[Lassila, Swick 99] **O. Lassila and R.R. Swick**: Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation, February 22, 1999. `http://www.w3.org/TR/REC-rdf-syntax`

[Maler, DeRose 98a] **E. Maler and S. DeRose (eds.)**: XML Linking Language (XLink). *W3C Working Draft*. March 3, 1998. `http://www.w3.org/TR/1998/WD-xlink-19980303`

[Maler, DeRose 98b] **E. Maler and S. DeRose (eds.)**: XML Pointer Language (XPointer). *W3C Working Draft*. March 3, 1998. `http://www.w3.org/TR/1998/WD-xptr-19980303`

[Malhotra, Maloney 99] **A. Malhotra and M. Maloney (eds.)**: XML Schema Requirements. *W3C Note*. February 15, 1999. `http://www.w3.org/TR/NOTE-xml-schema-req`

[Megginson 98] **David Megginson (ed.)**: SAX - The Simple API for XML. `http://www.megginson.com/SAX/index.html`

[Pemberton et al. 99] **S. Pemberton et al.**: XHTML 1.0: The Extensible HyperText Markup Language. A Reformulation of HTML 4.0 in XML 1.0. *W3C Working Draft*, March 4, 1999. `http://www.w3.org/TR/1999/WD-html-in-xml-19990304`

[Robie et al. 98] **J. Robie, J. Lapp, and D. Schach**: XML Query Language (XQL). in: *Proceedings of the W3C Query Language Workshop (QL-98)*, Boston, MA, December 3-4, 1998. `http://www.w3.org/TandS/QL/QL98/pp/xql.html`

[Robie 99] **J. Robie (ed.)**: XQL (XML Query Language). Working draft. August 1999. `http://metalab.unc.edu/xql/xql-proposal.html`

[Welty, Ide 99] **C. Welty and N. Ide**: Using the right tools: enhancing retrieval from marked-up documents. in: *Journal Computers and the Humanities*. 33(10):59-84. April, 1999.

[XML/EDI-Group 99] **XML/EDI-Group**: XML/EDI, the E-Business Framework. `http://www.geocities.com/WallStreet/Floor/5815/`

Within the XML document structure, each circle in this illustration represents a node, which is called an XmlNode object. The XmlNode object is the basic object in the DOM tree. The XmlDocument class, which extends XmlNode, supports methods for performing operations on the document as a whole (for example, loading it into memory or saving the XML to a file. In addition, XmlDocument provides a means to view and manipulate the nodes in the entire XML document. Both XmlNode and XmlDocument have performance and usability enhancements and have methods and properties to A valid XML document is kind of like a stamp of approval that declares the document suitable for use in an XML application. You learn how to validate your own XML documents in Document Validation Revisited.Â The process of creating a schema for an XML document is known as data modeling because it involves resolving a class of data into elements and attributes that can be used to describe the data in an XML document. Once a data model (schema) is in place for a particular class of data, you can create structured XML documents that adhere to the model. The real importance of schemas is that they allow XML documents to be validated for accuracy. Keywordsâ€”XML-Tree, XML structure Tree, Simple API for XML (API), Document Object Model (DOM) Compressing XML, XML Integration. View. Show abstract. The core contribution of this work is that the proposed approach is more formal and can extract richer and more natural domain semantics from RDBs compared with the existing solutions.Â Then it performs an automatic RDB-to-OWL schema translation by following a set of predefined translation rules that are based on the conceptual correspondences between RDB schema and OWL DL ontology. Our prototype implementation and case studies show that the proposed approach is feasible and effective.