Capturing Data Using XML Paragraph-centric Documents

Y. Badr, M. Sayah, F. Laforest, A. Flory
LISI - INSA de Lyon
20 av. Albert Einstein
69621 Villeurbanne Cedex, France
{youakim.badr, frederique.laforest, andre.flory}@insa-lyon.fr, msayyah@ul.edu.lb

Abstract

Nowadays, graphical forms are used to capture input data and feed traditional databases. They are associated with rigid schema and constraints to ensure data validity and to control entry sequencce. Over the Internet, electronic documents are becoming widely exchanged and their data reused in a large range of tasks. As a direct consequence, a capture interface that relies on documents can serve as a flexible front-end, and as a natural way to capture data. The aim of our research is the development of a comprehensive capturing and mapping data system that ensures flexible and well-adapted information capture and at the same time efficient information retrieval. This system introduces a transformation process that allows the mapping between the document model and the traditional database model. We validate our transformation frame-work by implementing a prototype. The primary release of the prototype is subject to a substantial attention.

1 Introduction

Several database applications capture their data using graphical forms. Standard forms contain fields with limited size and predefined data types. Often, these fields are associated with constraints to ensure data validity and to control entry sequence. The data is modeled in a convenient way to conform to rigid schema of traditional data model i.e. relational or object-oriented model. Capturing data using forms in applications such as patient record or technical report is not flexible and can't evaluate easily. Although traditional database query languages allow an easy and efficient information retrieval, the flexibility in capturing data remains an important criterion in evaluating software applications.

In recent years there has been an increased interest in managing data with irregular structure by using electronic documents. The term semi-structured refers to such data. At the logical level, the semi-structured data may have missing attributes. Existing attributes may have multiple occurrences, and the same attribute may have different types. For this reason, traditional capturing doesn't work with data structure that may change without notice. Electronic documents present flexibility to capture text data, to navigate through paragraphs, and can serve as a front-end for data entry.

SGML [1] and XML [2] are two syntactic meta-languages that provides the notation for defining textual markups, called tags. Tags present semi-structured data in documents and give semantic to their contents.

We briefly mention two types of semi-structured documents and highlight some of their common features and differences: data-centric documents, and paragraph-centric documents.

Data-centric documents are characterized by fine-grained data of small independent units of information, called elementary information. Usually they are used in machine consumption and electronic data exchange. In capturing data, data-centric documents are less constraining than database forms but still too limiting.

Paragraph-centric documents are in between data-centric documents and free text documents. They are characterized by larger grained data, usually tagged paragraphs of free text, where each paragraph contains or hides relevant information for a domain of interest. They are designed for human consumption and currently in wide use for being more flexible in capturing data than database forms and data-centric documents. They are also easier than free text documents in managing information.

The aim of our research is the development of a comprehensive capturing and mapping data system that ensures flexible and well-adapted information capture, and at the same time efficient information retrieval.

In our novel system data is captured paragraph by paragraph from which relevant information is extracted and filled in a domain-oriented database. Actually, the use of traditional databases proved to be the best way to query and retrieve information. A previous paper [6] has presented a step in this direction. It considers information extraction as simple pattern extraction. We now propose a more elaborate technique based on a transformation process.

Our transformation process allows the mapping between the paragraph-centric document model and the traditional database model. Transformation rules are applied to transform paragraph-centric documents into data-centric documents. Mapping rules, on the other hand, are used to feed the database from data-centric documents.

The remainder of this paper is organized as follows: section 2 discusses related work in querying XML documents and mapping documents into database, section 3 presents the architecture of our system, section 4 describes the transformation processor and illustrates it through transformation rules, section 5 concludes our work; it introduces the primary release of our prototype and proposes future extensions.

2 Related work

2.1 Query languages for semi-structured data

Many query languages for extracting and restructuring the contents of XML documents have been proposed, but none is recommended by W3C [7]. XML query languages consider XML as a data definition language. Some extend traditional languages [5], others are inspired by XML [4, 8] or designed for semi-structured data and can be used for XML [3, 9, 10]. In fact, the data expressed in XML is strikingly similar to semi-structured data. The advantage of this approach is to give the user a uniform interface to access and manipulate data. [11] and [12] highlight common features and differences of some of these languages. In general, the query applies to data-centric XML documents containing only tagged elementary information. It restructures data and returns new documents. Most of current query languages do not support join, aggregation, or abstract data types. Others do not offer manipulation languages to insert, delete, and update elements in collections of documents.

In addition, XML queries do not evaluate efficiently and are hard to formulate by casual users. Moreover, In general, XML query languages are not like high-level SQL standards, they still require enhencements to be efficient in retrieval.

2.2 Mapping data from XML documents to traditional databases

Another trend to query and manage semi-structured data in XML documents consists of mapping irregular data to a traditional database model.

The disadvantage of this approach is that users have to switch between two data models. Sometimes, the data is inefficiently stored and missing data corrupts relations. This approach leads us to distinguish between two types of mapping which are tackled in the following sub-sections.

2.2.1 Mapping data from XML data-centric documents

This type of mapping, supported by many systems, applies to elementary tagged information in data-centric documents. STORED [13] is a query language that automatically generates data-mining techniques to map data between documents and relational databases. It closely relates to Lorel [3], UnQL [9], and struQL [10] and applies only to structured data sources that have regular structure with few outliers. Within the same context, an extension of OQL, OQL-Doc [14] allows to query documents with a structure obeying some grammar. The mapping of a document into an object structure is based on the grammar of the document annotated with database operations. This approach only applies to documents with fairly regular structure and small portions of data.

2.2.2 Mapping data from XML paragraph-centric documents

Currently there is an increasing use of paragraph-centric documents. These documents contain tagged fragments of free text or paragraphs that hide relevant information. The structure of such documents has to be defined as regions of interest and the relevant information extracted from these regions have to conform to their semantics. NoDoSE [15] provides a semi-automatic system to extract data from instances of a document type. It consists of a general structure-mining algorithm that infers structure and of an extraction plug-in tool that provides relevant information. NoDoSE requires users interaction. It completely supports Text and HTML documents and partially paragraph-centric documents. A schema file and a suitable load file store extracted data in a traditional data model.

Based on an application ontology that describes a domain of interest. [16] and [17] formulate rules to extract relevant information from paragraph-centric documents and apply a recognizer to produce values of tuples attributes in a generated database schema. This approach is limited to relatively small ontological model and small set of constants/keywords such as dates, places, times, currencies, etc. The extraction phase is also purely syntactic and doesn't support semantics of natural language processing or constraints on relevant information.

In our research, we are studying a comprehensive capturing system based on paragraph-centric documents. Efficient information retrieval from these documents remains a critical criterion. As previously mentioned XML query languages are not yet well studied. In our work, we propose a novel approach to capture data entry with document model and map relevant information to database model. Later, traditional query languages are used to retrieve data.

Unlike studied mapping techniques [13,15,16,17], our approach is more general. It supports information extraction from free text, processing and structuring extracted information and feeds traditional databases. This novel approach is based on a transformation process controlled by different types of rules.

3 System Architecture & Overview

Our system provides a capturing interface for a domain of interest based on paragraph-centric documents. It hinges on a transformation processor that allows the mapping between the paragraph-centric document model and the traditional database model.
The transformation processor applies rules to extract relevant information from paragraph-centric documents, to re-structure them in order to produce XML data-centric documents, and to feed the database. Fig. 1 illustrates how the process works.

3.1 Capturing User Interface CUI

The Capturing User Interface, which is a graphical interface, provides the user with a list of DTDs. Each DTD presents an activity in our domain of interest and describes paragraphs to include in the paragraph-centric document instance. Once the DTD is chosen, the user fills its paragraphs with natural free text. The CUI then generates an instance of a paragraph-centric document and calls the transformation processor.

3.2 Transformation process

The transformation process allows the mapping between paragraph-centric documents and database model. It relies on two sets of rules; transformation rules and data mapping rules. As shown in Fig. 1, an implementation of a transformation processor reads the paragraph-centric document, its DTD, and looks for the corresponding transformation sheet.

The transformation sheet contains different types of rules. The complete process comprises the following transformation rules:
- Selection rules: select efficiently paragraphs.
- Extraction rules: extract relevant information.
- Processing rules: validate and process extracted data.
- Re-structure rules: restructure and formulate data into a data-centric document.

This process ends with paragraph-centric document to fill in traditional database according to mapping rules.

3.3 Visual Validation

An instance of paragraph-centric document may simply contain errors: misspelling, omissions, abbreviations, additions, jargon-words, etc… The extraction mechanism can actually be used to support semi-automatic error detection and correction. In other words, the extracted information in each paragraph is submitted back to the user for a visual validation. Another type of validation concerns the entire structured document instance before feeding the database. Lexicon database, thesaurus or other kinds of background domain knowledge may help to reduce incoherent information during the capture phase.

3.4 Mapping data to database

In the data-mapping phase, the database is fed with data from data-centric document instances. Data is modeled as a tree of data-specific objects (a view of classes). Generally, element types correspond to classes, and attributes and terminal elements of PCDATA contents correspond to properties. This model maps data directly to traditional databases using middle-wares or mapping techniques.

Our system is intended to be a framework for transformation and structuring problem in XML documents. Thus, rather than build one big tool, we have designed the system as a set of independent components that communicate through interfaces.

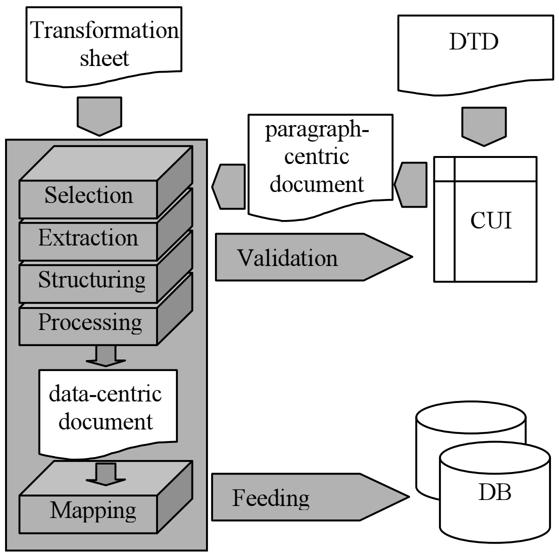

Figure 1: System Architecture

As a case study to apply our prototype, we consider patient records. Capturing patient records using traditional systems is a tedious task. In our system, capturing is more flexible, physicians write paragraphs like diagnosis, prescription etc… In background, all relevant information are extracted and stored in an existent medical database.

In the following sections, we describe in more details the transformation processor.

4 Transformation processor

The transformation processor is the core of our system. It is based on a transformation sheet of rules of different types. It reads both an XML document and its DTD, looks for the DTD transformation sheet, executes rules, and produces a new data-centric document.

In general, the transformation sheet consists of rules to select elements efficiently, to process element contents, to generate new elements, to output elements to the target document, and to feed traditional database. The target well-formed document may then contain, in addition to elements from the original document, a completely new set of elements generated by the transformation process. The result of the transformation is a data-centic document that conforms to a predefined DTD corresponding to the database structure.

The transformation processor cooperates with other components to extract information and map them to the database. A first version of our processor interacts with a graphical capturing component (CUI), an Information Extraction component (extractor), and a mapping component (mapper) through Java interfaces. Any component can be replaced independently from others. Thus, we can update or replace components without wadding through thousands of lines of source code.

The transformation processor manipulates the paragraph-centric document as a tree. The root is connected to its child nodes, which are XML elements and their contents. The transformation language operates, through its rules, by transforming one XML tree with leaves of larger grained data into another XML tree with leaves of fine-grained data. The resulting tree is serialized into a new well-formed XML document.

To clarify the transformation mechanism, let’s consider the paragraph-centric document shown in Fig. 2(a), which is composed of tagged paragraphs. Given this document, the processor looks for its DTD and loads its associated transformation sheet. Rules are applied to the document and a new data-centric document is produced. The final structure is depicted in Fig. 2(b).

```
< follow-ups >
    <diagnosis>
        bronchitis, 25/5/00 The patient complains
          of a loose caught and moderate fever
   </diagnosis>
  <previousHistory>
              father myocardial infraction
                                         1997
   </previousHistory>
  <prescription>
       3 pills telenol per day, 10 days
  </prescription>
< /follow-ups >
```

Figure 2(a) Paragraph-centric document

```
< follow-ups >
        <diagnosis> bronchitis</diagnostic>
        <date> 25/5/00 </date>
        <symptom>
          <illness> loose caught </illness>
          <illness> fever </illness>
        </symptom>
       <previousHistory>
          <illness>myocardial infraction</illness>
          <parent>father</parent>
          <date>1997</date>
       </previousHistory>
       <prescription>
           <quantity> 3 </quantity>
            < drug> telenol</drug>
            < frequency> day</frequency>
       </prescription>
   ….
< //follow-ups>
```

Figure 2 (b): Data-centric document

4.1 Transformation sheet overview

An important aspect of our system is the use of the transformation sheet. Indeed, the transformation sheet is an abstract model that describes the transformation mechanism. It contains rules executed in sequence to produce new data-centric documents and to map their data to the database. We distinguish five different types of rules:

- Selection rules to select elements efficiently, in our case tagged paragraphs.
- Extraction rules to extract relevant information from small corpus of free text.
- Processing rules to manipulate extracted information.
- Structuring rules to reorder the structure and generate new elements into the target document.
- Data mapping rules to feed database target document.

Each DTD has an associated transformation sheet. The processor uses it either to construct documentary fund or to fill traditional database. In the later case, the output contains intrinsic information and convenient structure to facilitate database feeding. Another advantage is that the transformation sheet is not embedded with the data (documents or DTD); any modification on transformation rules does not alter all documents.

As shown in Fig. 3, the transformation sheet is strongly inspired by the XSLT [18] style sheet; building the transformation processor on top of XSLT is a strategic choice. We used XML syntax to implement our rules as extension elements to XSLT style sheet. The motivation for taking this trend is to build a system based on XML background and especially to gain full advantage of XSLT features. By exploiting XML technology we benefit from future standards and its research arena.

A Transformation sheet contains a list of transform templates, similar to template-patterns in XSLT style sheet. A transformation template has a selection rule specifying elements it applies to and a list of processing and structuring rules to execute when the selection rule is matched. Mapping rules are also introduced within the template body.

A transformation template has the following syntax:

<Transform match="*pattern*">
  content
</Transform>

Each transformation template is defined by a *Transform* element. The attribute *match* and its value specify the selection rule. The *content* has restructuring, processing and data-mapping rules as sub-elements whose execution results in the target document.

The transformation processor manipulates the XML document as a tree of nodes. It scans it looking through each sub-tree in turn. As each tree in the XML document is read, the processor compares it with the selection rule of each transformation template. When a tree matches a selection rule's pattern, rules inside the transformation template are executed and returned values are tagged into the target document.

```
........
<Transform match='prescrition' >
 <Construct frame='prescription'>
   <Extract pattern= 'ER_QUANTITY'/>
     <Slot name='quantity'/>
     <Slot name='quantityUnit'/>
    </Extract>
    <Extract pattern='ER_FREQUENCY'/>
    <Extract pattern='ER_PERIOD'/>
 </Construct>
 <ClassMap>
   <ElementType Name='prescription'/>
   <ToClassTable>
     <Table Name='PRESCRIPTION'/>
   </ToClassTable>
   <PropertyMap>
     < ElementType Name='quantity'/>
     <ToColumn>
       <Column Name='clQuantity'/>
     </ToColumn>
   </PropertyMap>
     ............
</Transform>
```

Figure 3: Extract of transformation sheet

The data-mapping model is inspired by the data-specific object model and the syntax is similar to XML-DBMS mapping language [19]. Each data-mapping rule is a *ClassMap* element that specifies a table in database, to which an element type is mapped. The maps for table columns and a list of relations to other tables are described in the *content* of ClassMap element. A data-mapping rule has the following syntax:

<ClassMap> *content* </ClassMap>

The transformation processor invokes mapping component to feed database. Section 4.6 gives the details on data mapping.

4.2 Selection rules

A selection rule is a match pattern that identifies a particular element, a group of elements, or any part of the document. To write powerful match patterns, we use the XPath language [20]. Briefly, XPath addresses parts of the document by operating on the logical structure of XML document rather than its surface syntax. In support of this primary purpose, it provides basic facilities for manipulating of strings, numbers, and booleans.
We illustrate the selection rule syntax as an attribute-value pair of transformation template:

```
    <Transform match= XPath _Expression >
       …..
    </Transform>
```

The selection rule *match= XPath _Expression* allows a convenient syntax to match elements in paragraph-centric documents.

4.3 Extraction rules

Each element in a paragraph-centric document is a tagged paragraph of free text. The tags add semantics and define a context for their contents. Thus, the entire document describes an activity in a domain of interest, and each paragraph contains relevant information that describes part of this activity. One motivation for document fragmentation is to construct small fragments of natural language text. Often, Information Extraction (IE) systems are used to extract domain-specific information from short documents. It is a more limited task than full text understanding. A key component of any IE system is its set of extraction patterns [21], called also extraction rules, used to extract or filter information.

Paragraphs in paragraph-centric documents consist of a mixture of grammatical or ungrammatical text. In order to handle this problem, the extraction rules should combine syntactic/semantic constraints with delimiters that bound the text to be extracted. The choice of such rules has a strong impact on generalizing the problem and building up a useful prototype.

Given the choice of selecting an IE system, the WHISK [22], for example, could be a good fit. WHISK extraction patterns have two components: the first describes the context that makes a phrase relevant (context-based), and the second specifies the exact delimiters of the phrase to be extracted (delimiter-based).

An extraction rule is inserted in the transformation template and has the following syntax:

```
    <Extract pattern = 'Extraction_rule'/>,
```

where *Extraction_rule* is a pointer to a point entry in the dictionary of text extraction rules. When the extraction rule returns multi-slot values, it takes the following syntax:

```
    <Extract pattern = 'Extraction_rule'>
      <Slot  name='slotname_1'/>
        …..
      <Slot  name='soltname_n'/>
    </Extract >
```

Sometimes it is useful to seek for the identifier of an expression in a catalog. The LookupKey element returns in the target output the corresponding value and has the following syntax:

```
    <LookupKey of='Expression' catalog='Catalogname'/>
```

where Expression is an Xpath expression.

By referring to our previous discussion, processing natural language does not represent the aim of our work. Our framework is relatively new research area at the intersection of processing natural language, XML and traditional databases. Our goal, by processing natural language, is to generate extraction rules and to extract relevant information in order to feed traditional databases for decision-support or mining processing. As writing useful extraction patterns by hand is a difficult and time-consuming task, and requires domain-specific knowledge, several research efforts have focused on learning the extraction patterns from training examples provided by the user. Various patterns could be generated by machine learning algorithms to build a dictionary of extraction rules.

4.4 Structuring rules

Structuring rules are used to tag relevant information and insert them into the target document. Their capabilities include a set of useful operations on elements such as:

- Generation of constant text.
- Suppression of contents (e.g. preparing data to be assigned to tables).
- Moving or reordering text (e.g. exchanging the order of the first and last name).
- Duplicating text (e.g. duplicate records in one-to-many relations.
- Creating new information in terms of the existing information.

Structuring rules arrange the extracted data into a well-formed document. They are mainly conceived to put the data in a format that's easy to feed the traditional database. They present a super-set of Standard XSL Elements: xsl:element, xsl:attribute, xsl:copy, xsl:value-of, xsl:text.

To gather a combination of extraction rules into the target output within a single element, the *construct* element type can be used; its tag name is given by the *frame* attribute.

```
<Construct frame='tagname' type='fragment|defragment'>
    content
</Construct>
```

If the *type* attribute is set to "defragment", the selected paragraph is copied to the target document and all extracted information are delimited by their tags.

4.5 Processing rules

Processing rules apply operations to the extracted information or elements. The result is tagged and inserted into the target document. The basic categories of operations are:

- String manipulation: comparaison, substitution, etc …
- Aggregation operations: Sum, Count, max, average …
- Arithmetic operations: +,-,/,*, etc.
- Relational operators: <, >, <=, >=, <>, =.
- Mathematical functions: ceiling, floor, abs...

A processing rule has the following syntax:

<value-of select = '*operation|function*'/>,

where the result of *operation* or the returned value of *function* is inserted in the target document. A useful sample to demonstrate a processing rule is shown below:

<value-of select='*$unit*$price'* />

where $unit and $price are variables holding valid values.

4.6 Filling in existing database

An important aspect in our approach is the use of database to store fine-grained data and not documents fragments or content. Thus, in order to fill in a database with data from documents, it is necessary to map document structure to database structure [23]. However, we distinguish two mapping models; the table model [24] and the data-specific object model [25]. In the first case, the XML document is modeled as a single table or set of tables. The mapping is not possible if XML documents contain deeply nested elements. In the second case, the XML document is modeled as

a tree of data-specific objects (a view of classes); generally, element types correspond to classes, and attributes and terminal elements (PCDATA) correspond to properties.
Sub-element types could be viewed as classes with an inter-relationship to their parent, or as properties. Thus an instance of the view is a tree of objects. This model maps directly to object-oriented and hierarchical databases and can be mapped to relational databases using traditional object-relational mapping techniques [26] or SQL 3 objects [27].

We suggest a mapping processor, or middleware, which stores data according to data-mapping rules defined in the transformation sheet. It should preserve the hierarchical structure, as well as elements' contents and the values of their attributes. Because we intend to store only data, information on documents management (entities, DTDs…) are not stored in the database.

The XML-DBMS [19] is a middleware for transferring data between XML documents and relational databases. The XML-DBMS mapping language relies on XML language that describes both how to model object view for an XML document and how to map this view to feed relational database. A formal syntax of data-mapping rule to map a single class (element type) is illustrated in Fig. 4. The element ClassMap specifies the table, to which the class is mapped, the maps for its properties and a list of related classes.

```
<ClassMap>
 <ElementType Name='element _viewed_as_class'/>
 <ToClassTable>
  <Table Name='table_in_database'/>
 </ToClassTable>
<PropertyMap>
<ElementType Name='element_viewed_as_property'/>
<ToColumn><Column Name='table_column'></ToColumn>
<.PropertyMap>
 <RelatedClass >
 <ElementType Name='sub-element_viewed_as_class'/>
<ForeignKey>
   <Column  Name='foreignKey_in_related_table'/>
  <ForeignKey>
  </RelatedClass >
</ClassMap>
                  .......
```

Figure 4: Extract of data mapping rules

If a sub-element is viewed as a class then its relationship with its parent element type must be established in the map of the parent class using the element RelatedClass.

5 Conclusion

In this paper, we have presented an approach joining document model and database model. XML paragraph-centric documents are typically well adapted to capture patient records, technical reports etc… They provide flexibility to capture free text, while traditional databases retrieve data efficiently. As shown earlier, our approach hinges on transformation rules to transform paragraphs of free text into data-centric XML documents and on data-mapping rules to fill in databases. Furthermore, the stored data can be used for statistical studies, decision-support or mining techniques e.g. interactions between drugs in the medical domain.

Future research trends include automatic generation of the transformation sheet, enhancing the capture user interface, and tuning validation.

A primary release of the prototype supports capturing, transformation, and mapping. In fact, we built our prototype on top of existent XML tools. The SAXON [28] package is extended and used as a part of the transformation processor. To parse XML documents, we used Apache Xerces parser. On the other hand, a primary version of our capturing user interface allows users to choose a DTD for a specific domain activity and to generate paragraph-centric documents. One of the critical aspects of the prototype is to extract relevant information form free text. In this primary release, extraction rules are hand written with regular expressions. In the future, we intend to use the WHISK learning system, which learns from training instances and can construct a dictionary of extraction patterns (rules). We also believe that mapping data from document model to database model is a subject of substantial research attention.

References

[1] ISO, *Information Processing – Text and office Systems- Standard Generalized Markup Language,* International Organization for Standardization, Rdf No. ISO 8879:1986,1986

[2] World Wild Web Consortium, *Extensible Markup Language (XML)1.0*, W3C recommendation 10-February-1998, URL:http//www.w3c.org/XML/

[3] Lorel S. Abitaboul, D. Quass, J. Widom, and J.L. Wiener. *The lorel query language for semi-structured data.* International Journal on Digital Libraries1997

[4] J. Robie, XQL: *XML Query Language*, URL:http://www.ibiblio.org/xql/xql-proposal.html 1999.

[5] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. *XML-QL: a query Language for XML.* International WWW conference 1999.

[6] F. Laforest, A. Flory. *The Electronic Medical Record: Using Documents For Information Capture*. In Medical Informatics Europe, Hannover, Germany, August 2000.

[7] The World Wide Web Consortium (W3C) Home page, URL:http://www.w3c.org/

[8] W3C, *XML Extensible Stylesheet Language (XSL)*, URL: http://www.w3.org/Style/XSL, In Proc. WWW8 Conf., 1999.

[9] P. Buneman, S. Davidson, G. Hillebrand, and D. Succiu. *A query language and optimization techniques for unstructured data (UnQL).* In Proc. SIGMOD Conf., 1996.

[10] M. Fernandez D. Florescu J. Kang A. Levy D. Suciu. *Catching the boat with Strudel: experiences with a web-site management system (StruQL).* In Proc of ACM-SIGMOD International Conference on Management of Data , 1998

[11] A. Bonifati, S Ceri. *Comaprative Analysis of five XML Query Languages.* SIGMOD Record, vol. 29 No. 1, March 2000.

[12] M. Fernandez, J. Simeon, P. Wadler. *XML Query Languages and exemplars.* URL: http:// www-db.research.bell-labs/user/simeon/xquery.html

[13] A. Deutsch M. Fernandez D. Suciu , *Storing semistructured data with STORED*, In Proc of the ACM SIGMOD International Conference on Management of Data , 1999.

[14] S. Abiteboul, S. Cluet, V. Christophides, T. Milo, G. Moerkotte, and J. Simeon. *Querying Documents in Object Databases (OQL-Doc).* Dec 5, 1996.

[15] B. Adelberg, *NoDoSE: a tool for semi-Automatically Extracing Structured and Semistructured Data from Text Documents.* Proceeding of ACM SIGMOD international conference on Management of data, 1998, pages 283-294.

[16] D. Embley, D. Campewll, R. Smith, S. Liddle. *Ontology-Based Extraction and Structuring of Information from Data-Rich Unstructured Documents.*

[17] D. Embley, D. Campewll, R. Smith, S. Liddle. *A Conceptual-Modeling Approach to Extracting data from the Web.*

[18] W3C, *XSL Transformation Language (XSLT),* URL:http:// www.w3.org/TR/xsl

[19] R. Bourret, *XML-DBMS: Middleware for Transferring Data between XML Documents and Relational Databases.* URL:http:// www.rpbourret.com/xmldbms/index.htm

[20] W3C, XPath Version 1.0, URL:http://www.w3.org/TR/xpath

[21] I. Muslea, *Extraction Patterns for Information extraction tasks: A survey*, Univ of Southern California.

[22] S. Soderland, *Learning Information extraction rules for semi-structured and free text* (WHISK) Journal of Machine Learning 1998.

[23] *Modeling Relational Data in XML* URL:http://www.extensibility.com/tibco/resources/modeling.htm

[24] *XML representation of a relational database* URL:http://www.w3.org/XML/RDB.html

[25] Scott W. Ambler, *Mapping Objects to Relational Databases*, URL:http://www.ambysoft.com/mappingObjects.pdf

[26] Mark L. Fussell, *Foundations of Object-Relational Mapping*, URL:http://www.chimu.com/publications/objectRelational/

[27] D. Brashear, *SQL Reference Page* URL:http://www.contrib.andrew.cmu.edu/~shadow/sql.html

[28] Michael Kay, *SAXON*: URL:http://users.iclway.co.uk/mhkay/saxon/