INSTITUT FÜR INFORMATIK
Lehr- und Forschungseinheit für
Programmier- und Modellierungssprachen
Oettingenstraße 67, D–80538 München

Ludwig
Maximilians-
Universität
München

Data Modeling with Markup Languages: A Logic Programming Perspective

Franois Bry, Norbert Eisinger

appeared in *Proc. 15th Workshop on Logic Programming and Constraint Systems*,
WLP 2000, Berlin
http://www.pms.informatik.uni-muenchen.de/publikationen
Forschungsbericht/Research Report PMS-FB-2000-9, August 2000

Data Modeling with Markup Languages: A Logic Programming Perspective

François Bry and Norbert Eisinger
Institute for Computer Science
University of Munich, Oettingenstr. 67, 80538 Munich, Germany
http://www.informatik.uni-muenchen.de/

Abstract

This paper consists of three parts. In a first part, it is argued that generic markup languages such as SGML and XML, that have been developed primarily for modeling *documents* are also convenient for modeling *general data*. In a second part, XML and concepts related to XML are introduced. A third part presents a logic programming perspective to data modeling with generic markup languages.

Contents

1 From Document Modeling to Data Modeling

Modern markup languages, such as SGML (Standard Generalized Markup Language) [40] and XML (eXtensible Markup Language[1]) [50] which were initially conceived for modeling texts, are now receiving an increasing attention as formalisms for data and knowledge modeling. XML is currently establishing itself as a successor of HTML (HyperText Markup Language) [24] for a better modeling of texts as well as of other kinds of data. There are several reasons for this evolution:

- modern markup languages are "generic",
- they offer a rich spectrum of complementary data modeling facilities, and
- they give rise to specify the semantics of the terms used in modeling data.

[1]XML is developed by the *World Wide Web Consortium (W3C)* [47].

In the following, these reasons are discussed in more detail.

1.1 Modern Markup Languages are Generic

Modern markup languages such as SGML and XML are generic, i.e.,

- they serve to specify the semantic structure, not the layout of documents or data items, and
- they make it possible to freely specify application dependent document or data structures.

In the following, the term data refers also, but not exclusively, to text data. Thus, a data item may consist: of text only (such data items are also known as human-readable documents), of non-text only (such data items are also known as data-oriented documents), or of both (such data items are also known as mixed-model documents).

In the terminology of generic markup languages, data items are called documents. In the following, the term data item is used in lieu of document for stressing that not only (structured) texts are meant, but more generally (structured) data of any kind.

1.1.1 Markup

"Mark up" (in two words) usually refers to the annotations one adds to printer's proofs in order to improve or correct the text or the layout. "Markup" (in one word) was initially used to refer to formatting languages with which the layout, that is the rendering of a text on paper or on a screen, can be specified. Nowadays, "markup" is also used for languages such as SGML [40] and XML [50] with which the semantical structure of documents or data items can be specified, as well as for emerging forms of rendering of electronic documents such as in Braille (for visually disabled readers) or in synthesized speech.

1.1.2 Specific and Generalized Markup Languages

Markup languages are often distinguished into "specific markup languages", "generalized markup languages", and "generic markup languages". This classification stresses an historical evolution.

Specific markup languages are low level languages such as PostScript of Adobe and RTF (Rich Text Format) of the Microsoft Corp. that serve to specify the rendering of texts. Specific markup languages offer commands for modifying the appearance of characters or of portions of texts (for example, for selecting fonts, or a style such as bold or italics).

The name of "generalized markup language" was coined for the language DCF GML (Document Composition Facility Generalized Markup Language) [19]. This language, developed at IBM at the end of the sixties, was the first markup language designed for specifying the logical structure of documents referring to structuring elements such as "chapters", "sections", "title" and the like. The possible structure was predefined and the assignment of layout characteristics to structuring elements was predefined. Thus, the generalized markup language DCF ML provides with a higher level of abstraction than specific markup languages. Note that with DCF ML both the denomination "markup language" and the acronym "ML" were used for the first time.

1.1.3 Layout vs. Structure

Widespread specific markup languages such as PostScript of Adobe or RTF (Rich Text Format) of the Microsoft Corp. whose conceptual roots go back to the seventies, serve to specify the layout of data items. Here layout is not exclusively meant as the appearance of a data item when printed on paper, but more generally as any kind of presentation of a data item to human perception. Examples of such an extended notion of layout include the formats of data items as they are displayed on a terminal screen, vocalized by a speech synthesizer, rendered in the Braille script on a tactile output device, or presented by any other means on any device.

The family of generic markup languages started in the late eighties with the conception of its first specimen, SGML [40]. The purpose of a generic markup language is to specify the semantic – or logical – structure of data items, not their layout. Consider, for example, a data item representing a letter. A generic markup language can be used to mark some part of this data item as the salutation, another part as the complimentary close, and so on. In contrast to that, a specific markup language would be used to assign certain indentations, spacings, font weights, font sizes etc. to these parts of the data item, without making explicit which relationships the parts have to the whole.

In the following, the term presentation is reserved to refer to the layout of a data item in the extended sense above, while the term representation refers to how semantics is conveyed through structural elements of the underlying data modeling formalism.

The distinction between layout and structure is important, for a layout format is device or system dependent, whereas a semantic structure should not be. It is desirable that the semantic structure of data items be specified independently of any layout. This ensures both:

- independence of data modeling from data usage,
- independence of data modeling from presentation devices.

The first property, data independence from usage, is important because data is rarely used in a single manner only. For example, a list intended to provide tourist information about cities might be "misused" by someone looking up postal codes and telephone area codes in order to fill gaps in an address book, and by someone else extracting the numbers of inhabitants as raw data for some survey on population distribution. Neither of these "misusers" would be interested in what the author considered the primary information: sightseeing, transportation, night life, and so on. In cases where such "misuses" of an information server or database can be foreseen by the designers, they are often modeled relying on so-called database views, short views.

The second property, data independence from presentation devices, is important for several reasons. To begin with, different kinds of presentation devices require different layouts. For example, a structurally complex data item is likely not to be displayed using identical layouts on standard size screens and on small screens like those of cellular phones. Also, such devices are likely to become technically obsolete sooner than data. Moreover, a presentation format does not necessarily fully convey data semantics. For instance, it is common practice to rely on printed text layout for conveying semantic structure when using text processing systems or the markup language HTML. This practice often leads to semantic losses, especially when files are transferred from one text processing system to another, because the layout of the one system cannot always be faithfully mapped into that of the other system.

In order to specify layouts for classes of documents specified in a generic markup language, so-called style-sheet languages are used in addition. These languages basically allow the definition of layouts for those structural elements specified with the markup language. Such definitions do not have to be unique, thus ensuring the desired independence of the data from their presentations in various contexts.

1.1.4 Free Structures with Free Naming – Structure-Conveying Data

Generic markup languages neither impose any predefined structure, nor predefined names for the structural elements occurring in data items. Structure and names can be freely chosen, hence the denomination of generic markup language.

Thus, using generic markup languages it is possible to faithfully model the structure of data items needed in applications and to name the structural elements of a chosen structure in a way that is natural in the application context. The "naming freedom" provided by generic markup languages reminds of modern high-level programming languages that allow the programmer to rely on intuitive names for variables, routines, and other constructs of the programming language.

Using a generic markup language, an address book might thus consist of person elements, which in turn consist of first-name, last-name, geographical-address, email-address, and phone-number elements. A geographical-address element might in turn consist of subelements street, street-number, city, and state. An example of an entry in such an address book is as follows:

```
<person>
   <first-name>Francois</first-name>
   <last-name>Bry</last-name>
   <geographical-address>
      <street>rue de Vincennes</street>
      <street-number>70</street-number>
      <city>Bordeaux</city>
      <state>France</state>
   </geographical-address>
   <email-address>francois.bry@hotmail.com</email-address>
   <phone-number>0033556988223</phone-number>
</person>
```

The use of application relevant names for structural elements is at the origin of the expression of structure-conveying data. Note that data items structured in such a manner are not necessarily accompanied by a schema (here and in the following, the term schema refers to a DTD (Document Type Definition), an XML schema, or some other specification of the structure underlying the data items considered).

It is worth recalling that the widespread markup language HTML is not a generic markup language in spite of its syntax similarities with SGML and XML. Indeed, although HTML provides for structuring constructs, it is not free from layout constructs. Furthermore, names for structuring constructs cannot be chosen freely in HTML. The fact that interfaces to a number of style-sheet languages have been defined for HTML has no influence on the non-generic character of that markup language.

1.2 Data Modeling Facilities of Generic Markup Languages

The formalisms used by generic markup languages for specifying the structure of classes of documents or more generally of data items allow for

- complex structures as they can be specified with object oriented knowledge representation formalisms,
- data sharing like in object database models,
- optional elements, and
- different kinds of pointers.

Technically, the formalisms used by generic markup languages for specifying the structure of classes of data items basically – i.e., with minor restrictions – amount to context free grammars. Context free grammars are a well-established formalism used especially for specifying and compiling programming languages.

Complex structures are essential for they are ubiquitous in data modeling – from records in programming languages to objects in programming languages, artificial intelligence, software engineering, databases, and logics for knowledge representation.

Data sharing is a key feature in data modeling, for without it error prone redundancies would be unavoidable.

1.2.1 Modeling Exceptions – Semistructured Data

The formalisms provided by generic markup languages make it possible to specify complex structures. Therefore they are much richer than the data model of relational databases. They are also richer than the data models of current object database systems, because they allow optional elements.

Optional elements are very appealing in databases, for they make it possible to express exceptions, which often occur in practical database applications. Considering again the above-mentioned address book database, it might for example happen that not all entries have a first-name element, even though most do have one. The name of semistructured data has been coined for emphasizing the possibility of such exceptions in the framework of structure-conveying data. It is under the denomination of semistructured data that most database research on using markup languages for data modeling is currently pursued.

1.2.2 Pointers and Hypertext

The predominant reason for the success of modern markup languages – both, specific and generic ones – is the ability to define pointers between data items and to follow them at the click of a button. Thus, markup languages make it very easy to construct so-called hypertext systems, i.e., networks of information-containing nodes interconnected by hyperlinks (a more common term is hypertext links, which might wrongly suggest that nodes must consist of text rather than arbitrary data items; a more sloppy term is links, which could be confused with other kinds of pointers). There may be different types of hyperlinks. In the following, the available hyperlink types and their meanings are referred to as the hyperlink model. This term has been consciously chosen in allusion to the well-established and more general notion of hypertext model ([7, 26, 23]).

The generic markup language SGML is usually considered to have no hyperlink model, although its attribute types ID and IDREF enable a simple scheme of binary, directed hyperlinks whose source and target must be part of a single data item. The generic markup language XML inherits this mechanism from SGML, but is also accompanied by the XML Linking Language XLink [63]. XML+XLink provides constructs for a wide range of hyperlink types, which largely correspond (with other names) to the quite sophisticated HyTime model [26]. The constructs of XML+XLink cover, among others,

- binary, directed hyperlinks between a source data item and a target data item (if in addition source and target belong to a single data item, such hyperlinks can actually be represented in different ways: either using the ID/IDREF attribute types inherited from SGML, or using the more powerful constructs provided by XLink),
- undirected hyperlinks,
- hyperlinks that have more than two participating data items,
- out-of-line hyperlinks ("inbound" and "third-party" links in the more recent terminology of XLink), that is links that are not part of any of their participating data items and not even stored together with them,
- application-specific metadata that can be associated with a hyperlink.

This variety of hyperlinks is but one aspect of the richness of XML as a data modeling formalism. Some of the hyperlink types are very simple and convenient for expressing data sharing in complex structures, making possible structures corresponding to graphs that are no trees. More complex hyperlink types are primarily intended for browsing facilities, but seem to be usable also for expressing semantic relationships.

It is worth noting that, according to XLink [63], an XML link is not necessarily a hypertext link. An XML link is defined as an n-ary relationship between resources that *might* be a hypertext link. Thus, XLink can be seen as a kind of Entity-Relationship model [45].

1.3 Ontologies

Enriching standard data – like the numerical and string data of classical managerial databases – with more informative texts is an old issue in database research and in applications known as "data dictionaries". Data dictionaries are related to what is called "thesauri" in natural language and document processing. Data dictionaries and thesauri are basically agreed vocabularies for an application or a class of applications and often taxonomies, i.e., classifications of terms. Recently, the issue has gained new attention and has been enhanced with artificial intelligence knowledge modeling techniques leading to so-called ontologies [22, 39, 64].

An ontology basically provides a (often commented) vocabulary whose terms are precisely defined by texts such as dictionary entries or encyclopaedia entries. Further, an ontology also defines semantic relationships between terms using formal modeling techniques, in general taken from logic-based specification formalisms such as description logics. Thus, an ontology starts from precisely defined basic concepts, building up complex concepts by relying on relationships that are precisely defined as well. These relationships permit the construction of taxonomies, but also of richer structures.

With the advent of the Web and data interchange intensive applications such as electronic commerce, ontologies are becoming a key issue. They are used for ensuring software interoperability and data sharing [34, 35]. In spite of their practical relevance, data dictionaries and ontologies have up till recently not received as much attention from the database research community as database practitioners might have wished.

This discrepancy clearly reflects the fact that, up till now, textual data was not a central interest in database research and database system development, nor was the modeling of complex semantic relationships as in artificial intelligence knowledge modeling approaches such as description logics. This seems to be changing now. Arguably, textual data and the modeling of complex semantic relationships are gaining in importance within database research.

1.4 Data Modeling with Generic Markup Languages as an Interdisciplinary Area of Research

Data modeling with generic markup languages is an interdisciplinary area of research at the crossing of four traditionally distinct fields of research:

- databases
- artificial intelligence
- information retrieval
- document processing

This convergence is interesting, because each field brings its own focus, methods, and "philosophy".

From databases, the research area of data modeling with generic markup languages gains an interest for declarative query languages, of which SQL is the most known example. Declarativeness is a loosely defined notion to be understood here as meaning that the users of such query languages neither have to be aware of the computation strategy, nor of the internal organization of the data in memory, nor – or as little as possible – of termination issues and of efficiency. Indeed, queries expressed in query languages are automatically optimized. This makes query optimization possible. Query optimization guarantees a predictable “average efficiency”, which is one of the appreciated features of database systems. Also from databases, the area of data modeling with generic markup languages inherits its interest for data structures ensuring an efficient storage, retrieval, and updating of very large data sets. Conversely, database research itself is enriched by the attention to text data, to data accessible from the Web, and to richer data models allowing for exceptions.

From artificial intelligence, the research area gains data and knowledge modeling methods that go far beyond the relational or object database models. Artificial intelligence approaches to knowledge representation have always been driven by natural language applications, where extremely rich and complex semantics are encountered. Natural language applications have inspired most knowledge representation research and are at the origin of knowledge representation approaches like semantic networks and the nowadays ubiquitous object orientation. The need for software interoperability and data interchange in Web-based applications such as electronic commerce, health care management, and computational biology (also called bioinformatics) nowadays has led researchers as well as practitioners to express advanced artificial intelligence knowledge representation formalisms such as description logics by relying upon the generic markup language XML.

From information retrieval, the research area can learn how automatically to “grasp” knowledge from the content of – in general large – texts. The field of information retrieval itself gains from the consideration of structured texts, which up till recently have almost never been considered by the information retrieval community.

The contribution of the document processing field to the research area of data modeling with generic markup languages are indeed the generic markup languages and hyperlink models. Document processing itself might benefit from the interdisciplinary approach to data modeling with methods for declarative query answering and for an efficient storage.

2 An “XML Ontology”

Someone interested in XML [50] and related issues is faced with a large number of notions, names, acronyms, formalisms, and specifications, the importance and meaning of which are not always immediate: DTD, element, attribute, processing instruction, XML Schema, XPath, XPointer, XLink, style sheet, CSS, XSL, XSLT, XSL-FO, XQL, etc. This section aims at being a short “XML Ontology” introducing into these notions, names, acronyms, formalisms, and specifications.

2.1 From SGML to XML

Compared to specific and generalized markup languages, generic markup languages increase the abstraction level: They are “meta-level” languages making it possible to define “object-level” markup languages such as HTML [24] and DCF ML [19].

SGML [40] is the first generic markup languages which has been and still is widely used. XML [50], which has been developed officially since 1996, informally a little bit earlier, was defined as a simplification of SGML, but in a sense it is also an extension of SGML.

XML is a simplification of SGML because it precludes the so-called *markup minimization* of SGML, that is, leaving out some of the markup which is implicit from the context. An example of markup minimization (which is allowed in HTML 4.0, but forbidden in XHTML) would be to leave out the tags `</li>` indicating the ends of list items in the following HTML unordered list:

```
<ul>
   <li>From Document Modeling to Data Modeling</li>
   <li>An "XML Ontology"</li>
   <li>A Logic Programming Approach</li>
</ul>
```

Also, XML does not require that a document is necessarily accompanied by a "document type definition (DTD)" specifying the document's structure.

XML is an extension of SGML, because, in contrast to SGML, XML has a companion hypertext model. This hypertext model is called Xlink and is inspired by HyTime [26], although it uses different terminology. Technically its specification [63] is separate from the XML specification. Nevertheless XML and XLink really ought to be regarded together, because XML is designed to become the successor of HTML [24] as the markup language of the World Wide Web, making hyperlinks indispensable.

2.2 The acronym "XML"

The acronym XML is primarily used as the name of a generic markup language called "Extended Markup Language", specified in [50], and derived from SGML [40]. As explained above, the intentions of the XML designers suggest that the acronym also covers the hypertext model XLink specified in [63].

For the same kind of reasons the acronym XML is also used in a broader sense to refer to a whole collection of languages and formalisms including for example XPointer [66], XPath [65], and XSLT [71], which are all related to the generic markup language XML understood in the narrower sense.

2.3 Special Purpose Markup Languages defined in XML

Using a generic or meta-level markup language such as XML, special purpose (or generalized) markup languages can be defined. During the last years, several such languages have been defined using XML. Such markup languages are sometimes called "XML applications". Since "XML applications" is also used to refer to XML related software such as document or DTD[2] editors, parsers, and browsers, we prefer the denomination "special purpose markup languages".

2.3.1 Uses of XML from Biology to Client-Server Protocols

The following languages are a few arbitrarily chosen examples from a multitude of special purpose markup languages defined in XML:

Bioinformatic Sequence Markup Language, BSML [8]. BSML aims at a graphical presentation of genomic data.

Channel Definition Format, CDF [10]. CDF is an open specification developed by the Microsoft Corp. that permits a web publisher to offer frequently updated collections of information, called *channels*, from any web server for automatic delivery to compatible receiver programs on PCs or other information appliances.

Extensible HyperText Markup Language, XHTML 1.0 [49]. XHTML 1.0 is a redefinition of HTML 4.0 in XML 1.0.

[2]The meaning of "DTD" is explained below.

Mathematical Markup Language, MathML [29]. MathML is a special purpose markup language defined in XML for describing mathematical notation and capturing both its structure and content. The goal of MathML is to enable mathematics to be served, received, and processed on the Web, just as HTML has enabled this functionality for text.

Open Software Description, OSD [35]. OSD is a vocabulary jointly developed by Marimba, Inc. and the Microsoft Corp. aiming at describing software packages and their dependencies for heterogeneous clients. It is expected to be useful in automated software distribution environments.

Platform for Privacy References, P3P [36]. P3P is an extension of the protocol HTTP (HyperText Transfer Protocol) [25] with which web users can specify "privacy practices", that is whether and under what circumstances personal information should be disclosed. P3P offers a standard, machine-readable format that can be retrieved automatically and interpreted easily by browsers.

Precision Graphic Markup Language, PGML [37]. PGML is a language for two-dimension graphics meeting both the simple vector graphics needs of casual users and the precision needs of graphics artists.

Synchronized Multimedia Integration Language, SMIL [41]. SMIL is a language for integrating a set of independent multimedia objects into a synchronized multimedia presentation. Using SMIL, an author can (1) describe the temporal behavior of the presentation, (2) describe the layout of the presentation on a screen, and (3) associate hyperlinks with media objects.

Scalable Vector Graphics, SVG [44]. SVG is a language for describing two-dimension graphics in XML. SVG allows for three types of objects: (1) vector graphic shapes (e.g., paths consisting of straight lines and curves), (2) images and (3) text. With SVG, graphical objects can be grouped, styled, transformed and grouped into complex objects.

Vector Markup Language, VML [46]. VML is a language for scalable two-dimension structured graphics (based upon paths described using connected lines and curves) to be used with HTML and XML.

HTTP Extensions for Distributed Authoring, WEBDAV [48]. WEBDAV is an extension to the HTTP/1.1 protocol [25] using an XML syntax that allows clients to perform remote web content authoring operations.

2.3.2 Uses of XML within the XML Specification

Recently, two languages defined in XML have been developed that serve to meta-level specifications within the XML languages itself: XML Schema [60, 61, 62] and XLink [63]. Using XML within the specification of XML itself has the advantage that tools developed for XML applications, such as editors and parsers, can be applied to meta-level specifications.

XML Schema [60, 61, 62]. XML Schema is a markup language with which the structure of (classes of) XML documents can be defined. XML Schema offers much richer modeling facilities than the DTD formalism[3] such as simple and complex types, object classes and inheritance.

XML Linking Language Version 1.0, XLink 1.0 [63]. XLink is a formalism which allows elements to be inserted into XML documents in order to create and describe relationships, called "links", between "resources". XLink relationships can, but need not be hypertext links.

2.3.3 Uses of XML in XML Related Specifications

Finally, tools and formalisms related to XML, that are no components of the XML language itself, have been developed as markup languages defined in XML.

XSL Transformations Version 1.0, XSLT 1.0 [71]. XSLT is a language for transforming, for example for restructuring or for formatting, XML documents into other XML documents. A typical example of restructuring is the automatic generation of a sorted index or of a table of contents. A typical example of formatting is the generation of an XHTML [49] document from an XML document for rendering it with a standard browser. XSLT can also be used for querying XML documents.

Formatting Objects of XSL, XSL-FO [70]. XSL-FO is used to attach formatting, or rendering properties to an XML document.

Document Content Description for XML, DCD [14]. The DCD facility for XML is an RDF [39] (see below) vocabulary designed for describing constraints to be applied to the structure and content of XML documents.

Resource Description Framework (RDF) [39]. RDF is a formalism for modeling metadata based upon relationships reminding of the Entity-Relationship data model [45]. RDF gives rise to define sitemaps, content ratings, stream channel definitions, search engine data collections to be used for "web crawling", digital library collections, and provides a framework for distributed authoring.

2.4 XML Basics

In this section, the main notions of the XML markup language are explained.

2.4.1 Document Prolog and Document Instance

An XML document contains the following components in that order:

1. An optional *prolog* containing a number of declarations,
2. a *document instance* which is a textual, that is *linear*, representation of a tree structure called *document tree*.

In addition, comments (given like HTML comments as `<!-- ... -->`) and so-called *processing instructions* (given as `<? ... ?>`) can occur within the prolog as well as within the document instance. Processing instructions are used to provide application software, such as browsers or parsers, with additional information.

The following is an example of a simple XML document:

[3]This formalism is introduced below.

```
<?XML version="1.0" encoding="iso-8859-1"?>
<?XML:stylesheet href="http://www.XMLsoft.com/collection.xsl"
                 type="text/xsl"?>
<address-book>
   <card>
      <last-name>Eisinger</last-name>
      <first-name>Norbert</first-name>
      <email>eisinger@informatik.uni-muenchen.de</email>
      <phone>08921782218<phone>
   </card>
   <card>
      <last-name>Bry</last-name>
      <first-name>Francois</first-name>
      <fax>08921782211</fax>
   </card>
</address-book>
```

The three first lines of this XML document are its prolog. The first line is the so-called *XML declaration* which indicates which version of XML and which alphabet are used. The second and third lines are a processing instruction mentioning a *style sheet* (cf. below) to be used for example by a browser to display the document. The remaining lines define the document instance.

The tree structure specified by the document instance is made explicit in this example by the indentation. It is worth noting that XML requires no such indentation and does not distinguish between carriage return, line feed, and other spacing characters that are all treated as "spaces".

2.4.2 Opening Tags, Closing Tags, and Elements

A sequence of characters like `<card>` is called an *opening tag*. A sequence of characters like `</card>` is a *closing tag*. The portion of text beginning with an opening tag and ending with the corresponding closing tag with the same label is called an *element*. The text between the opening tag and the closing tag is called the *content* of the element. The content of an element may in turn contain elements. In the example above the content of the `address-book`-element consists of two `card`-elements.

A document instance must consist of exactly one element, that is, a document instance must start with an opening tag <*label*> and end with a closing tag </*label*>. In other words, a document instance must specify a tree, not a general forest.

2.4.3 Empty Elements

Elements may be *empty*, i.e., may have empty content as in:

```
<a></a>
```

Empty elements may be written in an alternative syntax combining the opening and closing tags into one tag:

```
<a/>
```

Empty elements are especially useful for conveying structure, or in combination with attributes (see below).

2.4.4 Document Tree Nodes vs. Elements

XML does not preclude that "element free text" occurs in the document tree structure at the same level as elements, as for example the string "`This is element free text.`" in the following document instance:

```
<doc>
     <title>A Document</title>
     This is element free text.
     <chapter>This is a chapter</chapter>
</doc>
```

For this reason, the XML literature often mentions "document tree nodes" where one might expect a reference to "elements". The document instance of the previous example yields a document tree consisting of a root, which is an element of type "`doc`" with three child nodes that are, respectively, (1) an element of type "`title`", (2) the element free text "`This is element free text.`", and (3) an element of type "`chapter`".

Note that XML document instances and therefore XML document trees are ordered: Changing the order of the children of a node would result in a different XML document and a different XML document tree.

2.4.5 Well-Formed Documents

A document is *well-formed* if it enforces basic syntactic rules forbidding for example that opening tags `<a>` and `<b>` in that order are followed by corresponding closing tags in the reverse order:

```
<a> ... <b> ... </a> ... </b>
```

For the document to be well-formed, the tags have to be properly nested like left and right parentheses:

```
<a> ... <b> ... </b> ... </a>
```

2.4.6 Attributes

Elements may be assigned *attributes*. An attribute has a name and a value. In the following, the element of type "`card`" is assigned an attribute with name "`status`" and value"`important`", the subelement "`phone`" is assigned an attribute with name "`country`" and value"`Germany`":

```
<card status="important">
   <last-name>Eisinger</last-name>
   <first-name>Norbert</first-name>
   <email>eisinger@informatik.uni-muenchen.de</email>
   <phone country="Germany">08921782218<phone>
</card>
```

Quoting in XML, for example of attribute values, can use either simple quotes `'` or double quotes `"`, as long as the same characters are used as opening and as closing. The quoted text may contain the other quotation character, for instance `title="ma'am"`

Attributes are controversial, for they are often considered to be redundant. Indeed, one often can replace an attribute "`name="value"`" by a subelement of type "`name`" with content "`value`". Thus, the previous example could be replaced by the following attribute-free element:

```
<card>
   <status>important</status>
   <last-name>Eisinger</last-name>
   <first-name>Norbert</first-name>
   <email>eisinger@informatik.uni-muenchen.de</email>
   <phone>
      <country>Germany</country>
      <number-in-country>08921782218</number-in-country>
   <phone>
</card>
```

The following difference between elements and attributes is worth noting: While the content of an element might consist of one or of several subelements, which, in turn, might themselves contain subelements, an attribute value is always a flat string.

2.5 Defining Special Purpose Markup Languages in XML

Special purpose markup languages are defined in XML like in SGML through so-called *document type definitions*, short *DTDs*. Recently, it has been proposed to replace the DTD formalism by another one called *XML Schema* [60, 61, 62]. For reasons discussed below, XML Schema is a very promising alternative to DTDs.

2.5.1 Document Type Definition (DTD)

An XML document might include (or refer to) the definition of a *document type*. A *document type definition*, short *DTD* is basically a context free grammar. For example, the following document type *declaration* specifies a DTD which specifies the structure of the document instance of the previous example:

```
<!DOCTYPE address-book
   [<!ELEMENT address-book (card*)>
    <!ELEMENT card         (last-name, first-name, (email | fax), phone?)>
    <!ELEMENT last-name    #PCDATA>
    <!ELEMENT first-name   #PCDATA>
    <!ELEMENT email        #PCDATA>
    <!ELEMENT fax          #PCDATA>
    <!ELEMENT phone        #PCDATA>
   ]>
```

The document type *declaration* begins with "<!DOCTYPE" and ends with "]>". It contains a document type *definition* which begins with "<!ELEMENT address-book (card*)>" and ends with "<!ELEMENT phone #PCDATA>".

2.5.2 Internal DTD vs. External DTD

In the previous example, the document type definition is *internal*. Alternatively, it can be *external* as in the document type declaration:

```
<!DOCTYPE address-book SYSTEM "~bry/XML/DocTypes/dtd/addbook.dtd" >
```

In this case, the string "/XML/DocTypes/dtd/addbook.dtd" is passed to the underlying operating system so as to retrieve a file containing the document type definition. In this example, a Unix syntax is assumed for the file name.

2.5.3 #PCDATA

The reserved name `#PCDATA` used in the DTD given above simply means "text". In the XML Schema formalism [60, 61, 62], this simple type is called *string*.

2.5.4 Valid Documents

If the prolog of an XML document includes an (internal or external) DTD and if the structure of the document instance of this XML document is as defined by the DTD, then the XML document is said to be *valid*.

2.5.5 Abstract Syntax vs. Concrete Syntax

In XML, the syntax of the markup, that is of the tags, is called *abstract syntax*, while *concrete syntax* is used to refer to the syntax defined by a DTD. It is worth noting that, in general (for example in Compilers) these expressions are given other meanings.

2.5.6 DTDs and Context-Free Grammars

In principle, a DTD is a special context-free grammar in disguise: The element names are non terminal symbols, and the basic, or simple, type "`#PCDATA`" is a terminal symbol (to which the actual character string is associated during parsing as a *value*, as it is commonly done by compilers for instance for identifiers).

Note that context-free grammars defined by DTDs have the following particularities:

- Terminals symbols, namely the opening and closing tags, are always associated to nonterminal symbols.
- The empty word cannot be generated.

It is worth emphasizing that *empty elements* are no *empty words*.

Because of these particularities of context-free grammars defined by DTDs, a simple form of top-down parsers, the recursive-descent or LL1 parser [4], suffices to check the validity of XML documents whose structures are defined by DTDs.

2.5.7 Namespaces

So called *namespaces* [57] can be specified and used for avoiding name conflicts that might occur, for example when a DTD is composed from portions imported from different sources.

Note that a few predefined namespaces are part of the XML specification.

2.5.8 XML Schema

DTDs have the following characteristics, that are often considered harmful:

1. DTDs use a syntax distinct from that of document instances,
2. with DTDs, subelements with same names cannot be given distinct contents depending on the parent elements in which they occur,
3. modern data modeling concepts such as types, object classes and inheritance are not provided by DTDs.

It is rather inconvenient that DTDs use another syntax than the general XML syntax used for document instances, because this prevents from using identical tools, such as parsers, editors, and formatters for both, DTDs and document instances. In addition, the specific syntax of DTDs forces DTD designers to deal with two distinct languages, a first one for specifying DTDs, another one for specifying documents instances.

In many applications, it is natural to give the same names to some elements with different contents and occurring in different contexts. For example, it would be natural to have "name" elements within both, product and client data, even though product names and client names might have very different structures. With DTDs, this is not possible. Indeed, the DTD formalism is that of *context-free* grammars.

Types are often needed for specifying identical concepts occurring in different disguises. This is for example the case of "sender" and "addressee" in letters. While distinct element names make sense, the two will usually have the same structure. Using DTDs, one might have to define (part of) this structure twice. A formalism more sophisticated than that of DTDs offering user defined complex types would make it possible to define a type "address" and to refer to this type in specifying the structure of the elements "sender" and "addressee".

Finally, classes of objects with inheritance between classes and subclasses, the importance of which for data modeling is well known, are clearly missing in DTDs.

XML Schema [60, 61, 62] is a proposal aiming at overcoming the three above-mentioned drawbacks of DTDs. In the XML Schema formalism, one specifies the structure of documents and elements using kinds of templates with a structure very similar to that of the elements they specify. These templates are defined in a markup language which is itself defined in XML. For example, the following element specifies a *complex type* "address" in the XML Schema style:

```
<xsd:complexType name="address">
     <xsd:element name="first-name"    type="xsd:string"/>
     <xsd:element name="last-name"     type="xsd:string"/>
     <xsd:element name="street"        type="xsd:string"/>
     <xsd:element name="street-number" type="xsd:positiveInteger"/>
     <xsd:element name="city"          type="xsd:string"/>
     <xsd:element name="postal-code"   type="xsd:string"/>
     <xsd:attribute name="country"     type="xsd:NMTOKEN"/>
</xsd:complexType>
```

The following is an element (of a document instance) conforming to the above defined complex type "address":

```
<address country="fr">
     <first-name>Francois</first-name>
     <last-name>Bry</last-name>
     <street>rue de Vincennes</street>
     <street-number>70</street-number>
     <city>Bordeaux</city>
     <postal-code>F-31000</postal-code>
</address>
```

Note that two different kinds of elements, and therefore two different kinds of tags, are used in the previous example. The elements of type "`xsd:...`" belong to the schema defining the structure of (valid) document instances. These elements are defined using the special purpose markup language of XML Schema [61, 62]. They specify themselves the structure of those elements that might occur in a document instance, that is the elements of types "`address`", "`first-name`", "`last-name`", etc. The prefix "`xsl:`" refers to the XML Schema namespace.[4]

Unfortunately, the specification of XML Schema, that is [61, 62], is difficult to understand. By chance, [60] provides a more easily readable informal introduction to these rather difficult specifications.

[4]How such a prefix is bound to the XML Schema namespace is explained in [57] and [60].

Although it is not yet quite clear, how documents can be efficiently parsed against context-sensitive XML Schema specifications, XML Schema seems to have a promising future as a successor of the DTD formalism because it offers data modeling facilities such as complex types, classes, and inheritance that are nowadays common.

Note that former stages of the XML Schema proposal were called *XML Data*, *DDML* (for "Data Definition Markup Language"), and *XSchema*.

2.6 Localization Formalisms of XML

In defining hypertext links, in specifying the layout of document instances, and in retrieving data using a query language, one often has to "localize" elements in a document instance. This is the case if each occurrence of the string "XML" is to become an anchor for a link to the XML web page [50], if the first lines of all paragraphs but the first of each chapter have to be displayed with an indentation, or if graduate courses on markup languages have to be retrieved from the web pages of European universities. Two formalisms have been designed for such "localizations": XPath [65] and XPointer [66].

Both XPath and XPointer rely upon a path notation inspired from the path formalism used with Unix systems to localize a file, like for example:

```
~bry/XML/DocTypes/dtd/addbook.dtd
```

Like the Unix path formalism, XPath and XPointer give rise to *absolute* localization of elements, that is a localization referring to the root of a document tree, as well as *relative* localization of elements, that is a localization referring to a current position in a document tree.

In addition, XPath and XPointer give rise to precise characterizations of nodes in a document tree based upon element types, attribute names, attribute values, and strings occurring in the content of an element. For example, the XPath expression

```
address[street='rue de Vincennes']
```

characterizes elements of type "`address`" that have a subelement of type "`street`" containing the string "`rue de Vincennes`". Thus, XPath and XPointer can be used for specifying traversals through document trees in terms of element types, attribute names, attribute values, and strings encountered.

XPath has been designed for localizing *nodes* in the document tree. It is used by the transformation language XSLT [71] (see below). The intention of XPointer was to enhance XPath, so as to make the characterization of *ranges*, that is portions of text in a document, possible. Ranges are often needed as possible anchors of hypertext links. XPointer is used by the linking language XLink [63]. Note that XPointer makes it possible to specify ranges that span over several nodes of the document tree.

Although XPointer was intended as an extension of XPath, both formalisms evolved independently from each other. As a consequence, XPath is *not* a subset of XPointer. Most likely, a unifying localization formalism will be proposed soon.

2.7 Style Sheets

Style sheets are used to attach formatting, or rendering properties to a document. The name "style sheet" comes from the formatting menus of word processors.

Two style sheet languages are currently used for XML documents: Cascading Style Sheets, short CSS, [51, 52] and the Extensible Stylesheet Language, short XSL [69].

2.7.1 Cascading Style Sheets (CSS)

CSS was initially conceived for HTML [24]. A CSS style sheet consists of *CSS rules*. The left hand side of a CSS rule characterizes an element in a formalism reminding of

XPointer [66] and XPath [65], but much simpler. The right hand side of a CSS rule is a list of property-value-pairs like, for example, "`font-style: italic`".

The following CSS rules assign to HTML [24] elements some self-explaining formatting characteristics:

```
em     {font-style: italic; font-weight: bold; }
strong {text-transform: uppercase; font-weight: bold; }
h2     {text-transform: lowercase; }
h1     {font-size: 200%; }
body   {font-family: Verdama, sans-serif; }
```

The denomination "cascading" refers to a possible overriding: One might for example chose a font for elements of a certain type "`a`". This choice then applies recursively along the document tree structure for all direct and indirect subelements of all type "`a`" elements, except if another CSS rule specifies a conflicting property-value pair for such a subelement.

In case several CSS rules apply to a same element in spite of the overriding rule, then a rather simple *conflict resolution* algorithm makes the choice.

Note that the order of CSS rules in a CSS style sheet has no influence on the formatting specified by a CSS style sheet.

CSS has two version, CSS level 1 [51], short CSS1, and CSS level 2 [52], short CSS2. CSS2 builds on CSS1 and extends it. CSS2 improves over CSS1 by supporting media-specific formatting so that authors may tailor the presentation of their documents to visual browsers, aural devices, printers, braille devices, handheld devices, etc. CSS2 also supports content positioning, downloadable fonts, table layout, features for internationalization, that is formatting languages with other writing direction than Latin languages, automatic counters and numbering, as well as some properties related to user interface. With very few exceptions, all valid CSS1 style sheets are valid CSS2 style sheets.

2.7.2 The Extensible Stylesheet Language (XSL)

XSL [69] is based on other principles than CSS [51, 52]: While formattings specified using CSS cannot significantly change the structure of the document, except for example leaving out an element, XSL allows significant structure changes, such as generating a sorted index or a table of contents. Also, using XSL a list of references sorted by years of publication can be rendered sorted by author. It is worth emphasizing that such a restructuring involves a rather complex processing such as sorting and duplicating entries, for example if there are publications with several authors.

XSL consists of two formalisms:

- A transformation language called XSL transformation, short XSLT, [71].
- A formatting language called XSL Formatting Objects, short XSL-FO, [70].

XSLT is a simplification of the language DSSSL [17] (pronounced "dissel") which was designed for SGML [40]. XSLT is discussed below with the query languages it resembles.

2.8 XLink: A Kind of Entity Relationship Model

2.8.1 Links and Arcs

The XLink specification [63] defines a *link* as a relationship between an arbitrary number of *resources*. A *resource* can be (part of) an XML element, a string, an XML document, or "non XML" data such as graphic data. Thus, links are not necessarily binary, and if they are, they are not necessarily directed.

Using or following a link is called *traversal*. Although a link can connect an arbitrary number of resources, traversal always involves a pair of the link's resources. Information about how to traverse a pair of resources, including the direction of traversal and possibly application dependent information, is defined through *arcs*. Arcs are components of links. They serve to specify the so-called *traversal semantics* of the links. Each arc relates two resources of the link, called the *starting resource* and the *ending resource* of the arc.

The XLink formalism makes it possible to attach to links and arcs freely defined, application dependent roles (XLink attributes "`role`" and "`arcrole`") and titles (XLink attributes "`title`"). Thus, links can be used for data modeling like a kind of entity-relationship model [45] or for application specific software.

2.8.2 Hypertext Links

Links may also, but do not have to, be hypertext links. Everyone is nowadays familiar with the simple hypertext links made popular by HTML [24] and the World Wide Web. Such a simple hypertext link has the following characteristics:

- It involves a pair of resources called *anchors*,
- it is directed from the *source anchor* to the *destination anchor*,
- it is activated by the user,
- its traversal results in displaying the destination anchor of the link either in the current or in a new browser window.

XLink offers a much richer notion of hypertext link than HTML. For XLink, an hypertext link may involve an arbitrary number n of resources. How an n-ary hypertext link can be traversed is specified by its arcs. Thus, each arc of an XLink hypertext link might be seen as corresponding to an HTML hypertext link. In contrast to HTML, XLink offers several options for specifying:

- When an arc is to be traversed (XLink attribute "`actuate`"),
- how the ending resource of an arc can be displayed (XLink attribute "`show`").

An arc may be traversed at the following "times":

- When the user initiates traversal, for example by a mouse click (`actuate="onRequest"`),
- when the starting resource of the arc is being processed by the user agent (`actuate="onLoad"`).

Traversal of an arc causes the ending resource of the arc to be displayed at one of the following "places":

- In the same window as the starting resource of the arc and replacing the starting resource in this window (`show="replace"`),
- in a new window (`show="new"`),
- in the same window and in the same context as the starting resource, enabling so-called "stretch text" (`show="embed"`).

Further application dependent values of the attributes "`actuate`" and "`show`" are possible, thus giving rise to further extensions of the hypertext model.

2.8.3 Linkbases

Xlink makes it possible to specify hypertext links in a document that does not contain any of the links' resources. Note that all the constituting components of a link, especially its resources and arcs, have to be defined together, but the resources may be defined by URI references to anywhere in the World Wide Web. If a resource of a link is defined as the linking element itself or as one of its subelements, the resource is called a *local* resource of the link. Arcs are classified as "outbound", "inbound", or "third-party" depending on whether their starting resource or their ending resource or neither of the two is local. Though it is not required, any one link typically specifies only one kind of arc throughout, and thus might be referred to as an inbound, outbound, or third-party link.

Third-party links make it possible to specify all hypertext links of an application within a so-called "linkbase". While linkbases might give rise to a better data modeling, they require an expensive and sometimes conceptually difficult preprocessing. Indeed, arcs have to be "attached" to their starting resources, if they are to be traversed from these resources.

2.8.4 Integration of XLink within XML

Interestingly, links are specified by attaching specific attributes and specific subelements to elements of any type. This makes it very easy to extend an existing document, the structure of which is dictated by the need of an application, into a document with (hypertext or non-hypertext) links. However, this also reflects the loose integration of XLink into the "document tree model" of XML. This too loose integration makes it difficult to rely on XSL for formatting XML documents with hypertext links à la XLink.

2.9 Query and Transformation Languages

Query languages are needed when XML is used as a data model for databases. As discussed in [13, 68, 3] and in the first section of this paper, this is a very promising direction of research.

A number of database projects have investigated various aspects of using XML as a data model for databases, see in particular [1, 2, 9, 20, 30, 15, 21, 43, 11, 16, 32, 33, 31].

The articles [28, 6] explain and compare query languages proposed for XML.

2.9.1 XML-QL, YATL, and Lorel

From the database community, three languages inspired by SQL have been proposed for querying XML data: XML-QL [15], YATL [12], and Lorel [2]. Demos of these languages are available at [58, 27, 5, 42].

XML-QL, YATL, and Lorel can be seen as rule-based languages. An XML-QL, YATL, or Lorel query is a kind of rule with two parts: a *selector* and a *constructor*. The selector serves to select elements from XML data. In doing so, variables similar to SQL and Prolog variables are bound. The constructor of an XML-QL, YATL, or Lorel query serves to restructure the selected data by specifying novel arrangements of the variables bound during the processing of the selector part of the query.

2.9.2 XQL

From the document processing community, the query language XQL [67] has been proposed for XML. Like XML-QL, YATL, and Lorel queries, an XQL query has a selector part. However, an XQL query has no constructors. Thus, XQL does not permit to rearrange in a new structure the data selected by a query. Thus, XQL is less powerful than XML-QL, YATL, and Lorel.

2.9.3 Quilt

More recently, in June 2000, another query language for XML, called Quilt [59], has been proposed. Quilt aims at combining the advantages of the query languages formerly proposed for XML.

2.9.4 XSLT

XSLT [71] was not conceived as a query language, but as a transformation language. However, its expressive power and its use are very similar to that of query languages.

The main difference between XSLT and query languages is that XSLT is less declarative in the sense that it assumes a traversal of the document tree. In contrast, query languages proposed for XML leave the choice of a document tree traversal to the query evaluator.

More precisely, XSLT has two computing models: a *recursive computing model* and an *imperative computing model* that depend upon the XSLT constructs occurring in an XSLT program.

The recursive computing model assumes a depth-first traversal of the document tree. The imperative computing model leaves the specification of the document tree traversal to the programmer. XSLT provides with constructs such as "`for-each`" with which it is possible to successively examine subelements of an element.

2.10 Modeling Metadata

Platform for Internet Content Selection (PICS) [38]. PICS is a formalism for associating metadata with Internet content in order to control the access to Internet resources. PICS was originally designed to help parents and teachers control what children access on the Internet. PICS also facilitates other uses such as code signing and privacy. PICS offers a language of "labels" for describing URLs and a language of "PICSRules" for specifying filters called "profiles". Profiles allow or block access to URLs depending on the labels of that URLs.

Resource Description Framework (RDF) [39]. RDF is seen as a more general purpose successor technology to PICS. RDF integrates a variety of web-based metadata activities including sitemaps, content ratings, stream channel definitions, search engine data collection (web crawling), digital library collections, and distributed authoring. RDF uses XML an syntax.

2.11 The Document Object Model (DOM)

DOM [53] is a platform and language independent interface that allows programs and scripts to dynamically access and update the content, structure and style of documents. DOM provides a standard set of objects for representing HTML and XML documents, a standard model of how these objects can be combined, and a standard interface for accessing and manipulating them. There are two specifications of DOM: DOM Level 1 [54] and DOM level 2 [55].

2.12 Infoset

The specification "XML Information Set" [56], short infoset, describes an abstract data set which contains the useful information available from an XML document. Basically, the infoset specification provides with the abstractionneeded when dealing with an "XML document" which is generated, for example from a database, and not stored in a file.

3 Data Modeling with Markup Languages and Logic Programming

3.1 The XLP Scheme

The *eXtended Logic Programming (XLP) Scheme* is a research project starting at the University of Munich aiming at investigating the application of deduction methods to solving problems related to data modeling with a generic markup language such as XML. The XLP Scheme is based on

- first, viewing XML elements and documents and semistructured data items as extended terms, the syntax of which is specified by an XML DTD, in the XML-Schema formalism, or in any other convenient manner such as a grammar of some kind,
- second, on using rules syntactically similar to Prolog clauses for expressing element and data item transformations, views (in the database sense), and queries to collections of XML elements, XML documents, or more generally semistructured data items.

Within the project, a logic programming language with code name XLP is to be developed. This language will be used for investigating the following issues.

3.1.1 A Language for Rigid and Flexible Terms

Meta-level constructs such as path variables and expressions are used to enhanced the term language of first-order logic with *flexible* subterm dependencies. This notion is explained in the following.

First-order logic terms only allow to express *rigid* subterm dependencies. For example, one can express with the first-order logic term `a(..., b(...), ...)` that an element (or term) of type `b` is a direct descendant of an element (or term) of type `a`. In an XML-like notation, this rigid subterm dependency might be expressed as follows:

```
<a>
   ...
   <b>
      ...
   </b>
   ...
</a>
```

Enhancing the term language with meta-level constructs seems to be a promising approach to express the kind of more complex subterm dependencies used for processing XML data in a similar syntax. For example, relying on a special meta-level element of type `meta` and on attributes `type` and `occurrences`, one can express that intermediate elements of the given type are required between `a` and `b` with the given number of occurrences. With the convention that `type="_"` means any type and `occurrences="*"` means zero or more times, one can express that an element (or term) of type `b` is a descendant at any depth of an element (or term) of type `a` with arbitrary intermediate elements:

```
<a>
   ...
   <meta type="_" occurrences="*">
      <b>
         ...
      </b>
   </meta>
   ...
</a>
```

Since the depth of the element of type `b` is not fixed, it is appropriate to call such a term *flexible*.

Note that other operators than "*" are needed for expressing the rich subterm dependencies of formalisms like XPath and XPointer.

3.1.2 Term-Oriented vs. Path-Oriented Element Localization

Expressing subterm dependencies using an enhanced term language, as outlined above, gives rise to a semantically richer localization of elements than using paths as it is commonly done with XPath and XPointer. Indeed, *term-oriented localizations* make it possible to express a context for the element to be retrieved, while path-oriented localizations do not. For example, here is how one can localize elements of type `b` that are descendants at any depth of an element of type `a` and have a preceding sibling of type `c`:

```
<a>
   ...
   <meta type="_" occurrences="*">
      <c>
         ...
      </c>
      <b>
         ...
      </b>
   </meta>
   ...
</a>
```

It is worth noting that the formalism of XML-Schema presents interesting similarities with such a term-oriented localization.

To be investigated are (1) algorithms for an efficient term-oriented element localization, (2) possible restrictions of the term language enhancing the efficiency of such algorithms, as well as (3) the complexity of a term-oriented element localization.

3.1.3 Unification for Term-Oriented Localization and Parsing

With a term-oriented localization language, localization can be seen as a kind of unification. For such a localization, the standard unification of automated deduction and logic programming cannot be used, since instead of rigid terms, flexible terms are dealt with. Thus, an extension of unification to flexible terms is to be investigated.

Such an extended unification is promising for parsing a document instance against a specification of a document structure in terms of the XML Schema formalism [60, 61, 62]. Indeed, an XML Schema structure specification defines a document structure using a special kind of flexible terms. Thus, parsing amounts to "unifying" an "XML Schema term" and a "document instance term". Since XML Schema structure specifications are not necessarily context-free, such a unification is likely not to be a matching, but is likely to bind variables in both "terms" involved.

3.1.4 Negation as Failure in Query and Transformation Languages

Negation as failure is an issue with a language such as XML. It can occur at two distinct level:

- First, when elements are localized for query, transformation, or formatting purposes, negative context conditions are often desirable, for example for precluding sibling elements of some kind.

- Second, negative statements like in SQL queries and PROLOG clauses are likely to be useful in queries and transformation rules.

Thus negation at both, the term and the formula level, is desirable. Semantics and processing issues related to negation as failure in markup languages still remain to be investigated. These issues will be investigated within the XLP Scheme.

3.2 Further Research Issues Related to Logic Programming

In the following, a few further research issues in data modeling with markup languages related to logic programming are briefly discussed.

3.2.1 Proof Methods and Strategies

The expertise of the automated deduction and logic programming communities in proof methods and strategies is likely to be applicable to the rule languages that are used for querying and transforming XML data. In particular, forward and backward chaining methods seem applicable for processing XML queries and views. As for deductive databases, granularity is an issue, that is whether computation and deduction should be performed set-wise or item-wise.

3.2.2 Meta-level Data Modeling and Meta-Programming

The expertise of the logic programming community in modeling metadata is likely to be useful for the processing of the various kind of metadata considered by the markup language community: DTD, XML Schema, PICS, RDF, and to some extent XLinks are of concern.

The use of XML for defining meta-level specifications such as XML Schema, XML query languages, and XML transformation languages calls for frameworks for querying such metadata. For example, it might be interesting to query XML data that are XML queries, or integrity constraints, or transformations referring to other XML documents. Issues like the satisfiability and the finite satisfiability of such metadata, are likely to be of interest – after all, XML data are finite.

3.2.3 Applications of Logic Grammars

Finally, logic grammars as investigated in the logic programming and natural language processing communities [18] are likely to be applicable in the framework of markup languages.

Acknowledgment

Part of this article has been written while the first author was visiting the LaBRI (Computer Science Research Laboratory of Bordeaux, http://www.labri.u-bordeaux.fr) in March 2000 on a visiting professorship. The support of the University of Bordeaux is appreciated.

References

[1] S. Abiteboul. Querying Semi-Structured Data. In *Int. Conf. on Database Theory*, 1997.

[2] S. Abiteboul et al. The Lorel Query Language for Semistructured Data. *Int. Jour. on Digital Libraries*, 1(1):68–88, 1997.

[3] S. Abiteboul et al. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann, 2000.

[4] A. V. Aho et al. *Compilers – Principles, Techniques, and Tools*. Addison-Wesley, 1986.

[5] The BBQ Page. http://www.npaci.edu/DICE/MIX/ BBQ/, 2000.

[6] A. Bonifati and S. Ceri. A Comparative Analysis of Five XML Query Languages. *SIGMOD Record*, March 2000.

[7] P. D. Bra et al. An Extensible Data Model for Hyperdocuments. In *4th ACM Conference on Hypertext*, pages 222–231, 1992. http://wwwis.win.tue.nl/~debra/echt92/final.ps.

[8] Bioinformatic Sequence Markup Language (BSML). http://visualgenomics.com/.

[9] P. Buneman. Semistructured Data. In *ACM Symp. on Princ. of Database Systems*, 1997.

[10] Channel Definition Format (CDF). http://www.w3.org/TR/NOTE-CDF/submit.html.

[11] S. Ceri et al. XML-GL: a graphical language for querying and restructuring XML documents. *Computer Networks*, 31(11–16):1171–1187, May 1999.

[12] S. Cluet et al. Your Mediators Need Data Conversion! In *Proceedings of the ACM-SIGMOD International Conference on Management of Data*, pages 177–188, 1998. http://cosmos.inria.fr:8080/cgi-bin/publisverso?what=abstract&query=138.

[13] D. Maier. Database Desiderata for an XML Query Language. In *The Query Languages Workshop*, 1998.

[14] Document Content Description for XML (DCD). http://www.w3.org/TR/NOTE-dcd.

[15] A. Deutsch et al. XML-QL: A Query Language for XML. Submission to W3C, http://www.w3.org/ TR/NOTE-xml-ql/, 1998.

[16] A. Deutsch et al. Querying XML Data. *IEEE Data Bulletin*, 22(3):10–18, 1999.

[17] Document Style Semantics and Specification Language. ISO/IEC 10179: Information technologies/Processing languages, 1996.

[18] A. Gal et al. *PROLOG for Natural Language Processing*. Wiley, 1991.

[19] C. F. Goldfarb et al. An Online System for Integrated Text Processing. *Proc. American Society for Information Science*, 7:147–150, 1970.

[20] R. Goldman and J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In *Int. Conf. on Very Large Data Bases*, 1997.

[21] R. Goldman and J. Widom. Interactive Query and Search in Semistructured Databases. In *Int. Workshop on the Web and Databases*, 1998.

[22] N. Guarino, editor. *International Conference on Formal Ontology in Information Systems*. IOS Press, 1998.

[23] F. Halasz et al. The Dexter Hypertext Reference Model. *Communications of the ACM*, 27(2):30–39, 1994.

[24] HyperText Markup Language (HTML). http://www.w3.org/MarkUp/.

[25] Hypertext Transfer Protocol (HTTP/1.1). http://www.w3.org/Protocols/Classic.html, ftp://ftp.isi.edu/in-notes/rfc2616.txt.

[26] International Standard ISO/IEC 10744: Information technology – Hypermedia / Time-based Structuring Language (HyTime). Second edition 1997-08-01. http://www.ornl.gov/sgml/wg8/docs/n1920/html/n1920.html, 1997.

[27] The Lore Page. http://www-db.stanford.edu/lore/ demo/, 2000.

[28] M. Fernandez and others. XML Query Languages: Experiences and Exemplars. http://www.w3.org/1999/ 09/ql/docs/xquery.html, 1999.

[29] Mathematical Markup Language (MathML). http://www.w3.org/TR/REC-MathML.

[30] J. McHugh et al. Lore: A Database Management System for Semistructured Data. *SIGMOD Record*, 26(3), 1997.

[31] H. Meuss et al. Towards Aggregated Answers for Semistructured Data. Technical report, Institute for Computer Science and Center for Computational Linguistic, University of Munich, 2000. Submitted for publication, http://www.pms.informatik.uni-muenchen.de/publikationen/PMS-FB/PMS-FB-2000-6.ps.gz.

[32] F. Neven and T. Schwentick. Query Automata. In *ACM Symp. on Princ. of Database Systems*, 1999.

[33] F. Neven and T. Schwentick. Expressive and Efficient Pattern Languages for Tree-Structured Data. In *ACM Symp. on Princ. of Database Systems*, 2000.

[34] Ontology.org. http://www.ontology.org/.

[35] Open Software Description (OSD). http://www.w3.org/TR/NOTE-OSD.html.

[36] Platform for Privacy References (P3P). http://www.w3.org/P3P/introduction.html, http://www.w3.org/TR/P3P/.

[37] Precision Graphic Markup Language (PGML). http://www.w3.org/TR/1998/ NOTE-PGML.

[38] Platform for Internet Content Selection (PICS). http://www.w3.org/PICS/.

[39] Resource Description Framework (RDF). http://www.w3.org/RDF/.

[40] Standard Generalized Markup Language (SGML). ISO 8879:1986.

[41] Synchronized Multimedia Integration Language (SMIL). http://www.w3.org/TR/ REC-smil.

[42] The Strudel Page. http://www.research.att.com/sw/ tools/strudel/, May 2000.

[43] D. Suciu. An Overview of Semistructured Data. *SIGACT News*, 29(4), 1998.

[44] Scalable Vector Graphics (SVG). http://www.w3.org/Graphics/SVG/Overview. htm8.

[45] B. Thalheim. *Entity-Relationship Modelling – Foundations of Database Technology.* Springer-Verlag, 2000.

[46] Vector Markup Language (VML). http://www.w3.org/TR/NOTE-VML.

[47] World Wide Web Consortium (W3C). http://www.w3.org/.

[48] HTTP Extensions for Distributed Authoring (WEBDAV). http://andrew2.andrew. cmu.edu/rfc/rfc2518.html.

[49] Extensible HyperText Markup Language (XHTML 1.0). http://www.w3.org/TR/ xhtml1.

[50] Extensible Markup Language (XML). http://www.w3.org/XML/.

[51] Cascading Style Sheets, level 1 (CSS1). http://www.w3.org/TR/REC-CSS1, December 1996. Revised January 1999.

[52] Cascading Style Sheets, level 2 (CSS2. http://www.w3.org/TR/REC-CSS2/, May 1998.

[53] Document Object Model (DOM). http://www.w3.org/DOM/, 1998.

[54] Document Object Model (DOM) Level 1 Specification. http://www.w3.org/TR/REC-DOM-Level-1/, 1998.

[55] Document Object Model (DOM) Level 2 Specification. http://www.w3.org/TR/DOM-Level-2/, 2000.

[56] XML Information Set. http://www.w3.org/TR/xml-infoset, 2000.

[57] Namespaces in XML. http://www.w3.org/TR/REC-xml-names/.

[58] The XML-QL Page. http://db.cis.upenn.edu/XML-QL/, May 2000.

[59] Quilt. http://www.almaden.ibm.com/cs/people/chamberlin/quilt.html, 2000.

[60] XML Schema Part 0: Primer. http://www.w3.org/TR/xmlschema-0/, April 2000.

[61] XML Schema Part 1: Primer. http://www.w3.org/TR/xmlschema-1/, April 2000.

[62] XML Schema Part 2: Datatypes. http://www.w3.org/TR/xmlschema-2/, April 2000.

[63] XML Linking Language (XLink) Version 1.0. http://www.w3.org/TR/xlink/, July 2000.

[64] XOL Ontology Exchange Language. http://www.ai. sri.com/ pkarp/xol/, 1999.

[65] XML Path Language (XPath) Version 1.0. http://www.w3.org/TR/xpath, November 1999.

[66] XML Pointer Language (XPointer) Version 1.0. http://www.w3.org/TR/WD-xptr, June 2000.

[67] XQL (XML Query Language). http://metalab.unc.edu/xql/xql-proposal.html, August 1999.

[68] XML Query Requirements. http://www.w3.org/TR/ 2000/WD-xmlquery-req-20000131, 2000.

[69] Extensible Stylesheet Language (XSL). http://www.w3.org/TR/xsl/, March 2000.

[70] XSL Formatting Objects. http://www.w3.org/TR/xsl/slice6.html#fo-section, March 2000.

[71] XSL Transformations (XSLT) Version 1.0. http://www.w3.org/TR/xslt/, July 2000.