ERX: A Conceptual Model for XML Documents

Giuseppe Psaila
Politecnico di Milano, Dipartimento di Elettronica e Informazione
Piazza L. Da Vinci, 32
I-20133 Milano, Italy
psaila@elet.polimi.it

ABSTRACT

The *eXtensible Mark-up Language* (XML) is able to represent any kind of structured or semi-structured document, such as papers, web pages, database schemas and instances, style-sheets, etc.. However, the tree-structure of XML documents, induced by nested mark-ups, does not provide a sufficiently expressive and general conceptual model of data in the documents, particularly when multiple source documents are processed at the same time.

This paper proposes the ERX (*Entity Relationship for XML*) conceptual model, an evolution of the Entity Relationship model that copes with the peculiar features of XML. ERX is devised to provide an effective support to the development of complex XML processors for advanced applications. By discussing an applicative scenario, the paper shows that suitable CASE tools can provide a practical support during the implementation of XML processors, by the automatic generation of software components based on the ERX model.

Categories and Subject Descriptors

I.7.m [**Document and Text Processing**]: XML Processors, Modeling XML Documents

1. INTRODUCTION

XML, the *eXtensible Mark-up Language* ([4; 11]) proposed by the World Wide Web Consortium (W3C), is the object of an increasing interest by researchers and industries. XML has been designed to become the format to exchange documents over the Internet, but is becoming a standard to represent any kind of documents in any application domain. This is due to the fact that it is able to represent any kind of structured or semi-structured document, such as papers, manuals, technical reports, web pages, database schemas and instances (see XML-Data [10]), style-sheets (see XSL [1]), etc.. XML is also able to represent the structure of other XML documents (see DOM [2] and SOX [7]), and meta-data (see RDF [9]). As a consequence, new complex and advanced applications designed to operate in Internet/Intranet environments are being developed, which use XML both to exchange data with other applications and as an internal representation [6]. In many cases, such applications deals with large repositories of documents, that must be maintained, consulted (see query languages [5; 8]), and published (for instance via Internet) with different user-dependent styles.

Since the amount of data may be large and the structural complexity of data may be significant, a conceptual model of data is necessary before starting the implementation of those software components, called *XML Processors* [4], that process the XML documents. However, the tree-structure of XML documents, induced by nested mark-ups and usually adopted by XML parsers, does not provide a sufficiently expressive and general conceptual model of data in the documents, because concepts of a given type are dispersed in the tree and it is not possible to view them as collections of homogeneous concepts. This is particularly evident in the case of XML processors that process multiple source documents at the same time: in fact, relationships between different document classes may be intricate to model and manage using the tree representation.

This paper proposes the ERX (i.e. *Entity-Relationship for XML*) conceptual model, an evolution of the Entity Relationship model (see [3]) that copes with the peculiar features of XML. ERX is devised to provide an effective support to the development of complex XML processors for advanced applications, by giving a conceptual model to the collection of documents that puts in evidence classes of concepts and their relationships, even across different classes of documents. ERX is not based on nested structures and links among them: it exploits a flat representation to put in evidence XML concepts and relationships among concepts. Since it is not constrained by the syntactic structure of XML, it is specifically focused on data management issues (differently from known proposals, such as DOM [2] and SOX [7], which are primarily focused on the structure of XML documents).

ERX is also devised to be of practical aid in the development of XML processors: by discussing an applicative scenario, the paper shows that suitable CASE tools can be very useful, if they are provided with powerful capabilities such as the automatic generation of software components based on the ERX model.

The paper is organized as follows. Section 2 discusses a simplified, but inspired by reality, case concerning the treatment of papers and style-sheets. Section 3 illustrates an applicative scenario, which shows how the ERX conceptual model

```
<!ELEMENT BIBITEM (#PCDATA)>
<!ATTLIST BIBITEM label ID #IMPLIED>
<!ELEMENT BIBLIO (BIBITEM)+ >
<!ELEMENT CITE EMPTY >
<!ATTLIST CITE label IDREF #REQUIRED >
<!ELEMENT EMPH (#PCDATA) >
<!ELEMENT SECTION (#PCDATA | CITE | EMPH)* >
<!ATTLIST SECTION title CDATA #REQUIRED >
<!ELEMENT TITLE (#PCDATA) >
<!ELEMENT PAPER (TITLE,(SECTION)*,(BIBLIO)?)>
<!ATTLIST PAPER name ID #REQUIRED>
```

Figure 2: The DTD for structured papers

might be profitably used. Section 4 introduces the ERX conceptual model. Finally, Section 5 draws the conclusions.

2. AN EXAMPLE: STYLE-SHEETS

In order to illustrate the ERX conceptual model in Section 4, we introduce a practical example.
Consider the problem of defining and processing style-sheets applied to documents. The goal of a style-sheet is to provide style rules that are applied to a given class of documents; the style-sheet defines the way each element (tag) of the document will appear on the output device (e.g. a HTML page displayed by a browser). The aim of style-sheets is to separate the presentation of a document and the content of the document.
In our example, the class of source documents is constituted by papers. We show a sample paper in Figure 1. The XML structure of such a document is defined by the DTD (*Document Type Definition*) reported in Figure 2. A paper is identified by its name (attribute `name` of tag `PAPER`). It has a mandatory title as content of tag `TITLE`. The body of the paper is constituted by a possibly empty list of *sections*; each tag `SECTION` has a mandatory attribute `title`, describing the section title. The content of a section can be a possibly empty sequence of generic text (`#PCDATA`), emphasized text (tag `EMPH`) and citations (tag `CITE`). While `EMPH` has only the content (the text to emphasize), `CITE` is an empty tag where attribute `label` specifies the label of the cited bibliographic item. Finally, the *bibliography* is described by a possibly missing and a non-empty tag `BIBLIO`, which contains a non empty list of *bibliographic items* (tag `BIBITEM`). The `BIBITEM` tag has associated a possibly missing (`#IMPLIED`) attribute named `label`: if specified, this attribute identifies the bibliographic item and can be referenced by `CITE` tags inside sections. The content of tag `BIBITEM` is the description of the bibliographic reference.
The XML source of the paper displayed in Figure 1 is reported in Figure 3.

2.1 The style-sheet

The content of the paper of Figure 3 is structured by means of tags, but nothing is said about the presentation style of the document. It can be specified by a style-sheet, which is independent of the particular paper instance.
Let us now introduce our sample style-sheet model, by discussing the portion of style-sheet reported in Figure 4, concerning the style rule for bibliographic items. This style-sheet is a toy case, defined for the sake of clarity; however, we were inspired by the fundamental concepts of XSL [1].

```
<?xml version="1.0"?>
<!DOCTYPE docs SYSTEM "docs.dtd">

<PAPER name="sample doc">
  <TITLE>  Using bibliography citations.  </TITLE>
  <SECTION title="A Citation">
    Here we have an example of citation. We refer
    to <ENPH> Knuth's paper </ENPH>, referred in
    <CITE label="Knuth68"/>, which
    introduced the concept of attribute
    grammar. </SECTION>
  <BIBLIO>
    <BIBITEM label="ASU85">
       A.V. Aho, R. Sethi, J. D. Ullman,
       "Compilers: Principles, Techniques, Tools",
       Addison-Wesley, 1985. </BIBITEM>
    <BIBITEM label="Knuth68">
       D. E. Knuth,
       "Semantics of Context Free Languages",
       Math. System Theory, Vol. 2, 1968.
    </BIBITEM>
  </BIBLIO>
</PAPER>
```

Figure 3: The XML version of the sample paper.

The body of the style-sheet is constituted by a sequence of `TARGET-ELEMENT` tags. Each of them defines the presentation style for a specific tag (or element) in the source document, called the *target element*. The name of the target element is specified by the attribute `name`.
The style rule for the target element `BIBITEM` specifies that the label is displayed in italic font between a pair of braces, followed by the reference displayed in normal font. In detail, at first the `ALIGN` tag sets the alignment to *justified*, while the `FACE` tag sets the *italic* mode for the current font; then, the `WRITE` tag prints a square bracket, and the `WRITE-ATTR` tag prints the value of attribute `label`, associated to the processed `BIBITEM` tag. After that, the closed square bracket is printed, and the font face is reset to the normal mode. At this point, the `CHILDREN` tag processes the content of the processed `BIBITEM` tag (as a result, the reference data are printed). Finally, the process ends printing a new line.

3. APPLICATIVE SCENARIO

This section defines the applicative scenario for the ERX model. This is essential to understand the issues considered in the design of the model, and how it can be exploited to automatically generate software components of the XML processors.

3.1 Architecture of XML Processors

At first, we have to understand the architecture of an XML processor which exploits the ERX conceptual model.

Single DTD Processors are XML processors which deal with one single class of documents (thus, they process documents defined on one single DTD). The software architecture we propose for a processor belonging to this category is composed of the following components.

• **Parser.** The parser is an XML parser able to parse documents compliant with the DTD.

Using Bibliography Citations.

A Citation

Here we have an example of citation. We refer to *Knuth's paper*, referred in [Knuth68], which introduced the concept of attribute grammar.

BIBLIOGRAPHY

[ASU85] A.V. Aho, R. Sethi, J. D. Ullman, "Compilers: Principles, Techniques, Tools", Addison-Wesley, 1985.

[Knuth68] D. E. Knuth, "Semantics of Context Free Languages", Math. System Theory, Vol. 2, 1968.

Figure 1: The formatted paper.

```
<?xml version="1.0"?>
<!DOCTYPE style SYSTEM "style.dtd">

<STYLE-SHEET>
   .  .  .
  <TARGET-ELEMENT name="BIBITEM">
    <ALIGN type="justified"/>
    <FACE type="italic"/>
    <WRITE string="["/> <WRITE-ATTR name="label"/>
    <WRITE string="]"/> <FACE type="normal"/>
    <CHILDREN/> <NEWLINE/>
  </TARGET-ELEMENT>
</STYLE-SHEET>
```

Figure 4: The style-sheet of the running example.

• **XML-ERX Mapper.** The parsed documents are passed to a specialized component, called *XML-ERX Mapper*, which is responsible to translate the content of the source XML document into the ERX representation.

• **ERX Data Manager.** The ERX representation is managed by the specialized component called *ERX Data Manager*: it stores the ERX representation of the source documents, and provides suitable interfaces to the software components (the *XML-ERX Mapper* and the *Output Servers*) which work on the ERX representation.

• **Output Servers.** The components called *Output Servers* are responsible to produce the external effects for which the XML processor is developed. For example, XML documents might be processed in order to rewrite them into new XML documents compliant with a different DTD, to show documents in the browser window, to perform actions on external devices, and so on. Each of these processing tasks is performed by a specialized *Output Server*, and, if it is the case, there can be multiple *Output Servers* in the same processor.

Multiple DTD Processors are processors which have to process multiple classes of source documents (each class is defined by a different DTD) at the same time.

An example is the case of documents and style-sheets (Section 2), because a paper cannot be processed without knowing the related style-sheet, defined by a different DTD. Another case is represented by a tool that processes several collections of documents, which are similar from the conceptual point of view but are defined on different DTDs, to rewrite them into documents compliant with a unique DTD. The software architecture we propose for a processor be-

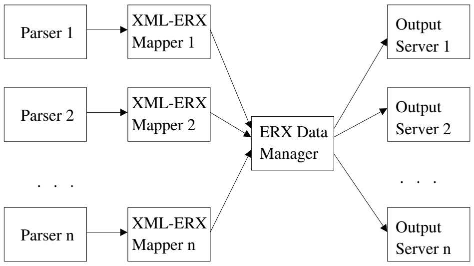

Figure 5: The architecture of a Multiple DTD Processor.

longing to this category is shown in Figure 5. In order to deal with multiple classes of source documents, the processor requires several parsers, one for each considered DTD (in practical solutions, it can be a general XML parser driven by different DTDs). Furthermore, each parser is coupled with a specific *XML-ERX Mapper* (as for parsers, these might be a unique, parameterized, software components). Observe that all the *XML-ERX Mappers* have the same target ERX model.

Discussion 1: The ERX Data Manager stores the ERX representation of the source XML documents. However, once the source documents are read, two distinct situations may occur.

1. The processor immediately processes the data and produces the outputs. In this case, the *ERX Data Manager* should use main memory data structures.
2. The processor does not process immediately the data; the actual processing phase may by performed in a subsequent time by a separate process. In this case, the *ERX Data Manager* must store the ERX representation on a persistent storage device.

In the second case, a Relational DBMS might be very useful. In fact, there are applications that require to store very large collections of documents to perform, e.g., queries on their content. Hence, the reliable technology of Relational DBMSs should be exploited for these applications.

3.2 A Development System

In order to make effective the use of ERX models in the development of practical applications, a good *Development System* is necessary. The minimal configuration for such a system should be the one reported in Figure 6.

• The first component of the development system is the *CASE tool*: it assists the designer both to design the ERX

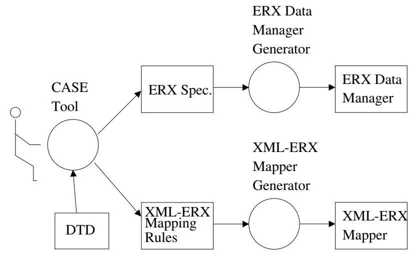

Figure 6: The architecture of the Development System.

conceptual model, and to define mapping rules from the source DTD to the ERX model. In the former case, the CASE tool produces a file with the specification of the particular ERX model (e.g. written in XML). In the latter, it produces a document containing the mapping rules.
• The ERX specification is read by the *ERX Data Manager Generator*, which generates an ERX Data Manager (a software library) able to manage ERX representation compliant with the model defined by the ERX specification.
• The XML-ERX Mapping Rules are read by the *XML-ERX Mapper Generator*, which generates an XML-ERX Mapper, a software library that maps the XML tree obtained by parsing the source document into the ERX representation.

If the illustrated development system is available, designers can take significant profit by it: in fact, by means of the CASE tool a declarative specification of the ERX model and the XML-ERX Mapping Rules can be defined (with the effect of automatically building a project documentation). Furthermore, the automatic generation of software components speeds up the implementation and improves the reliability of the developed XML processor.

4. THE ERX CONCEPTUAL MODEL

We are ready to introduce the ERX conceptual model. Concepts featuring ERX are discussed by means of the ERX model reported in Figure 7: this is the conceptual model for the case of papers and related style-sheet introduced in Section 2. We consider the case of a processor that processes multiple papers at the same time, all referring to the same style-sheet.

Entities. An entity describes a complex (structured) concept of the source XML documents. Entities are represented as solid line rectangles; the entity name is inside the rectangle. For example, a paper is an entity (the processor can process several papers at the same time), as are bibliographic items and sections. Note that ERX entities are different w.r.t. XML entities, which can be viewed as macros.
An *instance* of an entity X is a particular occurrence of a concept in the source documents. For example, an instance of entity `BIBITEM` is the first (or the second) bibliographic item of Figure 3.

Relationships. A relationship describes correlations existing between entities X and Y. A relationship is represented as a rhomb labeled with the name of the relationship, connected to X and Y by solid lines; these lines are labeled with a *cardinality constraint* $(l{:}u)$, which specifies for each instance of entity X (resp. Y) the minimum number l and the maximum number u of associated instances of Y (resp. X). For example, the relationship named `CITED`, between

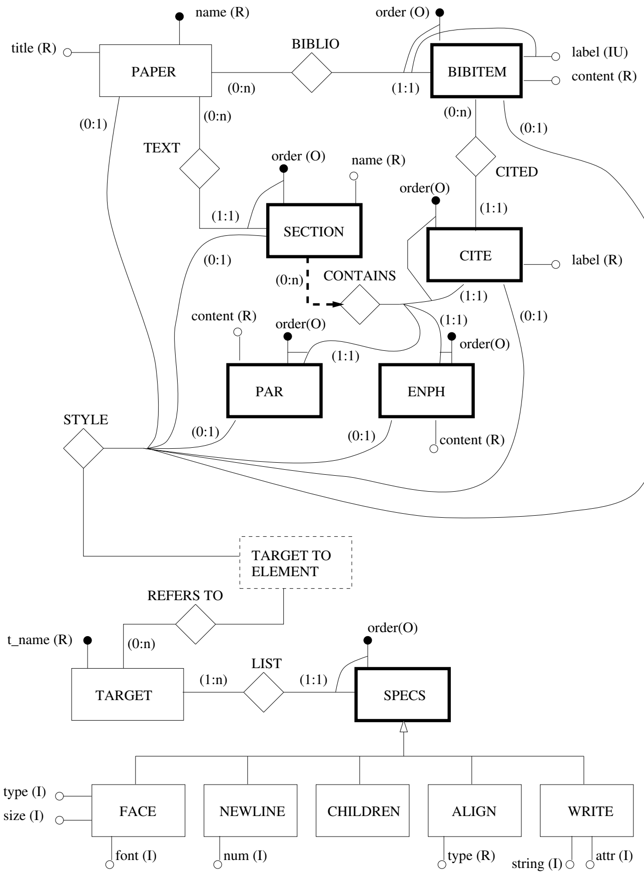

Figure 7: ERX conceptual model for papers and style-sheets.

the entities `CITE` and `BIBITEM`, says that an instance of `CITE` is associated to one and only one instance of `BIBITEM` (cardinality `(1:1)`), while an occurrence of `BIBITEM` can be associated to an unlimited number of occurrences of `CITE`, even none (cardinality `(0:n)`).

A complex form of relationship is represented by a *relationship with alternatives*: an instance of an entity X is associated with instances of alternative entities Y_1, Y_2, ..., Y_n; the cardinality constraint for X considers all associations of an instance of X with instances of any Y_1, Y_2, ..., Y_n. An example is the case of an instance of entity `SECTION`, that can be associated to zero or more instances (cardinality `(0:n)`) of entities `CITE`, `EMPH` or `PAR` (this latter describing text in sections) by the relationship `CONTAINS`.

Orthogonally, a relationship can be a *containment relationship*. Given two entities X and Y, a containment relationship from X to Y denotes that in the source XML document each instance of X contains instances of Y. Containment relationships are represented as normal relationships, except for the line on the X side, which becomes a dashed arrow entering the rhomb. In figure 7, `CONTAINS` is a containment relationship; it denotes that entities `CITE`, `EMPH`, and `PAR` are contained in a `SECTION`. Observe that the nesting of tags in XML documents would suggest to extensively use containment relationships in the ERX model; however, during the definition of the model, one should try to abstract as much as possible the actual document structure. This is why in Figure 7 only one containment relationship is used:

in fact, from a conceptual point of view it is relevant to evidence that sections are structurally composed of paragraphs, emphasized text and citations; conversely, for bibliographic items and style specifications this is not relevant, so that we obtain a more general model.

Attributes. Entities can have *attributes*: they represent elementary concepts associated to an entity. Attributes are represented as small circles, labeled with the attribute name, and connected to the entity they belong to by a solid line.
Usually, attributes of XML tags correspond to attributes of entities, as for attribute `name` of `PAPER`, or attribute `label` of `CITE`. But entity attributes can correspond to the textual content of tags as well. For example, the content of tag `EMPH` is represented by the `content` attribute of entity `EMPH`. Differently, the content of tag `TITLE` is represented by attribute `title` of entity `PAPER`: this is correct, since in the DTD of Figure 2 one and only one instance of tag `TITLE` is mandatory, and this tag contains only generic text. Furthermore, we might have attributes not appearing in the source document, whose value is determined by the XML processor.
A special category of attributes is constituted by *key attributes*, represented as black filled circles. Key attributes are those features of the entity that uniquely identify each instance of the entity. For example, a paper is identified by its name (attribute `name`), and a style specification for a target (entity `TARGET`) is identified by the attribute `t_name`.
Attribute names are associated to a qualifier, which indicates specific properties of the attribute. Qualifiers `(R)` and `(I)` denotes that the attribute is *required* or *implied* (i.e. optional), respectively; as an example, attribute `title` of entity `PAPER` is required, while attribute `num` of entity `NEWLINE` is implied. The qualifier `(O)` denotes that the attribute is an *order attribute*, i.e. its value is the order with which the instance of the entity appears in the document. The additional qualifier `U` (unique) can be specified for a non-key attribute (for key attributes it is implicit), meaning that only one instance can have a given value for the attribute (e.g. attribute `label` of `BIBITEM` is implied, but if specified its value must occur for one bibliographic item only). Finally, key attributes are qualified only by `(R)` or `(O)`.

Strong and Weak Entities. There are two types of entities. *Strong Entities* (denoted by thin lines) represent primary concepts in the source documents. This is the case, e.g., of entity `PAPER` (recall that we are considering the case of multiple papers processed at the same time) or style-sheet targets (entity `TARGET`).
Weak Entities (denoted by thick lines) are concepts defined in the context of another concept. For example, a section and a bibliographic item exist in a specific paper, hence they are modeled by the weak entities `SECTION` and `BIBITEM`, that are correlated with the strong entity `PAPER`. A weak entity can be defined in the context of another weak entity: weak entities `PAR`, `CITE`, and `EMPH` are correlated with the weak entity `SECTION`, since they are defined in the context of sections[1]. The implicit uniqueness constraint of key attributes of a weak entity X applies in the scope of the stronger entity Y: this is denoted by an edge connecting the key attribute(s) of X to the edge of the relationship connecting X to Y. For instance, the `order` attribute of entity `SECTION` indicates the position of a given section inside a given `PAPER`, thus the same number of section can be associated to multiple sections, provided that they belong to different papers. This is valid also in case of alternatives: inside a section, only one child can have a specific position, either a citation or an emphasized text or a generic text (`PAR`). Finally, consider attribute `label` of entity `BIBITEM`, qualified as `(IU)`: since it is connected to the relationship `BIBLIO` between `BIBITEM` and `PAPER`, the uniqueness constraint is applied only in the scope of a paper (a specific label cannot appear in multiple bibliographic items of the same paper, but can appear in several papers).

Hierarchies. Specialization hierarchies are possible in ERX. If an entity X is specialized into several sub-entities Y_1, Y_2, ..., Y_n, this means that an instance of the super-entity X is actually an instance of one of the sub-entities Y_1, Y_2, ..., Y_n. Key attributes can be specified only in the super-entity X, and all the attributes of X are common to (or inherited from) all the sub-entities, which can have specific attributes. For example, the style for a target is constituted by a sequence of *style specifications* (entity `SPECS`), which in turn are either face specifications (`FACE`), or new line specifications (`NEWLINE`), or references to children (`CHILDREN`), or alignment specifications (`ALIGN`), or write instructions (entity `WRITE`, in which both tags `WRITE` and `WRITE-ATTR` are fused); notice the definition of specific attributes for sub-entities, which correspond to attributes of the XML tags.

Interfaces. The last feature considered by ERX is the concept of *interface*. An interface divides two parts of the conceptual model that correspond to different classes of source documents, or parts that are homogeneous and semantically different w.r.t. the rest of the model. In the example of Figure 7, the upper half of the figure contains the conceptual model for papers, the lower part of the figure contains the conceptual model for the style-sheet. The interface `TARGET TO ELEMENT` (represented by a dashed line rectangle) plays the following roles: styles for targets (entity `TARGET`) are correlated with elements in the paper through the interface, that hides to styles the actual conceptual model of targets; on the other side, an element in the paper is correlated with the corresponding style through the interface, that hides the actual conceptual model of styles. Note that there is no cardinality constraint on relationship edges connected to the interface: in fact, an interface is not an entity, but a placeholder for entities in the two parts of the model.
This solution provides several advantages. At first, interfaces clearly maintain a semantic distinction between the different parts (or modules) of the model, but nevertheless they clearly put in evidence the correlations between these parts. Second, interfaces give rise to the concept of *view*: the XML processor may need operate only on a restricted view of the model, ignoring the rest of the model. For example, a sub-task of the processor may need to work only on the view concerning the styles, e.g. to build HTML templates that will be successively instantiated.

Notes. Note that not necessarily for all the tags present in the source XML documents there is a corresponding entity in ERX. This is the case of tag `TITLE`: only one occurrence of it is certainly present in the paper, and its informative content is described by a single attribute. Similarly, there is

[1]Containment relationships are orthogonal w.r.t. weakness of entities. In fact, we might have strong entities (describing primary concepts) whose instances appear inside other entity instances.

not a BIBLIO entity in the model: tag BIBLIO is a container of bibliographic items; thus, entity BIBITEM suffices.
Also notice that the textual content of tags is represented by attributes `content` of entities. If a tag has generic text mixed to other tags as children (as in the case of tag SECTION), it must be modeled by means of a relationship with alternative, where a specialized entity has the attribute `content` (entity PAR, weak entity of SECTION, where the attribute `order` maintains the position of the content string w.r.t. the other children of SECTION).

ERX does not provide specific constructs for concepts like links, navigation, etc., which are concepts typical of web sites and thus too specific. This is coherent with XML, which is meant to provide a common syntactic format to documents and data; web pages and site contents constitute a particular application, but they are not the only one. In fact observe that the three components of a web site (data schema, presentation and navigation) can be designed using a high level, site-oriented, conceptual model [6]; this model can be described by three distinct, although correlated, classes of XML documents. The XML processors on the server-side may manage such documents by means of the ERX data model; this way ERX can be exploited as document oriented, server-side, data model, supporting the specific site-oriented conceptual model.

Discussion 2: The adoption of an Entity-Relationship like model provides significant advantages.

• A collection of documents is viewed as a database, that might be queried in order to extract data about documents. Relational queries can perform complex data manipulations; consequently, complex queries on XML documents should be easily processed.
• The XML processor can easily access the data from different entry points w.r.t. the unique entry point (the root) provided by the tree structure. This solution avoids the waste of processing time in navigating the tree to search the desired data items, and allows a better specification of the data extraction and manipulation process.
• A unique general conceptual model can be used to model document classes which are conceptually similar but structurally different, because defined by different DTDs.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we address the problem of defining a conceptual model for XML documents. We propose the ERX conceptual model, an evolution of the classical Entity Relationship model. ERX provides specific features which are suitable to model large collections of XML documents.
ERX is devised to be effective for building advanced XML processors, specifically processors that have to manipulate complex XML documents, or multiple classes of documents at the same time. In particular, the database-like view of the modeled documents offered by ERX is a valuable factor of the proposal, both because it determines a better understanding of the data in the documents, and because it opens the way to the development of applications that exploit the well known and reliable technology of relational DBMSs to manage and manipulate large collections of documents.
The paper shows by means of the definition of an applicative scenario that suitable CASE tools can be developed to assist the development of complex XML processors based on the ERX conceptual model.

Future work. We plan to follow several directions. At first, we are going to test the effectiveness of ERX on complex cases coming from industry and/or bank applications; these tests will be the occasion to validate ERX, specifically as far as its completeness is concerned.
From the implementation point of view, we are realizing several versions of the ERX data manager; in particular, we are investigating both a main memory solution and an implementation that exploits a relational DBMS; in fact, this latter solution seems very promising to build complex and distributed information systems based on XML.
A third research line will consider the definition of a formalism to specify XML to ERX transformation rules; this formalism is the first necessary step toward the development of CASE tools that generates XML-ERX mappers.

6. REFERENCES

[1] S. Adler, A. Berglund, J. Clark, I. Cseri, P. Grosso, J. Marsh, G. Nicol, J. Paoli, D. Schach, H. S. Thompson, and C. Wilson. A proposal for xsl. Technical Report NOTE.XSL.html, W3 Consortium, Aug. 1997.

[2] V. Apparao, M. Champion, A. Le Hors, T. Pixley, J. Robie, P. Sharpe, C. Wilson, and L. Wood. Document object model (dom) level 2 specification. Technical Report WD-DOM-Level-2-19990304, W3 Consortium, March 1999.

[3] C. Batini, S. Ceri, and S. Navathe. *Conceptual Database Design: An Entity-Relationship Aprroach*. Benjamin Cummings, Menlo Park, California, 1992.

[4] T. Bray, J. Paoli, and C. M. Sperberg-McQueen. Extensible markup language (xml). Technical Report PR-xml-971208, W3 Consortium, Dec. 1997.

[5] A. Deutsch, M. Fernandez, D. Florescu, , A. Levy, and D. Suciu. Xml-ql: A query language for xml. Technical Report NOTE-xml-ql-19980819, W3 Consortium, Aug. 1998.

[6] P. Fraternali, S. Ceri, and S. Paraboschi. Data-driven, one-to-one web site generation for data-intensive applications. In *Proc. 25th VLDB Conference*, Edinburgh, Scotland, September 1999.

[7] M. Fuchs, M. Maloney, and A. Milowski. Schema for object-oriented xml. Technical Report NOTE-SOX-19980930, W3 Consortium, Sept. 1998.

[8] L. Lakshmanan, F. Sadri, and I. Subramanian. A declarative language for querying and restructuring the web. In *RIDE-NDS, Computer Society Press*, 1996.

[9] O. Lassila and R.R. Swick. Resource description framework (rdf) model and syntax specification. Technical Report REC-rdf-syntax-19990222, W3 Consortium, Feb. 1999.

[10] A. Layman, E. Jung, E. Maler, N. H. Mikula, J. Paoli, J. Tigue, H. S. Thompson, and S. DeRose. Xml-data. Technical report, W3 Consortium, Dec. 1997.

[11] R. Light. *Presenting XML*. SAMS, 1987.