Extreme Markup Languages 2002

Montréal, Québec
August 6-9, 2002

Drawing inferences on the basis of markup

C. M. Sperberg-McQueen
World Wide Web Consortium

David Dubin
University of Illinois at Urbana/Champaign

Claus Huitfeldt
Avdeling for kultur, språk og informasjonsteknologi

Allen Renear
University of Illinois at Urbana/Champaign

Abstract

Various authors have sketched out proposals for identifying the meaning, or guiding the automated interpretation, of markup, sometimes with the goal of using the information expressed by markup to guide the extraction of information from documents and using it to populate reasoning engines. We describe one approach to the problems of building a system to perform such a task.

Drawing inferences on the basis of markup

Table of Contents

EXTREME MARKUP LANGUAGES 2002

Drawing inferences on the basis of markup

C. M. Sperberg-McQueen, David Dubin, Claus Huitfeldt, and Allen Renear

§ 1 Introduction

Text encoding has traditionally promised to make explicit our understanding of (or: theories about) a document ([Coombs et al. 1987] [Coombs et al. 1987], [Sperberg-McQueen 1991]). Well-designed SGML/XML markup languages such as Docbook [Walsh/Muellner 1999] or the TEI [ACH/ACL/ALLC 1994] appear to be fairly successful at this. However, a close look reveals that there are limits to the degree of explicitness that can be achieved with current encoding languages. The problem may be traced to the fact that although SGML/XML-based markup languages provide explicit rules for syntactic well-formedness and validity, they provide nothing analogous for semantic correctness. As a result they are reasonably well suited for generating data structures, but are not, at least not without further development, as effective at expressing interpretations [Renear 2001]. And many developers of programming systems have wished they had a mechanism for specifying, more exactly than XML 1.0 DTDs allow, exactly what application data structures or what tables and columns in a SQL database are to be built from XML data when it is received.

Our immediate area of concern is the task of providing a clear, explicit account of the meaning and interpretation of markup.

Scores of products and projects which use XML and SGML assume implicitly that markup is meaningful, and use its meaning to govern the processing of the data. A number of authors have described systems for exploiting information about the meaning or interpretation of markup; among those most relevant to the work described here are [Simons 1997], [Simons 1999], [Welty/Ide 1997], [Ramalho et al. 1999], [Sperberg-McQueen et al. 2001a], [Sperberg-McQueen et al. 2001b], and [Thompson 2001].

There are a variety of reasons to be interested in this problem. Better understanding of the meaning properly assigned to markup seems certain to make it easier to write better, smarter tools for creating, managing, and exploiting markup. [Ramalho et al. 1999] make a strong case that it will enable substantially improved validation and quality assurance, enabling automated systems to detect a large number of errors in the use of markup or in the data which cannot be detected by purely syntactic or datatype-oriented methods. [Welty/Ide 1997] argue that a more formal approach to markup semantics will dramatically improve information retrieval. Even if the kind of logical inference they have in mind proves too time-consuming to perform at query time, it may be helpful in reducing inessential variation in markup practice within a collection, or in masking the variation in markup within heterogeneous collections [Schatz et al. 1996]. For non-documentary uses of XML, the meaning of markup may be seen in its mapping to target data structures — or rather the meaning can be seen in the range of target application data structures the markup may legitimately be mapped to; this position has been put succinctly by Henry Thompson in discussions of earlier work on this topic, and underlies the work reported in [Thompson 2001]. On a purely practical level, experience with systems of the kind we describe here can be expected to provide useful information about how to make the documentation of markup applications clearer and more useful.

Our main motivation for this work, however, is not so much its manifold practical applications as its intrinsic interest.

In earlier work [Sperberg-McQueen et al. 2001a], we proposed to identify the meaning of markup in a document as the set of inferences licensed by it[1] and outlined a 'straw man' proposal for defining the proper interpretation of markup in a given markup language and formulating certain simple rules for mechanically generating the proper inferences from documents marked up in that language. Implementation of the straw man proposal demonstrated that the straw man proposal is too simple to explain real markup languages; its rules lead to a number of false inferences. In [Sperberg-McQueen et al. 2001a], we sketched in general terms how a better account of meaning and interpretation in markup could be constructed; this paper reports on work toward the concrete realization of one part of that 'framework' model and will outline some of the problems encountered in specifying the inferences licensed by commonly used DTDs.

§ 2 The problem

The framework outlined in [Sperberg-McQueen et al. 2001a] includes:

- some representation of the document, probably in a form which can be used with some existing inference system. In our current work, we use a Prolog interpreter as our inference engine.
- a set of *skeleton sentences*. These are sentence schemata or patterns which can be used to form sentences which express the meaning of each construct in the markup language. A skeleton sentence is like a natural-language sentence, or a Prolog predicate, or an expression in some other formal system, in which certain words or items have been replaced by blanks. When the blanks are filled in properly, a normal sentence in the language is the result. (Some readers will be reminded of the game marketed a few years ago under the name Mad Libs.) In the systems we describe here, the blanks are typically to be filled in with information from the document itself, and each blank is associated with a *deictic expression* showing how to fill it in.[2]
- some language, possibly a rather small one, in which to write the *deictic expressions* which can be associated with the blanks in the skeleton sentences; in common languages like Docbook, HTML, and TEI, it is easy to foresee the need for expressions meaning "the contents of this element", "the value of this attribute", "the nearest ancestor of type *bibl*", "the value of the *xml:lang* attribute on the nearest ancestor element which has a value for it", etc.
- some categorization of predicates according to the rules governing inferences from them. For example, some properties of elements are inherited by descendant elements, and others are not — a satisfactory account of the meaning of markup should capture this general distinction and others like it.
- some generic routines for generating statements about the document by extracting suitable information from the document and using it to fill the blanks in the skeleton sentences, thus forming full sentences in the target language.
- (optionally) rules allowing further inferences. These are often not closely tied to specific markup of specific document instances, but may express more general rules about properties expressed by markup (e.g., "if something is an author, and not identified as a corporate author, then it is a person", or "if something is a person, then it is human").

Several of these components will be discussed separately below.

2.1 *An example*

To illustrate the basic ideas, let us consider as an example the sample purchase order used in the Primer for W3C XML Schema 1.0 [Fallside 2001]. If the meaning of markup is to be found in the set of inferences we can draw from it, what inferences can be drawn from the sample purchase order, and how?

```
<?xml version="1.0"?>
<purchaseOrder orderDate="1999-10-20">
    <shipTo country="US">
        <name>Alice Smith</name>
        <street>123 Maple Street</street>
        <city>Mill Valley</city>
        <state>CA</state>
        <zip>90952</zip>
    </shipTo>
    <billTo country="US">
        <name>Robert Smith</name>
        <street>8 Oak Avenue</street>
        <city>Old Town</city>
        <state>PA</state>
        <zip>95819</zip>
    </billTo>
    <comment>Hurry, my lawn is going wild!</comment>
    <items>
        <item partNum="872-AA">
            <productName>Lawnmower</productName>
            <quantity>1</quantity>
            <USPrice>148.95</USPrice>
            <comment>Confirm this is electric</comment>
        </item>
        <item partNum="926-AA">
            <productName>Baby Monitor</productName>
            <quantity>1</quantity>
            <USPrice>39.98</USPrice>
            <shipDate>1999-05-21</shipDate>
        </item>
    </items>
</purchaseOrder>
```

The sample document above shows an order for a lawnmower and a baby monitor; the items are to be shipped to one Alice Smith of Mill Valley, California, and billed to one Robert Smith of Old Town, Pennsylvania.

In the following sections, we will illustrate the discussion with examples drawn from this document.

2.2 *A Logical Approach*

Our general approach to the problem of representing the meaning of markup is the construction of a formal logical system. This system will serve both a theoretical objective: illuminating *how* document markup licenses inferences; and one practical one: supporting the mechanical calculation of the inferences.

As with most efforts to formalize a domain area we do not begin from scratch, but rather accept the expressive devices and deduction rules of an existing system of predicate logic, and supplement those with (i) constants distinctive to our domain of interest; and (ii) a set of 'axioms' or 'premises' which reflect not fundamental truths about logic but rather fundamental facts about markup about which we wish to reason.

In translating crucial propositions from another notation into a more purely logical notation, we are following the practice of most work on programming-language specification and semantics; cf. [Guttag/Horning 1993], which explicitly makes it a goal of the Larch system to provide a basis (i.e. an axiomatic basis) for reasoning about programs. The notion of the meaning of some sentence as the set of inferences to be drawn from it is also borrowed from this field; it came to our attention from [Turski/Maibaum 1987].

2.3 *Kinds of sentences*

We provisionally identify several kinds of sentences in our system. In this section we list these kinds and give intuitive characterizations of each. We also give examples of what such sentence would mean, using a stylized English sentences that corresponds naturally and unambiguously to the expressions of predicate logic. Then the following sections we explore Prolog representations that support inferencing. Rigorous formal characterizations of these sets of sentences are underway and will be reported on in later publications.

image sentences	These directly translate the marked up document into the notation of the logical system. In practice, they constitute a more or less literal and

mechanical translation from a given markup notation (e.g., XML) into a set of sentences. These sentences ascribe certain properties — such as having a certain generic identifier or a certain attribute name and value — to certain objects which map 1:1 to the elements, attributes, and so on of the input document. Strictly speaking the objects so described are not XML elements but the result of mapping XML elements (or element information items) into our logical system. We will call these objects *images* of the document instance; the information items to which the images correspond we will call their *pre-images*. For our purposes, we regard the translation into images sentences as sufficient if by working from the logical form alone we can construct an XML document with the same infoset [Cowan/Tobin 2001] as the original document.

Examples of image sentences:

- *a* has the generic identifier "*authorname*".
- *a* has the content "Alan Turing".
- *a* has the value "de" for the *lang* attribute.

The graph structure of the document instance must also be represented if the image sentences are to capture the basic information set of the input document. The following are thus also examples of image sentences:

- The first child of *a* is *b*.
- *b* has an immediately following sibling; this immediately following sibling is *c*.

property rules

These associate specific properties — such as being a quotation, being in the German language, or being a name — with the generic identifiers and attribute names of a particular markup vocabulary. Where appropriate these rules will also specify the conditions for inheritance of the property.

Examples of property rules:

- If *x* has the generic identifier "*authorname*" then *x* is a *name-of-an-author*.
- If *x* has the value "de" for a *lang* attribute, then: *x* is in the German language, and any descendent *y* of *x* is in the German language unless there exists some element *z* which is both a descendent of *x* and either an ancestor of *y* or identical to *y* and *z* has a *lang* attribute with a value other than "de".

propagation sentences

These sentences result from applying a property rule to an image sentence:

Image sentence:

- *a* has the generic identifier "*authorname*".

Property rule:

- If *a* has the generic identifier "*authorname*" then *a* is a *name-of-an-author*.

Resulting propagation sentence:

- *a* is a *name-of-an-author*.

In more complicated cases the generation of all implied propagation sentences also requires applying the inheritance conditions indicated by the property rule:

Image sentence:

- *a* has the value “de” for a *lang* attribute.

Image sentence:

- *a* is the immediate parent of *b*.

We note in passing that *b* does not have an attribute-value specification for the *lang* attribute, and there is thus no image sentence giving that attribute any value for *b*.

Property rule:

- If *x* has the value “de” for a *lang* attribute, then: *x* is in the German language, and any descendent *y* of *x* is in the German language unless there exists some element *z* which is both a descendent of *x* and either an ancestor of *y* or identical to *y* and *z* has a *lang* attribute with a value other than “de”.

Resulting propagation sentences:

- *a* is in the German language; *b* is in the German language.

One may reasonably wonder whether the variables in the antecedent and consequent of property rules really should be ranging over the same set of objects, with the result that one and the same thing both, e.g., has a lang attribute and is in German. Currently we believe that this is a reasonable assumption and one which simplifies our system — allowing the hierarchical relations to continue to apply with being mapped to corresponding higher level relations. However, we are prepared to reevaluate this assumption as we accumulate more practical experience translating documents and calculating licensed inferences.

mapping rules These rules associate objects and properties in the propagation sentences with objects and properties in the application domain. Like the property rules, mapping rules express generalizations rather than specific claims about document instances.

Examples of mapping rules:

- If *x* is an article and *y* is a name-of-an-author and *x* is the parent of *y*, and *z* is denoted by *y*, then *z* authored *x*.
- If *x* is a meeting-name and *y* is an attendee-name, and *w* is denoted by *x* and *z* is denoted by *y*, then *y* attended *x*.

Note that in some cases the consequent of a mapping rule attributes a property to a document component, but in other cases no property is being attributed to the document component (at least directly), but only to the object that the component denotes.

application sentences These sentences are typically (but not necessarily) about objects and properties external to the document, such as authors, customers, prices.

They result from applying a mapping rule to one or more propagation sentences.

Examples of application sentences are:

- *a* is the author of (the article) *b*.
- *c* is a meeting.
- *a* attended *c*.

fundamental axioms

Axioms of the sort common to systems supporting commonsense reasoning. Examples:

- If *x* occurs before *y*, then *y* does not occur before *x*.
- If *x* occurs at the same time as *y* then *y* occurs at the same time as *x*.
- If *x* is a part of *y* and *y* is a part of *z*, then *x* is a part of *z*.
- if *p* is necessary and (if *p* then *q*) is necessary, then *q* is necessary.

Further discussion of the fundamental axioms is outside the scope of this paper.

world knowledge

These are sentences about the world or some part of it. They may be particular fundamental facts (water is a material substance, five is a number) particular contingent facts (the United States signed the Berne treaty), or they may be rules — in practical systems, business rules will be of particular importance here. Strictly speaking, these sentences, like the fundamental axioms, are also outside the scope of this paper, but they are a necessary part of the target system: without them, it is unlikely that many useful applications will be built.[3]

Some applications may not need to have all of these kinds of sentences present in the logical system. In some cases, the main value will lie in the application sentences and the theorems which will follow from combining them with the fundamental axioms and world knowledge. The image and propagation sentences will be present only in order to help in generating all the appropriate application sentences. In some cases, the image sentences will serve no purpose at all except the generation of the propagation sentences.

2.4 *Two approaches: inference and direct instantiation*

2.4.1 Inference-driven approach

We are exploring two approaches to the task of generating useful sets of inferences. The *inference-driven approach*, illustrated in figure1, uses all the different kinds of sentences outlined above. We will represent the description of the markup vocabulary and input document directly in the reasoning system (in the form of property rules, mapping rules, and image sentences) and use normal inference techniques to generate all the other sets of sentences. As shown in the figure, image sentences are derived from an XML document instance; property rules and mapping rules are derived from a formal tag set definition. From the image sentences and the property rules, an inference process derives the propagation sentences. From the propagation sentences and the mapping rules, an inference process derives the application sentences. From the application sentences, fundamental axioms, and world knowledge, an inference process derives further sentences.

2.4.2 Direct-instantiation approach

In the other approach, application sentences are generated directly by instantiating (filling in the blanks in) the sentence schemata or *skeleton sentences* described in the framework proposal of [Sperberg-McQueen et al. 2001a]. A system overview is given in figure 2. In this approach, XSLT

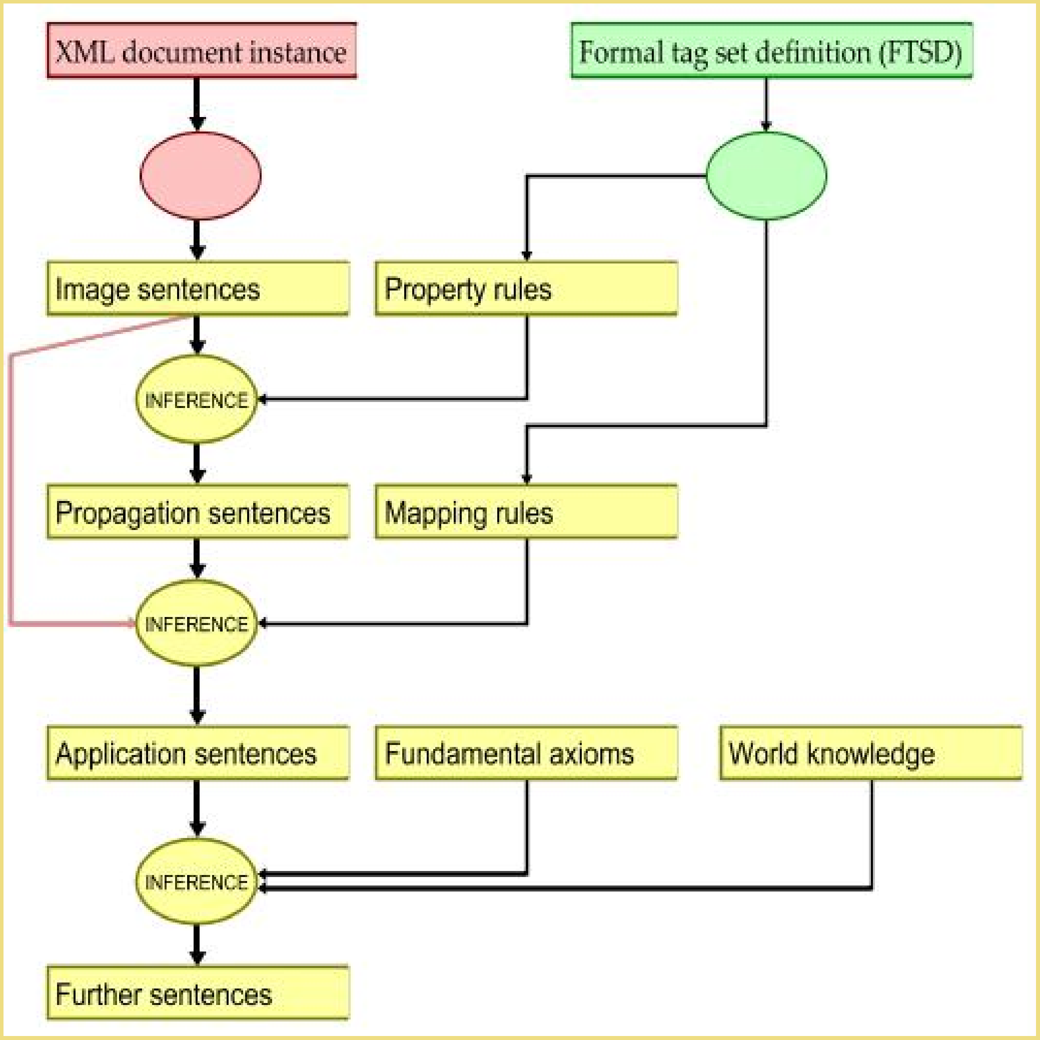

Figure 1 Data flow in the inference-driven approach

transformations are used to map directly from the XML document instance into application sentences.[4] The XSLT stylesheets are specific to particular document types, and they directly embody the property and mapping rules for that document type. This is not, however, the most compact or intuitive form in which to represent the property and mapping rules; in order to make the mapping and property rules accessible for human inspection and for reuse, however, they are formulated in declarative form in the formal tag set definition, and a second-level transformation is applied to generate the appropriate first-level transformation sheets which take a document instance as input and produce application sentences as output.

If it is desirable for auditing or other purposes, an instantiation-based system can also be used to generate image sentences, propagation sentences, and explicit property and mapping rules. An augmented data flow for such a system is shown in figure 3.

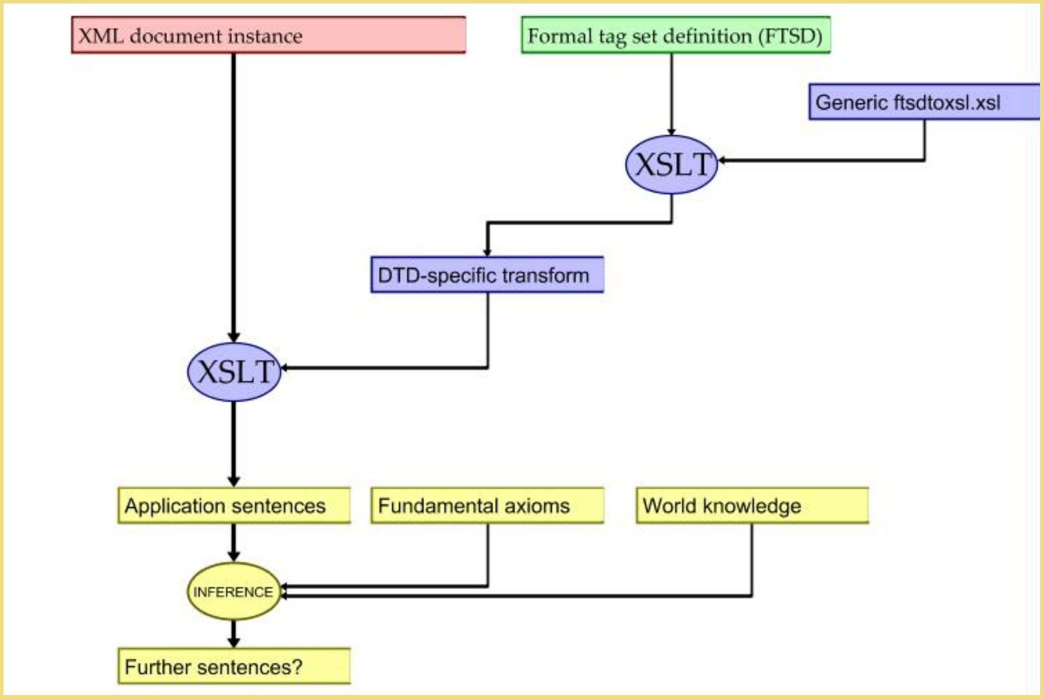

Figure 2 Data flow in the direct-instantiation approach

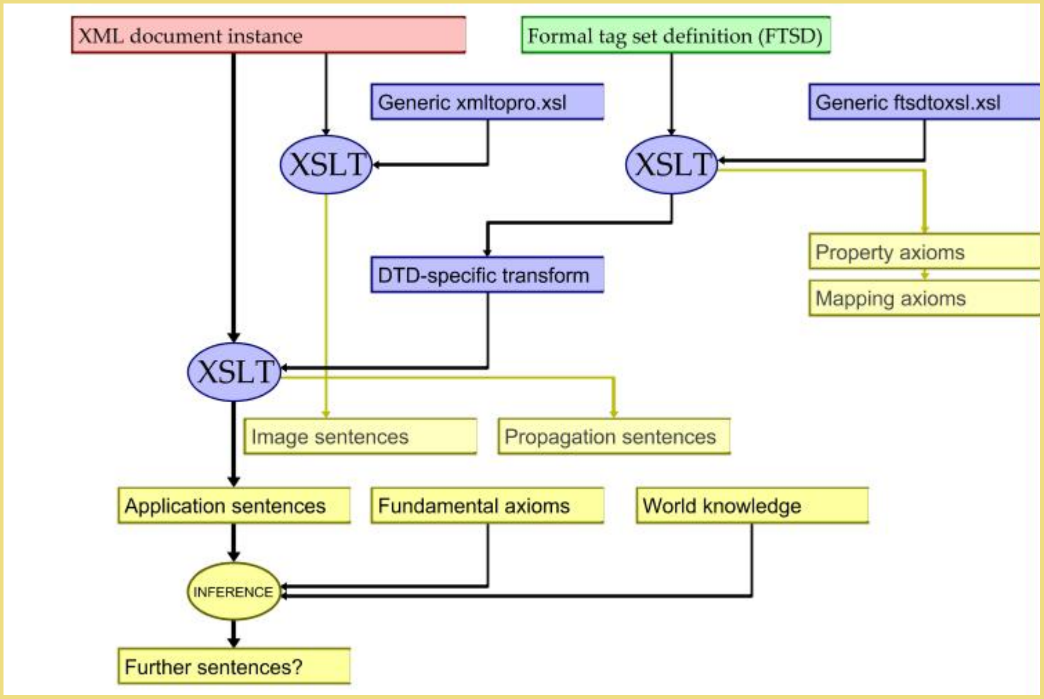

Figure 3 Data flow in the direct-instantiation approach, augmented

§ 3 Document representation

The *image sentences* described above provide a representation of the input document with sufficient detail to enable the original document to be recreated — or, more precisely, a document sufficiently similar to the original to have the same basic information set [Cowan/Tobin 2001]).

In this section, we describe our Prolog representation of image sentences. In earlier work [Sperberg-McQueen et al. 2001a], we described a different Prolog representation, but it is inefficient and exposes somewhat more of the internal implementation data structures than is desirable. The representation outlined here is based on several simple principles:

- The lower level implementation choices are encapsulated as thoroughly as possible. References to data structures are hidden behind Prolog variables, thus insulating the higher level predicates from changes we may later make at the lower levels.
- Identifiable entities, such as nodes, documents, or paragraphs are treated as objects with identities. They are referenced by unique identifiers, rather than by address or location.
- The W3C Document Object Model (DOM) includes methods for node navigation and attribute retrieval that can simplify our definitions. We emulate these methods in Prolog, though ours is not a full DOM implementation.

Less crucial from some points of view, but worth mentioning, is the fact that the Prolog implementation we use (SWI Prolog) provides an SGML/XML parser, which we call and whose results we use to create our representation. To allow portability to other Prolog implementations, none of the parser-specific predicates are essential to the basic system architecture, nor are they intimately coupled with the portable code. All functions performed by the parser could be handled by XSLT transformations that would write the image predicates as output. We have also defined XML serialization routines which operate on the predicates described below, so that the reasoning system can read and write XML directly.

We store facts about the XML tree in image sentences using predicates like the following:

```
node(node6).
gi(node6, quote).
parent(node6, node4).
first_child(node6, node7).
nsib(node6, node10).
attv(node6, lang, de).
content(node9, 'Die Welt ist alles, was der Fall ist.' ).
```

These might be paraphrased roughly as follows:

- **node(node6).** The object called *node6* is a node in the tree.
- **gi(node6, quote).** The generic identifier (sometimes called the *element type*) of *node6* is *quote*.
- **parent(node6, node4).** The parent of *node6* is *node4*.
- **first_child(node6, node7).** The first child of *node6* is *node7*.
- **nsib(node6, node10).** The following sibling of *node6* is *node10*.
- **attv(node6, lang, de).** *node6* has an attribute called *lang*, which has the value **de**.
- **content(node9, 'Die Welt ist alles, was der Fall ist.').** The content of *node9* is the string “Die Welt ist alles, was der Fall ist.”

The identifiers for nodes are generated automatically and carry no significance. The first few nodes of the purchase order example might look something like this:

```
node(node1).
gi(node1,purchaseOrder).
first_child(node1, node2).

attv(node1, orderDate, '1999-10-20').
node(node2).
gi(node2,shipTo).
parent(node2, node1).
first_child(node2, node3).
nsib(node2, node11).

attv(node2, country, 'US').
node(node3).
gi(node3,name).
parent(node3, node2).
nsib(node3, node5).

content(node4,'Alice Smith').
parent(node4, node3).
node(node5).
gi(node5,street).
parent(node5, node2).
nsib(node5, node7).

content(node6,'123 Maple Street').
parent(node6, node5).
node(node7).
gi(node7,city).
parent(node7, node2).
nsib(node7, node9).

content(node8,'Mill Valley').
parent(node8, node7).
node(node9).
gi(node9,state).
parent(node9, node2).
nsib(node9, node10).
```

From Prolog atoms of this kind, it is possible to derive other convenient predicates, such as:

```
children(node6, [node7, node8]).
attl(node11, [purpose='title', lang='la']).
psib(node10, node6).
child(node4, node6).
ancestor(node6, node1).
nearest_anc(node6, p, node4).
nearest_anc(node9, lang, 'de', node6).
```

These might be paraphrased as follows:

- **children(node6, [node7, node8]).** The object called *node6* has as children *node7* and *node8* in that order.
- **attl(node11, [purpose='title', lang='la']).** Node *node11* has attribute-value specifications for attributes *purpose* and *title*, with the values **title** and **la** respectively.
- **psib(node10, node6).** The previous sibling of *node10* is *node6*.
- **child(node4, node6).** *node6* is a child of *node4*. This is the inverse of the *parent* relation.
- **ancestor(node6, node1).** *node1* is an ancestor of *node6*.
- **nearest_anc(node6, p, node4).** The nearest ancestor of *node6* which has the generic identifier *p* is *node4*.
- **nearest_anc(node9, lang, 'de', node6).** The nearest ancestor of *node9* which has the value **de** for the attribute *lang* is *node6*.

§ 4 Propagation sentences

The sample purchase order has no attributes with inherited values, or with complex rules for calculating default values; the propagation-sentence layer of our system, which is intended to handle such cases, therefore has nothing much to do and can be passed over without comment.

§ 5 Application sentences and further inferences

Expressed in English, the markup of the example purchase order allows us to infer propositions in the application domain like the following:

- There exists an object (in this case an abstract object) of the type purchase order (which for convenience we will refer to by the name P-123).
- Purchase order P-123 was placed on 20 October 1999.
- There is someone named Alice Smith now able to receive mail at 123 Maple Street, Mill Valley, California. (For brevity's sake, we will refer to her as S-45 and to her address as A-45.)
- There is someone named Robert Smith now able to receive mail at 8 Oak Avenue, Old Town, Pennsylvania. (For brevity we will refer to him as S-46 and regard him as a customer.)
- The items on P-123 should be shipped to person S-45 at address A-45.
- Customer S-46 is to be billed for the items on purchase order P-123 at the prices indicated.
- Purchase order P-123 includes one (1) item of part number 872-AA.
- The item denoted by part number 872-AA is a lawnmower.
- Customer S-46 is to be billed $148.95 for the item denoted by part number 872-AA.
- Purchase order *P-123* includes one (1) item with part number 926-AA.
- The item denoted by part number 926-AA is a baby monitor.
- Etc.

In the typology given above, these propositions would all be expressed by *application sentences*.

We express application sentences in Prolog using a simple object-oriented language which defines objects and classes of objects, properties of objects and their values, and relations among objects, using the following predicates:

- **object(O)**: *O* has been instantiated as an object.
- **obj_class(O,C)**: Object *O* is of class *C*.
- **models(O,L)**: The real-world object modeled by *O* is represented at the syntax level by the XML elements which are the pre-images of the nodes in *L*.
- **class(C)**: *C* is a class of objects.
- **subclass(Sub,Super)**: Class *Sub* is a subclass of class *Super*. Subclasses can take the same properties and participate in the same relations as their Superclasses.
- **property_of(C,P,T)**: Objects of class *C* have property *P*, which is of type *T*.
- **opv(O,P,V)** Object *O* has value *V* on property *P*.
- **relation(R,L)** Relation *R* can hold among objects of classes listed in ordered list *L*.
- **relation_applies(R,L)** Relation *R* applies to the objects in the ordered list *L*.

Expressed in Prolog using these predicates, the application sentences listed above would take something like the form outlined below.

The vocabulary makes use of the following classes, properties, and relations; we assume here the simple types of XML Schema, but the specifics of the simple type system are not relevant to our argument.[5] The purchase order has a date property, and relations with ship-to and bill-to addresses, comments, and items.

```
class(purchase_order).
property_of(purchase_order,date,'xsd:date').
relation(shipto,[purchase_order,address]).
relation(billto,[purchase_order,address]).
relation(has_comment,[purchase_order,comment]).
relation(has_item,[purchase_order,item]).
```

The actual purchase order in hand allows us to infer the existence of an instance of class *purchase_order*; we can assert its existence by giving it an arbitrary identifier and stating the values of its properties, and asserting its occurrence in members of various relations:

```
object(p123).
obj_class(p123,purchase_order).
models(p123,[node1]).
opv(p123,date,'1999-10-20').
relation_applies(shipto,[p123,a45]).
relation_applies(billto,[p123,a46]).
relation_applies(has_comment,[p123,c926]).
relation_applies(has_item,[p123,p123_i01]).
relation_applies(has_item,[p123,p123_i02]).
```

We call out persons as a special class, because (let us say) we know that they are important for the application area; a more purely mechanical translation from the schema might not define a separate class for persons.

```
class(person).
property_of(person,name,'xsd:string').
```

The two addresses each allow us to infer (for application purposes, at least) the existence of a person at that address:

```
object(s45).
object(s46).
obj_class(s45,person).
obj_class(s46,person).
opv(s45,name,"Alice Smith").
opv(s46,name,"Robert Smith").
```

Addresses have a number of simple properties, mostly string-valued. To illustrate the subclass predicates in our system, we have followed the XML Schema primer's international purchase order example in defining separate types for US and UK addresses. Because we have defined *person* as a separate class of objects, we need to use *relation*, not *property_of*, to describe the link between address elements and their first child (the *name* element).

```
class(address).
class(us_address).
class(uk_address).
subclass(us_address,address).
subclass(uk_address,address).
relation(is_at_address,[address,person]).
property_of(address,street,'xsd:string').
property_of(address,city,'xsd:string').
property_of(us_address,state,'xsd:string').
property_of(us_address,zip,'xsd:string').
property_of(uk_address,postcode,'xsd:string').
property_of(uk_address,exportcode,'xsd:positiveInteger').
```

The two addresses in our sample offer no particular difficulties:

```
object(a45).
obj_class(a45,us_address).
opv(a45,street,"123 Maple Street").
opv(a45,city,"Mill Valley").
opv(a45,state,"CA").
opv(a45,zip,90952).
...
relation_applies(is_at_address,s45,a45).
relation_applies(is_at_address,s46,a46).
```

Items on the purchase order have product name and part number, price, and ship date.

```
class(item).
property_of(item,productname,'xsd:string').
property_of(item,quantity,'xsd:positiveInteger').
property_of(item,us_price,'xsd:decimal').
relation(has_comment,[item,comment]).
property_of(item,shipdate,'xsd:date').
property_of(item,partnum,'xsd:string').
```

The simple structure of the example makes it easier to fill in the required information:

```
object(p123_i01).
obj_class(p123_i01,item).
opv(p123_i01,productname,"Lawnmower").
opv(p123_i01,quantity,1).
opv(p123_i01,us_price,148.95).
relation_applies(has_comment,[p123_i01,c926]).
opv(p123_i01,partnum,"872-AA").
...
```

Both purchase orders and items can have arbitrary numbers of comments; to model this one-to-many relation, we make comments a class of objects by themselves.[6]

```
class(comment).
property_of(comment,contents,'xsd:string').
```

The two comment elements in the document are represented straightforwardly:

```
object(c926).
obj_class(c926,comment).
models(c926,[node39]).
opv(c926,contents,"Hurry, my lawn is going wild.").

object(c927).
obj_class(c927,comment).
models(c927,[node55]).
opv(c926,contents,"Confirm this is electric").
```

Several observations may be worth making. First, some of these inferences are redundant and provide no new information. No e-commerce site will rely on incoming purchase orders for the knowledge that part 872-AA is a lawn mower or that one of its prices is $148.95. But the redundancy is intentional and may be useful: if the internal system catalog shows 872-AA as a power drill, there is a contradiction which should be resolved before the purchase order is fulfilled. Similarly, most production systems should know from the data provided by the U.S. Postal Service that the zip code given in the ship-to address is not valid for Pennsylvania. One step in resolving the problem is to notice the contradiction; the redundant information can help make that happen.

Some other inferences are also possible, if we apply some knowledge of the real world; in our system as currently structured, these are not application sentences but appear in the data flow diagrams as "Further sentences".

- There is a street in Mill Valley called Maple Street.

  The Maple Street in Mill Valley has a house 123 on it.

  And so on. Lots of inferences might be drawn from the addresses alone, but we will ignore most of them here.
- (Depending on how the purchase order was generated, we may be more or less likely to infer:) The item denoted by part number 872-AA was quoted to customer S-46 as having a price of $148.95.
- Because purchase orders are supposed to be placed only in good faith, we might infer that customer S-46 is willing to pay for the items on purchase order P-123 at the prices indicated.
- Because purchase orders are sometimes filed by the person who will receive the goods, it is possible that the act of placing this order was performed by Alice Smith.

  Because purchase orders are sometimes filed by the person who will pay for the goods, e.g., because they are giving someone a gift, it is possible that it is Robert Smith who placed the order.
- From the comment "Hurry, my lawn is going wild!" a human or a computer system with a good grasp of English syntax and pragmatics, may well infer that it was Alice Smith, not Robert Smith, who placed the order. This may be an important inference, but since it relies on our understanding of English, rather than on our understanding of the purchase-order markup, it is beyond the scope of systems like those we describe in this paper.
- Because there is no point in shipping goods to people who cannot use them, and since people can use goods only if the people are alive, we may infer that Alice Smith was alive when this order was placed.[7]
- A similar argument may lead us to infer that Robert Smith was probably alive when the order was placed.

There are some inferences we may be tempted to draw, and which may in fact be true in the common case, which are probably not, strictly speaking, licensed by the purchase order or its markup. We mark these tempting but not necessarily valid inferences with a star.

> * Person S-45 is desirous of receiving items 872-AA and 926-AA.
>
> * Person S-46 has expressed a willingness to pay for the items on purchase order P-123.

The first inference, though not strictly justified by the markup, may be true often enough that we may use S-45's address for mailings about lawn-care products.

The second inference, although it goes beyond the meaning of the markup as documented by its creator, may nonetheless be licensed by the knowledge at our disposal. If our business rules require some indication of willingness to pay the bill whenever the ship-to and bill-to addresses are not the same, then we may indeed be able to draw this inference — if customer S-46 had not expressed a willingness to pay, then the purchase order would not exist in this form at this point in the processing path. That is, the inferences we can draw from the markup itself may interact with general rules already present in our logical system (the *world knowledge* mentioned above) and generate further inferences. Such inferences are beyond the scope of the system we are describing, though clearly for some purposes the prospect of drawing such further inferences will be the main reason for wishing to generate application sentences in a logical notation.

§ 6 Skeleton sentences and mapping rules

One immediate problem we face is the development of a notation (specifically, an SGML/XML DTD) for expressing what [Sperberg-McQueen et al. 2001a] call “sentence skeletons”, or “skeleton sentences”. These are sentences, or rather fragments of sentences, either in English or some other natural language or in some formal notation, for expressing the meaning of constructs in a markup language. They are called skeleton sentences, rather than full sentences, because they have blanks at various key locations; a system for automatic interpretation of marked up documents will generate actual sentences by filling in the blanks in the skeleton sentences with appropriate values from the documents themselves.

Some of the skeleton sentences for our sample inferences in English might look like this, if we used parenthetical expressions to show how the blanks are to be filled in:

- Purchase order _____ (p.o. ID) was placed on _____ (date).
- There is someone named _____ (name) now able to receive mail at _____ (address).
- There is a street in _____ (city) called _____ (street).
- The items on _____ (p.o. ID) should be shipped to person _____ (ship-to-name) at address A-45.
- Customer _____ (bill-to-name) is willing to pay for the items on purchase order _____ (p.o. ID) at the prices indicated.
- Purchase order _____ (p.o. ID) includes _____ (quantity) item of part number _____ (part number).
- The item denoted by part number _____ (part number) is a _____ (productName).
- Customer _____ (bill-to-name) is willing to pay _____ (price) for the item denoted by part number _____ (part number).
- The item denoted by part number _____ (part number) was quoted to customer _____ (bill-to-ID) as having a price of _____ (price).

The skeleton sentences for our Prolog facts would look like this, if we use comments and XPath expressions to say how to fill in the blanks. First, the purchase order; the XPath expressions are to be interpreted as if the *purchaseOrder* element were the current node:

```
object(X),                   /* X is an arbitrary ID */
obj_class(X,purchase_order),
models(X,Y),                 /* Y is generate-id(.) */
opv(X,date,Z).               /* Z is string(./@date) */
```

Next, the person and address items; here, the *shipTo* or *billTo* element provides the current node for the XPath expressions:

```
object(P),
object(A),
obj_class(P,person),
obj_class(A,address),
opv(P,name,N),               /* N is string(./name) */
relation_applies(is_at_address,P,A),
opv(A,street,S),              /* S is string(./street) */
opv(A,city,C),                /* C is string(./city) */
opv(A,state,T),               /* T is string(./state) */
opv(A,zip,Z).                 /* Z is number(./zip) */
```

Note that if the current node has `country="US"`, then we should not assert **obj_class(A,address)**, but instead **obj_class(A,usaddress)**, and similarly for `country="UK"` and **obj_class(A,ukaddress)**. Similar skeleton sentences can be constructed along similar lines.

In existing markup systems, as in the imaginary vocabulary being used in the purchase order example, the appropriate values for variables are often to be taken from whatever occurs in the document at a specific location. In the TEI DTD, for example, the current page number for the default pagination is given by the value of the '*n*' attribute on the most recent '*pb*' element, while the identifier for the language of any natural-language material is given by the value of the '*lang*' attribute on the smallest enclosing element which actually has a value for the '*lang*' attribute; the language itself is described by whatever '*language*' element in the TEI header has that identifier as the value of its '*id*' attribute.

In the skeleton sentence, we need to label each blank with some expression which describes unambiguously how to derive the appropriate value to be used in the sentences constructed from this skeleton. The parenthetical notes and comments used in the examples above illustrate the point; in real systems we need a more formal way to provide the information. Since these expressions typically "point" to other nearby markup structures, we refer to them as "deictic expressions"; they express notions like "the contents of this element" or "the value of the 'lang' attribute on the nearest ancestor which has such a value" or "the value of the '*type*' attribute on the nearest ancestor of type '*div*'".

Several theoretical and practical problems arise in using skeleton sentences to say what the markup in some commonly used DTDs actually means, in a way that allows software to generate the correct inferences from the markup and to exploit the information.

First, what formalism should be used to write the skeleton sentences? We can easily adopt some existing formalism, e.g., that of Prolog or whatever inference engine we choose to use; can we devise a formalism that will not commit us to a particular inference engine?

There appears to be a significant difference among (a) skeleton sentences which serve to formulate in some formal notation the specific facts expressed by the markup in the document (the *mapping rules* described above), (b) sentences or skeleton sentences which express invariant rules about specific properties captured by the markup, e.g., "The value of the '*lang*' attribute is inherited; the value of the '*n*' attribute is not inherited" (the *property rules*) and (c) sentences or skeleton sentences which express invariant rules about textual and other constructs, e.g., "The author of a letter is physically located at the place given in the place-date line, on the date given in the place-date line, unless the letter is falsified or forged in some way" (which were described above as *world knowledge*). Sentences in group (b) serve to capture useful generalizations about the way markup constructs in a given DTD behave; sentences in group (c) are important for certain kinds of inferences, but appear to have relatively little to do with the markup itself. What is the best way to reflect these differences in function among the sentences and skeleton sentences of a markup system?

One experimental syntax for skeleton sentences uses Prolog notation for the structure of the sentences, a small XML vocabulary for the framework and for filling the blanks, and XPath as the language for the deictic expressions. Some of the skeleton sentences above would look like this in this syntax:

```
<var name="A" val="concat('o-',generate-id(.))"/>
<var name="P" val="concat('o-',generate-id(./name))"/>
<rule>object(<val var="P"/>).</rule>
<rule>object(<val var="A"/>).</rule>
<rule>obj_class(<val var="A"/>,address).</rule>
<rule>obj_class(<val var="P"/>,person).</rule>
<rule>relation_applies(is_at_address,
                    [<val var="A"/>,<val var="P"/>]).</rule>
<rule>opv(<val var="P"/>, name,  <de xv="string(./name)"/>).</rule>
<rule>opv(<val var="A"/>, street,<de xv="string(./street)"/>).</rule>
<rule>opv(<val var="A"/>, city,  <de xv="string(./city)"/>).</rule>
```

Each skeleton sentence is tagged as a *rule* element, the contents of which follow Prolog notation. Deictic expressions within the rule are marked by *de* elements, which carry an attribute whose value is an XPath expression. Frequently used expressions can be assigned to variables, for compactness and clarity. When the skeleton sentences are applied to the document instance in order to produce sentences in the notation of the logical system, a current element is selected and the XPath expressions are evaluated in that context. Their values are written out as part of a concrete sentence which has the same overall form as the skeleton sentence, but has no blanks.

The experimental syntax allows the creator of the skeleton sentences to specify the node from whose context the XPath expressions should be evaluated, by embedding the skeleton sentences within larger structures, which themselves identify element types by means of XSLT match patterns. A larger fragment of the documentation for the fictional purchase order language looks like the following example; the rules defined do not generate the Prolog shown above but a different set of Prolog application sentences. Each set of rules is embedded in an *elemtype* element which describes the element type to which they apply.

```
<elemtype match="purchaseOrder">
 <doc>
  <para>This XML element represents a purchase order.</para>
 </doc>
 <rule distributed="false" lang="Prolog">
  purchase-order(<de xv="generate-id(.)"/>).
 </rule>
 <rule distributed="false" lang="Prolog">
  dated(<de xv="generate-id(.)"/>, <de xv="@orderDate"/>).
 </rule>
</elemtype>

<elemtype match="shipTo">
 <doc>
  <para>The person and address to whom to ship.</para>
 </doc>
 <rule distributed="false" lang="Prolog">
  person(<de xv="generate-id(.)"/>, <de xv="name"/>).
 </rule>
 <rule distributed="false" lang="Prolog">
  address(a-<de xv="generate-id(.)"/>,
  <de xv="street"/>,
  <de xv="city"/>,
  <de xv="state"/>,
  <de xv="zip"/>).
 </rule>
 <rule distributed="false" lang="Prolog">
  person-address(<de xv="generate-id(.)"/>,
    a-<de xv="generate-id(.)"/>).
 </rule>
 <rule distributed="false" lang="Prolog">
  ship-to(<de xv="generate-id(..)"/>,
  <de xv="generate-id(.)"/>,
  a-<de xv="generate-id(.)"/>).
 </rule>
</elemtype>
```

§ 7 Some challenges

In our attempts to formulate skeleton sentences for existing real-world vocabularies like HTML, TEI, and Docbook, some questions have arisen which have proven difficult to answer.

In skeleton sentences like “[this element] is in English”, what do the deictic expressions like “this element” actually refer to? In some cases the inferences appear to relate to the linguistic components of the text itself (“This document is written in English”), and in some cases to the text's formal properties (“Augustine's *Confessions* is divided into 13 books”). In some cases, the markup appears to license inferences about some object or entity in the real world (“Henry Laurens was in Charleston on 18 August 1775”), but sometimes the entities referred to are not at all in the real world (“Harry Potter missed the Hogwarts Express on 1 September”). In still other cases, the inferences appear to apply to the electronic encoding of the text itself (“This metadata was last revised on 20 July 1998”) or to some other witness to the same text (“The recipient's copy of this letter is preserved in [some particular archival collection, with some particular call number]”). Attempting to disentangle these lands the would-be formulator of skeleton sentences promptly in a thicket of ontological questions which have not yet received adequate attention.

The ontological questions become even more thorny in connection with markup systems like that of the Text Encoding Initiative, which are intended for use by a wide variety of projects which are expected to have widely different views about the ontological commitments to be associated with the TEI markup. Do statements about Augustine's *Confessions*, for example, relate to some abstract text distinct from each physical copy of the text, or is the phrase “Augustine's *Confessions*” merely a convenient shorthand for “all the physical documents which witness Augustine's *Confessions*”? It would appear essential to decide this question in order to formulate skeleton sentences for markup languages like the TEI, but the TEI itself is intentionally coy about the issue, in order to ensure that textual Platonists and textual constructivists can both use TEI markup. It is a challenge to build a similar ambiguity or vagueness into the set of skeleton sentences which document the prescribed interpretation of TEI markup.

Notes

1. In this we follow a proposal made by Turski and Maibaum in their discussion of programming-language semantics [Turski/Maibaum 1987]; they put the proposal thus (p. 4):

   “Two points deserve special attention: we expect programs to be capable of expressing a meaning and we want to be able to compare meanings. Unless we are very careful, we may very soon be forced to consider an endless chain of questions: what is the meaning of ...? what is the meaning of the meaning of ...? etc. Without going into a philosophical discussion of issues certainly transgressing any reasonable interpretation of the title of this book, we shall accept that *the meaning of A is the set of sentences S true because of A*. The set *S* may also be called the set of consequences of *A*. Calling sentences of *S* consequences of *A* underscores the fact that there is an underlying logic which allows one to deduce that a sentence is a consequence of *A*.

   “Usually the *A* itself is a set of sentences, thus we are saying in fact that the meaning of a set of sentences is the set of consequences that are deducible from it.”

2. The term *deictic expression* comes from traditional grammar, where it is used to denote pointing expressions like “this one over here” or “that one over there” — from the Greek word *deixis*, for pointing.

3. The history of work in artificial intelligence includes many examples of knowledge bases which capture the kind of information we believe will go here, and attempt to show how to build useful applications using it. Recent relevant work includes [Fikes/McGuinness 2001].

4. Obviously, XQuery functions can also be used.

5. The simple types assigned are more or less those assigned in the XML Schema primer; we have not attempted to model the USState type here, which is a restriction of *xsd:string* with an enumerated set of values. We have retained the type *xsd:positiveInteger* for zipcode; we are strongly tempted to change it to *xsd:string* for the sake of verisimilitude, because

leading zeroes are not omissible in zip codes, but we have decided to follow the schema primer here.

6. We could allow many-valued properties, but prefer a more strictly relational approach.

7. Actually, if the order was placed by Robert Smith, it is not entirely safe to infer that Alice Smith was alive; it is plausible, however, that Robert Smith thought she was alive. Unless, of course, we are reasoning about events in a detective story, in which case it may only be the case that Robert Smith wished to give the *impression* that he thought Alice Smith was alive.

Bibliography

[ACH/ACL/ALLC 1994] Association for Computers and the Humanities, Association for Computational Linguistics, and Association for Literary and Linguistic Computing. 1994. *Guidelines for Electronic Text Encoding and Interchange (TEI P3)*. Ed. C. M. Sperberg-McQueen and Lou Burnard. Chicago, Oxford: Text Encoding Initiative, 1994.

[Coombs et al. 1987] Coombs, J. H., Renear, A. H., and DeRose, S. J. 1987. Markup systems and the future of scholarly text processing. *Communications of the Association for Computing Machinery 30*, 11 (1987), 933–947.

[Cowan/Tobin 2001] Cowan, John, and Richard Tobin, ed. 2001. “XML Information Set.” W3C Recommendation 24 October 2001. [Cambridge, Sophia-Antipolis, Tokyo]: World Wide Web Consortium. http://www.w3.org/TR/xml-infoset/

[Fallside 2001] Fallside, David C. 2001. XML Schema Part 0: Primer. W3C Recommendation, 2 May 2001. [Cambridge, Sophia-Antipolis, Tokyo]: World Wide Web Consortium. http://www.w3.org/TR/xmlschema-0/

[Fikes/McGuinness 2001] Fikes, Richard, and Deborah L. McGuinness. 2001. “An Axiomatic Semantics for RDF, RDF-S, and DAML+OIL (March 2001).” W3C Note 18 December 2001. [Cambridge, Sophia-Antipolis, Tokyo]: World Wide Web Consortium. http://www.w3.org/TR/daml+oil-axioms

[Guttag/Horning 1993] Guttag, John V., and James J. Horning, with S. J. Garland et al. 1993. *Larch: languages and tools for formal specification*. New York: Springer-Verlag. Texts and monographs in computer science 981.

[Hughes/Cresswell 1968] Hughes, G. E., and M. J. Cresswell. 1968. *An introduction to modal logic*. London: Methuen.

[ISO 1986] International Organization for Standardization (ISO). 1986. *ISO 8879-1986 (E). Information processing — Text and Office Systems — Standard Generalized Markup Language (SGML)*. International Organization for Standardization, Geneva, 1986.

[Ramalho et al. 1999] Ramalho, José Carlos, Jorge Gustavo Rocha, José João Almeida, and Pedro Henriques. 1999. “SGML documents: Where does quality go?” *Markup Languages: Theory & Practice* 1.1 (1999): 75-90.

[Renear 2001] Renear, A. 2001. Raising the bar: Text encoding from a logical point of view. CLIP 2001: Computers, Literature, Philology, Gerhard-Mercator University, Duisburg, Germany, December 2001.

[Rowe 1988] Rowe, N. C. 1988. *Artificial Intelligence through Prolog*. Prentice Hall, Englewood Cliffs, NJ.

[Schatz et al. 1996] Schatz, B., Mischo, W. H., Cole, T. W., Hardin, J. B., Bishop, A. P., and Chen, H. 1996. Federating diverse collections of scientific literature. *Computer 29* (May 1996), 28–36.

[Simons 1997] Simons, Gary F. 1997. “Conceptual Modeling versus Visual Modeling: A Technological Key to Building Consensus.” *CHum* 30.4: 303-319.

[Simons 1999] Simons, Gary F. 1999. “Using Architectural Forms to Map TEI Data into an Object-Oriented Database.” *CHum* 33.1-2: 85-101. Originally delivered in 1997 at the TEI 10 conference in Providence, R.I.

[Sperberg-McQueen 1991] Sperberg-McQueen, C. M. 1991. “Text in the Electronic Age: Textual Study and Text Encoding, with Examples from Medieval Texts.” Literary and Linguistic Computing, 6:1, 34-46.

[Sperberg-McQueen et al. 2001a] Sperberg-McQueen, C. M., Claus Huitfeldt, and Allen Renear. 2001. “Meaning and interpretation of markup.” *Markup Languages: Theory & Practice* 2.3 (2001): 215-234. http://www.w3.org/People/cmsmcq/2000/mim.html

[Sperberg-McQueen et al. 2001b] Sperberg-McQueen, C. M., Claus Huitfeldt, and Allen Renear. 2001. “Practical extraction of meaning from markup.” Paper given at ACH/ALLC 2001, New York, June 2001. (Slides at http://www.w3.org/People/cmsmcq/2001/achallc2001/achallc2001.slides.html)

[Swick/Thompson 1999] Swick, Ralph R., and Henry S. Thompson, ed. 1999. *The Cambridge Communiqué*. W3C NOTE 7 October 1999. http://www.w3.org/TR/schema-arch

[Thompson 2001] Thompson, Henry S. 2001. “Normal Form Conventions for XML Representations of Structured Data”. Talk at XML 2001, Orlando, December 2001. http://www.ltg.ed.ac.uk/~ht/normalForms.html

[Turski/Maibaum 1987] Turski, Wladyslaw M., and Thomas S. E. Maibaum. 1987. *The specification of computer programs*. Wokingham: Addison-Wesley.

[Vorthmann/Robie 2001] Vorthmann, Scott, and Jonathan Robie. 2001. “Beyond schemas: Schema adjuncts and the outside world”. *Markup Languages: Theory & Practice* 2.3: 281-294.

[W3C 2000] W3C (World Wide Web Consortium). 2000. “XHTML 1.0: The Extensible HyperText Markup Language. A Reformulation of HTML 4 in XML 1.0.” W3C Recommendation 26 January 2000 [Cambridge, Sophia-Antipolis, Tokyo]: W3C. http://www.w3.org/TR/xhtml1/

[Walsh/Muellner 1999] Walsh, N., and Muellner, L. 1999. *DocBook: The Definitive Guide*. O'Reilly and Associates, Inc., Sebastopol, CA.

[Welty/Ide 1997] Welty, Christopher, and Nancy Ide. 1997. “Using the Right Tools: Enhancing Retrieval from Marked-up Documents.” *CHum*y 33 (1999): 59-84. Originally delivered in 1997 at the TEI 10 conference in Providence, R.I.

The Authors

C. M. Sperberg-McQueen
World Wide Web Consortium, MIT Laboratory for Computer Science

C. M. Sperberg-McQueen is the Architecture Domain Lead for the World Wide Web Consortium and a visiting researcher at the University of Bergen.

David Dubin
University of Illinois at Urbana/Champaign, Graduate School of Library and Information Science

David Dubin is a research scientist at UIUC's Information Systems Research Laboratory, working mainly with the Electronic Publishing Research Group. His research interests are in the areas of information retrieval, text and document processing, and classification. He is a member of the Association for Computing Machinery, the American Society for Information Science, and the Classification Society of North America.

Claus Huitfeldt
Avdeling for kultur, språk og informasjonsteknologi, Bergen University Research Foundation

Claus Huitfeldt is an associate professor of philosophy at the University of Bergen and the director of the the University of Bergen Research Foundation's department for research in culture, society, and technology (AKSIS) and of the Humanities Information Technology Research Programme. From 1990 to 1999 he was the director of the Wittgenstein Archives at the University of Bergen, which created an electronic edition of Wittgenstein's posthumous papers.

Allen Renear
University of Illinois at Urbana/Champaign, Graduate School of Library and Information Science

Allen Renear is an Associate Professor in the Graduate School of Library and Information Science. His principal research focus is on how digital documents function as knowledge representation systems.

As a student at Bowdoin College and Brown University, he specialized in epistemic logic and the philosophy of science; after several years teaching philosophy, he joined Brown's Computing and Information Services in 1984, working first as a systems analyst and project leader, and then as a strategic planner. During this time he consulted on or managed many humanities computing projects and became involved in a variety of text encoding and computing activities — including X3V1.TG8, the Text Encoding Initiative (TEI), and the Association for Computers and the Humanities (ACH). In 1988 Renear helped design the Brown Women Writers Project (WWP), serving at various points as WWP Co-Director, Acting Director, and Director, and in 1993 he became founding Director of the Scholarly Technology Group. He is currently the chair of the Open eBook Publication Structure Working Group.

Extreme Markup Languages 2002
Montréal, Québec, August 6-9, 2002
This paper was formatted from XML source via XSL
Mulberry Technologies, Inc., August 2002