How to Structure and Access XML Documents With Ontologies

Michael Erdmann and Rudi Studer

Institut für Angewandte Informatik und Formale Beschreibungsverfahren (AIFB)
Universät Karlsruhe (TH)
D-76128 Karlsruhe (Germany)
`http://www.aifb.uni-karlsruhe.de/WBS`
e-mail: {`erdmann`|`studer`}`@aifb.uni-karlsruhe.de`

Abstract

Currently dozens of XML-based applications exist or are under development. Many of them offer DTDs that define the structure of actual XML documents. Access to these documents relies on special purpose applications or on query languages that are closely tied to the document structures. Our approach uses ontologies to derive a canonical structure, i.e. a DTD, to access sets of distributed XML documents on a conceptual level.We will show how the combination of conceptual modeling, inheritance, and inference mechanisms on the one hand with the popularity, simplicity, and flexibility of XML on the other hand leads to applications providing a broad range of high quality information.

Key words: Ontologies, XML, Semantic Queries

1 Introduction

The Extensible Markup Language (XML) [3] is currently on the way to conquer the web and especially the electronic business (cf. e.g. [23]). 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. The success of XML is primarily based on its flexibility: everybody can write a document type definition (DTD) to define the structure of XML documents that represent information in the form he or she desires. If partners (e.g. in the electronic business) agree on a common DTD, documents can be created, transported, imported, and interpreted in a consistent way, preserving the semantics the sender intended. In the more general case, where XML sources are presented (in the web) without an explicit agreement on (not to mention a formal definition of) the semantics of certain tags and document structures, the task of correctly interpreting the data is more difficult. If information of several different sources should be integrated, correct assumptions about the meaning of certain elements are crucial for successful information retrieval.

In a survey of XML vocabularies for the electronic business [16] several industrial standards for electronic commerce based on XML are listed such as standards for electronic catalogs or representing business transactions. The survey shows that these standards have some amount of overlap. Applications and enterprises that need to work with several of these standards are forced to translate between different information sources by providing customized routines implementing the integration task.

Thus one can say, the biggest advantage of XML (its extensibility) is at the same time its biggest handicap. XML is so flexible that XML documents cannot be automatically provided with an underlying semantics, except the hierarchical information that is encoded in the mark-up tags. Different tag sets, different names for elements or attributes, or different document structures in general aggravate the task of accessing the represented information in a useful manner. Applications not only have to be aware of the DTD defining a class of documents, they must also be informed about the underlying semantics of tags and the meaning of the document structure. But this semantics is outside the scope of XML and related standards, possibly with the exception of RDF and RDFS (cf. [17] and [4]). In this paper we will present an approach to enable integrated access to different XML document types via the use of ontologies. An ontology [12] provides a formal, shared specification of concepts, their relationships, and other realities of some domain. This knowledge is independent of any representation, i.e. ontologies define semantics on a conceptual level, albeit each ontology has to be formulated in some representation language. Our approach fulfills the task of information integration exactly on this conceptual level in that it lifts information from the syntactic or representational level to the more abstract level of concepts and relationships. We will provide information integration capabilities to XML processing by defining a semantic underpinning for XML documents based on ontologies. Thus, the content of these documents can be uniformly accessed and integrated.

The paper is structured as follows: Section 2 shows how access to XML documents is currently realized by XML query languages and how adding semantics to XML can solve some of the problems arising here. The framework for doing this is set up by ONTOBROKER, an information integration and retrieval system which is briefly presented in section 3. 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 Query Answering

Although the focus of this paper does not lie in answering queries to single XML documents but in offering integrated access to these document sets via an ontology, this comparison of queries with and without ontology support is quite illustrative of the power of ontologies for integration tasks. This section will describe current ways of accessing semi-structured data in general and XML documents in particular, discuss deficits of these approaches, and will show how adding semantics to XML can overcome these deficits.

Access to semi-structured data is currently realized by a number of query languages, e.g. Lorel [1], Lorel for XML [11], XML-QL [8], or XQL [20,21]. 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) get Peter and Hans as answers.

```
<skills>
  <people>
    <person>
      <name>Peter</name>
      <know-how>SGML</know-how>
    </person>
    <person>
      <name>Hans</name>
      <know-how>SGML</know-how>
      <know-how>Perl</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 assumption 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
```

A combined query achieving the expected results can be formulated in another XML query language, namely XQL [20]. 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 official '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 hand crafted 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 P 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 [15] to a knowledge base (cf. section 3.1). The facts in this central knowledge base are collected from the same XML documents that were addressed by the XML-QL and XQL queries. The knowledge base 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' (possibly complex and varying) structures. 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 relationship between the ontology and the structure of XML documents has to be established. In section 4, we will show how a document type definition (DTD) can be derived from an ontology. Thus direct links between XML tags and ontological terms can be manifested.

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 Tom is a programmer and this fact is represented by the following addition to the above XML document:

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

Given this situation the "traditional" languages for querying XML become even more impractible, i.e. Tom 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 Tom even from the altered document while the other queries would not.

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 or at least if the information seeker is aware of the document structure. To locate relevant data he cannot specify his information needs conceptually (even if elaborated path expressions 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 directly be made available to information seekers.

In [9] an approach is presented that enhances the XML query language XQL [21] with ontological knowledge by preprocessing queries. XQL queries are rewritten and only the translated queries are answered by an XQL query processor. The results basically consider the concept hierarchy, thus, query rewriting is a partial solution to the mentioned deficits of XML query languages. Our approach enables true semantic queries to the contents of collections of XML documents and relieves the information seeker from the burden of knowing the structure of documents more extensively than realized in [9]. The ontology provides a conceptual view to XML documents. The information contained therein is integrated, enhanced (via axioms in the ontology, e.g. inheritance), and made accessible in a uniform manner for the user.

3 Defining the Context

3.1 ONTOBROKER

The last section presented current ways to access information stored in XML documents and some drawbacks of these approaches, resulting from the lack of a schema level or an ontology. In this section we will describe our system ONTOBROKER [1] that is built around the notion of ontology as its central pillar. System development started 1997. Since then ONTOBROKER has been constantly enhanced and extended. The work presented in this paper is one contribution to the further development of ONTOBROKER. This approach to intelligent information integration and retrieval is briefly presented in this section to provide the framework in which we will use ontologies to access information contained in XML documents. For a more comprehensive description of ONTOBROKER confer [7].

The ONTOBROKER project uses techniques such as 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 (in a proprietary format), to collect this data, store it centrally in a knowledge base, and to make the populated knowledge

[1] `http://ontobroker.aifb.uni-karlsruhe.de`

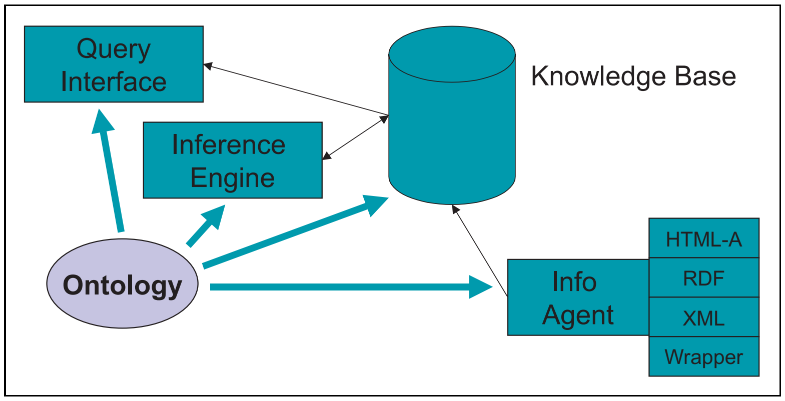

Fig. 1. A Sketch of ONTOBROKER's architecture.

base accessible through query facilities. In the knowledge base facts are stored in Frame Logic [15], an object oriented and logic-based language from the deductive database community. Underlying all parts of ONTOBROKER is an ontology that defines the vocabulary for annotating documents, for formulating queries, and for structuring the knowledge base. The ontology is a conceptualization of a domain and is formalized in Frame Logic as well. 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 interface allows information seekers to pose queries to the knowledge provided by ONTOBROKER. The knowledge base contains all basic facts and can be realized by powerful data-base management systems, thus large amounts of information 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 for collecting the raw facts from distributed sources. It is composed of modules that realize the gathering of different kinds of documents. The info agent can access (i) HTML documents annotated with ONTOBROKER's metadata language HTML-A (a slight extension of HTML), (ii) it realizes wrappers to multiple instance documents, to databases, or other wrappable documents, (iii) it may access RDF metadata [6], 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 the 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 interface 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. In a similar way the Observer system [19] uses ontologies to access and integrate information stored in distributed and independent databases.

The architecture illustrates that the info agent abstracts from the concrete realization of the information resources. The kind of resource is irrelevant for all other components of the ONTOBROKER architecture. Users can access information stemming from resources represented by any of the supported document types in a uniform way. Likewise the info agent will abstract from different document structures of XML documents.

How ONTOBROKER's info agent maps the structure of XML documents to an ontology follows in the next subsection and in section 4.

3.2 The Role of DTDMAKER *in* ONTOBROKER

ONTOBROKER can retrieve and integrate information stored in XML documents because its info agent provides mappings between the structure and contents of these documents and conceptual entities of an ontology. The information stored in XML must be associated with the correct parts from the ontolgy. This association must be hand coded if there does not exist a canonical relationship between XML structure and ontology.

A prerequisite for making information retrieval from XML documents possible in ONTOBROKER is the definition of a domain ontology. In our example and in the ONTOBROKER project in general this ontology is formalized in Frame Logic. This Frame Logic representation of an ontology serves as input to a translator that automatically derives canonical document type definitions (DTDs) for XML documents (cf. section 4). This component is called DTDMAKER. It produces DTDs that define a rather unrestricted class of documents, i.e. DTDMAKER defines relatively little constraints on the structure of the documents. But these restrictions are sufficient to enable a translation of knowledge encoded in XML documents (that conform to such a DTD) to facts in the knowledge base that adheres to an ontology.

The info agent gathers XML documents and imports their contents into the knowledge base. This import process needs the ontology to retranslate the representation of instances of ontological concepts and relationships from XML to the knowledge base. If the documents conform to the DTD that is produced by DTDMAKER the direct relationship to the ontology makes this task a straightforward one. An overview of the relationship between the ontology, the DTD, and XML instances is given in figure 2.

Even if XML documents do not conform to the derived DTD and, thus, do not embody this direct relationship to the ontology, they can be accessed by ONTOBROKER. But since queries are formulated in terms of the ontology, mappings from these document structures to the ontology have to be formulated. In this case the

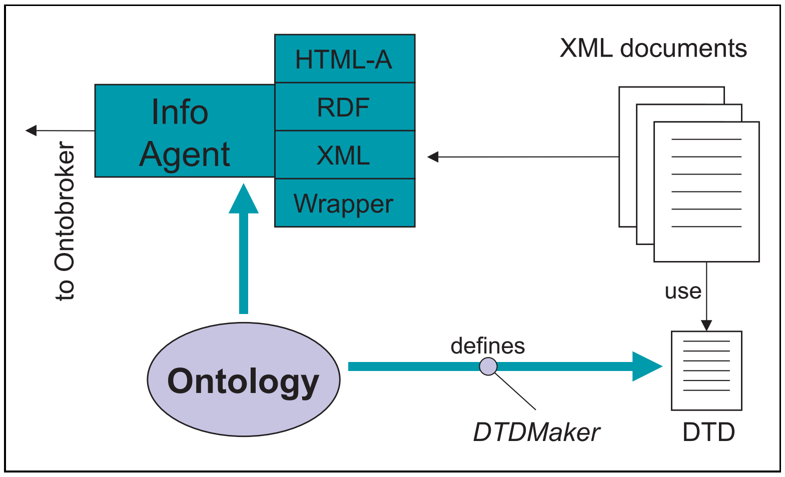

Fig. 2. Relationship between ontology, DTD and XML.

DTD generated by DTDMaker can be viewed as a canonical XML document structure that can serve as a reference. Other XML structures and vocabularies can be mapped into this canonical form, e.g. via XSLT stylesheets that contain the necessary templates, before the info agent of Ontobroker can import them into the knowldedge base.

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. The provision of a DTD has the advantage that any validating XML application can access the DTD, can check the XML document for structural validity, and, hence, to some extent also for semantical validity. The DTD is produced 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, ontologies and DTDs. 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). See figure 3 for an overview of these different levels of applications for XML documents. Generating a DTD from an ontology means that to some extent valid XML documents (complying to the generated DTD) are also valid with respect to the ontology.

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

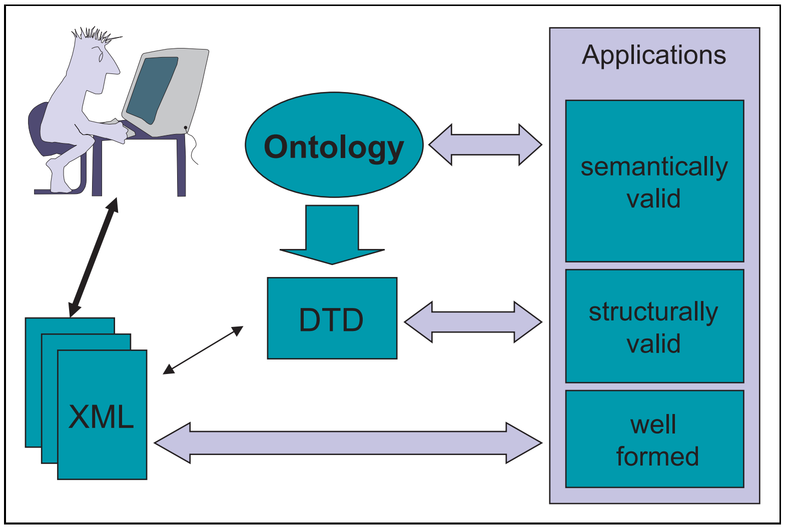

Fig. 3. Application scenario for (ontology based) XML documents.

4.1 General Functionality of DTDMAKER

The tool DTDMAKER takes an ontology in FLogic [2] as input and translates certain parts of the ontology into structural descriptions in a DTD. The general idea of DTDMAKER is as follows:

- Each concept from the ontology is mapped to an element type in the DTD.
- For each attribute of these concepts DTDMAKER defines a subelement and an XML attribute for the concept element.
- 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`.

All further knowledge that might be encoded in an ontology does not get transferred into the resulting DTD.

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 additional information about which concepts become elements and which are modeled via the `type` attribute. These kinds of variations must be handled when mapping arbitrary XML documents to ontological terms.

[2] Conceptually, the representation language of the ontology is unimportant, as long as concepts in a hierarchy and relationships between concept can be defined.

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

Person[name =>> STRING; email =>> STRING; editor =>> Book;
        publication =>> Publication; address =>> STRING].
Employee[employeeNo =>> STRING].
AcademicStaff[supervises =>> PhDStudent].
Researcher[cooperatesWith =>> Researcher].
Student[studentID =>> NUM].
PhDStudent[supervisor =>> AcademicStaff].
Publication[author =>> Person; title =>> STRING;
        year =>> NUM; abstract =>> STRING].
Book[editor =>> Person].
JournalArticle[journal =>> Journal; firstPage =>> NUM;
        lastPage =>> NUM].
Journal[editor =>> Person; volume =>> NUM; number =>> NUM;
        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].
```

Table 1
Example Ontology of persons and puclications based on [2].

The example ontology given in table 1 is a subset of an ontology that specifies among other things the domain of researchers and publications [2] . It 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 `NUM`, 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 axioms. These axioms 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 table 2 for a larger portion of the DTD). Since inheritance is a central feature of ontologies that is not supported in XML [3] 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.) *C1*, *C2*, *C3*, *C4* 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|...">
```

[3] XML schemas provide a partial solution to this problem, cf. [18,24], and section 5 for a discussion

DTDMAKER maps concepts from the ontology to element types in the DTD, i.e. for each concept an element type is defined. The content models 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|author|title|year|abstract|editor)*>
```

In an XML document this element may be instantiated 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 4.3 for a discussion of the strictness of the DTD).

Additionally, attributes from the ontology are expressible as XML attributes as well, i.e. ontological attributes are mapped to corresponding XML attributes, e.g.

```
<!ATTLIST Book
  year      CDATA #IMPLIED
  abstract  CDATA #IMPLIED
  title     CDATA #IMPLIED
  author    CDATA #IMPLIED
  editor    CDATA #IMPLIED>
```

For some attributes this attribute notation seems more natural while for others the subelement notation is appropriate or even mandatory (e.g. for a `Publication` that can have multiple `author`s), e.g.

```
<Book title="The XML Handbook" year="1998">
  <author>Charles F. Goldfarb</author>
  <author>Paul Prescod</author>
</Book>
```

The last step for mapping an ontology to a DTD is the specification of the content model of the attribute elements. This content specification is governed by the attribute value ranges as defined in the ontology. The attribute `title` of `Publication` has the atomic value range `STRING` and thus does not specify a relationship to another concept. Consequently this `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 `Researcher`s is expressed with the `cooperatesWith` attribute. The element type definitions produced by DTDMAKER match these intentions:

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

An `author` element may have one or more `Person` subelements that represent the `author`s of the embracing `Publication`. Note, that 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 | Student |
      AcademicStaff | PhDStudent | Researcher)* >
```

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 immediately compatible with the designed ontology. A part of the DTD derived from the ontology (see above) is presented in table 2. The presented DTD contains four 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. `email` and `publication` are attributes of `Person`. (iii) In the next section optional XML attributes are defined that represent ontological attributes of the respective concept. The interpretation of e.g. the `title` subelement of `Book` and the `title` attribute of a `Book` element are identical. (iv) In the last section the content model 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 instance. 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

```
<!-- entities for realizing the is-a hierarchy -->
<!ENTITY % Person "Person | Employee | Student |
      AcademicStaff | PhDStudent | Researcher" >
<!ENTITY % Researcher "Researcher | PhDStudent" >
<!ENTITY % Publication "Publication | Book | Article |
      JournalArticle | Journal" >
<!ENTITY % Article "Article | JournalArticle" >
<!ENTITY % Book "Book" >

<!-- element declarations for ontology concepts -->
<!ELEMENT Person (#PCDATA | name | email | adress |
      publication | editor)*>
<!ELEMENT Researcher (#PCDATA | name | email | address |
      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)*>

<!-- ATTLIST declarations for ontology attributes -->
<!ATTLIST Person
  address     CDATA #IMPLIED
  phone       CDATA #IMPLIED
  email       CDATA #IMPLIED
  publication CDATA #IMPLIED
  name        CDATA #IMPLIED
  editor      CDATA #IMPLIED>

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

Table 2
Part of the generated DTD based on the ontology of table 1.

state:

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

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

```
<Researcher employeeNo="247">
  <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>
      <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-onto.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`, and possibly related concepts. To be able to represent an arbitrary set of concepts within a single document in addition to what has been stated up to now, the DTD contains another single element type called `Container` that might embrace any concept elements without restrictions concerning order or number.

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 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. Exactly this connectedness enables the integration of information.

4.3 DTD $\neq$ 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 ontology. 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, at least for the representation of ontology attributes as XML elements. So, any possible subelement (depending on the attributes' value ranges) must be added to the attributes' 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`. Following good ontology modelling practice these two attributes would have been named differently, e.g. `editorOf` and `editedBy`, respectively. Nevertheless, the element declaration for `editor` contains the appropriate value types `%Book;` and `%Person;`. 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 [24] that provide a higher expressiveness than DTDs. But currently, the XML Schema specifications are still in a working draft state. Moreover, following good ontology modelling practice one would not use the same name for semantically different relationships. Hence, this conflict can be easily avoided in most cases.
- 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. In the area of information integration assumptions should be as few as possible. Making DTDMAKER more strict would allow `PCDATA` only for attributes with atomic value ranges, but such an approach would increase the number of assumptions.
- 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 restrictions as possible were coded into the ontology. Although the XML specification defines three cardinalities: (i) exactly once, (ii) optional, and (iii) zero or many times DTDMAKER does not exploit this feature. 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 model the cardinalities in closer conformance to the ontology, but again would add further assumptions on the structure of documents (see above).
- DTDs are much weaker for representing ontological knowledge than any logic oriented or OO modelling language. Thus, XML only serves the purpose of serializing *instances* of concepts and relations, modelled in such richer languages. The DTD defines the admissible structure of these instance documents. Therefore, the lack of expressibility of DTDs, that prohibits the formulation of e.g. axioms of any kind or of Description Logic-like class expressions is not relevant

for this application. DTDMAKER is not meant to translate the whole ontology into a DTD but only those parts necessary for representing instances, i.e. names of concepts and attributes, the concept hierarchy and the relationships between concepts. The true semantic processing has still to be carried out outside of XML, e.g. within ONTOBROKER.

5 Discussion

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 sets of differently structured XML documents rather independently of their actual linear representation. The ontology provides a shared vocabulary that integrates the different XML sources, that makes the information uniformly accessible, and thus 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 (or, under certain circumstances, that pursued by [9]). 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 [8], Lorel for XML [11], XSL [5], and XQL [20]) 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 [17,4] is manifold. Both define an ontology (or schema) that is used to structure the contents of XML documents. We use Frame Logic and automatically derive a DTD that constrains the possible document structures. In the RDF/RDFS context the schema 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 and logic-based tradition where each class has its own local set of attributes, whereas in RDF attributes are global and not uniquely associated with a class. The expressibility of Frame Logic is richer than of RDF/RDFS, since in Frame Logic arbitrary axioms can be formulated to derive new information. This is currently not possible in RDF/RDFS.

A similar approach for combining ontologies and distributed information sources can be found in the Untangle project. In [13] and [22] it is described how an SGML DTD can be translated into the core of an ontology represented in CLASSIC, a Description Logics system. In another project, Infosleuth [14], an ontology is built based on the database schemas of the sources that should be accessed. [14] describes a stepwise process of achieving the common vocabulary. Both approaches integrate information based on an ontology but neither access XML documents nor care about the impact an ontology could have on the structure of the sources, e.g. when designing a DTD.

Another approach for integrating information from various data sources is the Strudel system [10]. Strudel's mediator component is based on a semistructured data model and provides an integrated view of the sources by evaluating STRUQL-queries. Thus the Strudel system lacks the semantic level that is provided in our approach by the domain ontology.

XML schemas will go in a similar direction of raising the level of abstraction when designing the structure of documents. [18] define a list of requirements for a future standard for XML schemas that are the designated successor of DTDs. The current working drafts produced by the W3C Schema Working Group can be found in [24]. In XML Schema elements can inherit attributes and content models from 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 consider schemas as an additional source of information as soon as the standardization process will be a little more advanced.

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. [22,14]). 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.

References

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

[2] R. Benjamins, D. Fensel, S. Decker, A. Gomez Perez, KA2. Building ontologies for the internet: A midterm report, *International Journal of Human Computer Studies*, 51(3), 1999. pp. 687-712.

[3] 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`

[4] 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`

[5] S. Deach ed., Extensible Stylesheet Language (XSL) specification, *W3C Working Draft*, March 27, 2000.
`http://www.w3.org/TR/xsl`

[6] S. Decker, D. Brickley, J. Saarela, and J. Angele, A query and inference service for RDF, *Proceedings of the W3C Query Language Workshop (QL-98)*, Boston, MA, December 3-4, 1998.

[7] S. Decker, M. Erdmann, D. Fensel, and R. Studer, Ontobroker: Ontology-based access to distributed and semi-structured information, R. Meersman et al. eds., *Semantic Issues in Multimedia Systems*, Kluwer Academic Publisher, Boston 1999.

[8] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu, A query language for XML. *Proceedings of Eigth International World Wide Web Conference (WWW8)*, 1999.

[9] M. Erdmann, and S. Decker, Ontology-aware XML queries, submitted, 2000.

[10] M. Fernandez, D. Florescu, J. Kang, and A. Levy, Catching the boat with Strudel: Experiences with a web-site management system, *Proceedings of the 1998 ACM International Conference on Management of Data (SIGMOD'98)*, Seattle, WA, 1998.

[11] R. Goldman, J. McHugh, and J. Widom, From semistructured data to XML: Migrating the Lore data model and query language *Proceedings of the 2nd International Workshop on the Web and Databases (WebDB '99)*, Philadelphia, Pennsylvania, June 1999.
`ftp://db.stanford.edu/pub/papers/xml.ps`

[12] T.R. Gruber, A translation approach to portable ontology specifications, *Knowledge Acquisition*. vol. 6, no. 2, 1993. pp199-221

[13] 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.

[14] V. Kashyap, Design and creation of ontologies for environmental information retrieval, *Proceedings of the 12th Workshop on Knowledge Acquisition, Modeling and Management (KAW'99)*, Banff, Canada, October 1999

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

[16] A. Kotok, A survey of XML business data exchange vocabularies, *XML.com*, Februrary 23. 2000.
`http://www.xml.com/pub/2000/02/23/ebiz/index.html`

[17] 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`

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

[19] E. Mena, V. Kashyap, A. Illarramendi, and A. Sheth, Domain specific ontologies for semantic information brokering on the global information infrastructure, N. Guarino, ed. *Formal Ontology in Information Systems*. IOS Press, 1998.

[20] J. Robie, J. Lapp, and D. Schach: XML Query Language (XQL), *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`

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

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

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

[24] XML Schema Working Group, XML Schema part 1: Structures. and XML Schema part 2: Datatypes. *W3C Working Draft*.
`http://www.w3.org/TR/xmlschema-1/`
`http://www.w3.org/TR/xmlschema-2/`