01-XW

World Multiconference on Systemics, Cybernetics and Informatics

July 22-25, 2001
Orlando, Florida, USA

PROCEEDINGS

Volume I

Information Systems Development

Organized by IIIS
International
Institute of
Informatics
and Systemics

Member of the International
Federation of Systems Research

IFSR

Co-organized by IEEE Computer Society
(Chapter: Venezuela)

EDITORS
Nagib Callaos
Ivan Nunes da Silva
Jorge Molero

XMLWrapper: Storing and Retrieving XML Documents Using Semantic Binary Object-Oriented Database (Sem-ODB)*

Li YANG, Naphtali RISHE, Jun YUAN, Maxim CHEKMASOV,
Luis PACHAS, Alejandro MENDOZA

High-performance Database Research Center
School of Computer Science
Florida International University
Miami, FL 33199, U.S.A.

ABSTRACT

As a fast growing data representation and exchange standard on the World Wide Web, XML has recently gained a lot of attention from both the academic and industrial communities. This paper addresses the issue of employing the power of database technologies on XML data. In this paper, a system called XMLWrapper, which is used to store and retrieve XML using the Semantic Binary Object-Oriented Database (Sem-ODB), is described. The Sem-ODB is based on Semantic Binary Object-Oriented Data Model (Sem-ODM), which is a data model that supports richer semantics than the traditional relational data model. The mapping mechanisms, which include the algorithms to map DTD constructs to Sem-ODM constructs and the meta-schema used to store the mapping information, are explained in detail in this paper.

Keywords: XML, DTD, Semantic Database, Semantic Schema, Mapping Mechanisms.

1. INTRODUCTION

XML, the eXtensible Markup Language [1], has become an increasingly popular data representation and exchange format in recent years due to its simplicity, flexibility, and self-describing nature. Unlike HTML (Hyper-Text Markup Language), which uses tags to specify the format of data, XML uses tags to represent the semantics of the data that it presents. Associating semantics with content makes XML documents not only human readable but also machine understandable.

There has been some research going on designing special-purpose systems to store XML documents ([10], [11]) as well as storing them in traditional databases ([3], [5], [6], [7], [8], [9]). Special-purpose XML data systems have the advantage of being well designed for fitting and tuning for "native" XML. However, the idea of starting from scratch rather than utilizing mature database technologies may not be easily accepted by people who are accustomed to traditional DBMSs. Though relational DBMSs have matured for the past 20 years, its structured two-level table and attribute based representation is not well suited for the nested structure of XML. In order to support XML, some extensions, such as support for set-valued attributes, must be added to reduce the fragmentation that may occur while mapping XML to relational tables [6]. Object-Oriented DBMSs support complex data types that can easily support the set-valued and nested structures in XML. We believe that it would be very natural to extend an OODB with XML-specific functions to support the storage and retrieval issues relevant to XML.

In our design of XMLWrapper, we use the Semantic Binary Object-Oriented Database System (Sem-ODB), as our underlying XML repository. Sem-ODB was developed at the High-Performance Database Research Center (HPDRC) [12] and is based on a conceptual data model, the Semantic Binary Object-Oriented Data Model (Sem-ODM [13]). It has been successfully deployed for highly complex applications such as applications intended for storage and processing of large amounts of earth science observations and the Terrafly Geographic Information System (GIS) [15].

The XMLWrapper project aims to exploit the power of Sem-ODB in modeling, querying and retrieving semi-structured data, while maintaining the autonomy of legacy database applications and XML applications. A key assumption is that XMLWrapper is based on the existence of a DTD (Data Type Definition) [2] for each XML document. DTDs are used to describe the structure of XML documents and are very much like schemas in databases. XMLWrapper can map any DTD to a Semantic Schema and then load XML documents conforming to this DTD into the Sem-ODB. After this, users can pose XML queries over the XML data and get results in XML without being aware of the underlying

* This research was supported in part by NASA (under grants NAG5-9478, NAGW-4080, NAG5-5095, NAS5-97222, and NAG5-6830) and NSF (CDA-9711582, IRI-9409661, HRD-9707076, and ANI-9876409).

storage they are using. The focus of this paper will be on the mapping mechanism between DTDs and Semantic Schemas.

1.1. Related Work

The potentials and limitations of using relational database engines for processing XML documents conforming to DTDs are discussed and explored in [6]. One big limitation of RDBMSs is that they do not support set-valued attributes. This might result in fragmentation when representing set sub-elements of XML in an RDBMS. [6] However, Sem-ODM supports 1-to-m attributes, which allows it to support set-valued attributes.

Storing and querying XML data using an RDBMS is also discussed in [8], but the focus of that paper is on mapping XML documents directly to relational tables; only very simple schemes are handled. It is not necessary for an XML document to conform to a DTD. However, without DTDs, tags in XML documents can be arbitrary and not interoperable. Though each XML document is self-describing, no common semantics for a group of similarly structured XML documents exists if they do not agree on some DTD. XML conforming to a DTD helps to solve problems in application interoperability and data integration on the Internet and business-to-business information exchange in E-commerce [3][4]. Hence, it is required that each XML document that users provide conforms to a DTD when using our approach.

Our approach is very similar to the one in [5], which describes an approach to mapping XML DTDs to relational schemas and to keeping the mapping information in a meta-schema to solve the problem of data model and schema heterogeneity. However, in our approach, we use the Sem-ODB (a kind of OODB) instead of a RDB as the XML repository. Sem-ODM has the features of Object-Oriented data models, such as inheritance, oid, explicit description of relationships and a high-level data model, while maintaining the simplicity of relational data models such as simple constructs. It is very easy to represent set-valued attributes and nested structures in Sem-ODM, thus avoiding the common problem of fragmentation when using RDBMSs to store XML. In our mapping mechanism, we first simplify DTDs before mapping them to Semantic Schemas.

1.2. Road Map

The rest of the paper is organized as follows. In Section 2, the XMLWrapper architecture and main modules are introduced. Section 3 first gives an overview on Semantic Schemas and DTDs, and then the data model mapping between DTDs and Semantic Schemas is further described in the rest of the section. We present our conclusions and plans for future work in section 4.

2. XMLWRAPPER ARCHITECTURE

As illustrated in Figure 1, there are five basic components, each of which may further consist of component modules, in the XMLWrapper architecture.

The first component is composed of two sub-modules: the **DTD Validator** and the **DTD Mapping Modules**. The DTD Validator is used to check that the DTDs users provided are syntactically correct. The DTD Mapping module maps the DTDs to Semantic Schemas and stores the mapping information and the meta-schemas of the DTD and corresponding semantic schema in the **KnowledgeBase**, which is the second component of the architecture.

The third component is composed of the **XML Parser** and the **XML Loader**. The XML Parser parses the XML documents that users intend to store in the Semantic DB and make sure that it is well-formed and valid according to the DTD that was validated in previous step. After this, The XML Loader extracts the XML data and stores it in the Semantic DB via SDB Engine APIs.

The fourth component processes XML queries that users input. First, the **XML Query Parser** parses the XML query and generates a parse tree, then the **Query Translator & Optimizer** translates the XML query into basic queries that are supported by SDB Engine APIs, generates an optimal execution plan, and sends it to the SDB Engine for execution. The last component, the **XML Document Generator,** reconstructs the data, which is retrieved from the underlying database system, into a readable XML document and returns it to users.

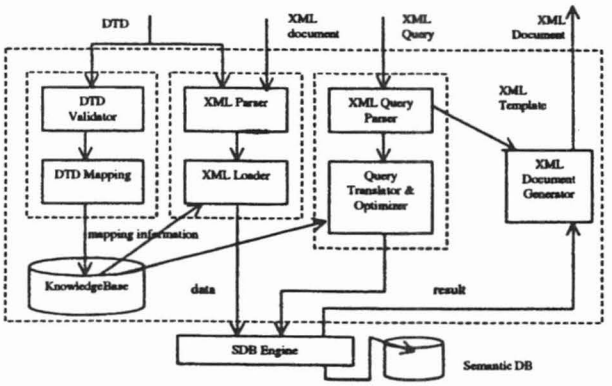

Figure 1. XMLWrapper Architecture

This paper will focus on the mapping component of the architecture and explain the mapping mechanisms in detail in the following section.

3. SCHEMA MAPPING

This section will describe how to map a DTD to a Semantic Schema of Sem-ODM. We first give a brief overview of the DTDs and the Semantic Schemas of Sem-ODM, then make a classification of DTD constructs and describe the mapping mechanisms drawn from the classification in XMLWrapper. Finally, this section will illustrate the meta-schemas used in XMLWrapper to help resolve data model heterogeneity.

3.1. DTD

A Data Type Definition (DTD) [2] describes the structure and constraints of XML documents. The purpose of a DTD is to define the legal building blocks of an XML document. Figure 2 is a DTD example that was extracted from [7] and is used as a running example in this paper.

```
<!-- book -->
<!ELEMENT book (front, body, references)>
<!ELEMENT front (title, author+, edition,
                 publisher)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (first, second, email?)>
<!ELEMENT first (#PCDATA)>
<!ELEMENT second (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT edition (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT body (part+ | chapter+)>
<!ELEMENT part (ptitle, chapter+)>
<!ATTLIST part id ID #REQUIRED>
<!ELEMENT ptitle (#PCDATA)>
<!ELEMENT chapter (ctitle, section+)>
<!ATTLIST chapter id ID #REQUIRED>
<!ELEMENT ctitle (#PCDATA)>
<!ELEMENT section (stitle, paragraph+)>
<!ATTLIST section id ID #REQUIRED>
<!ELEMENT stitle (#PCDATA)>
<!ELEMENT paragraph (#PCDATA)>
<!ELEMENT references (publications*)>
<!ATTLIST references reftype (book | article|
conferences | wwwaddress) "article">
```

Figure 2. A DTD Example

It can be seen from this DTD, that some elements and attributes are defined. Elements can be composed of sub-elements, which results in XML documents' nested structure. For example, each book can have front, body and references as its sub-elements. Each sub-element may have a cardinality of * or + or ?, which indicates how many times the sub-element can appear in that element. Elements may have attributes, for example, element chapter has an attribute named id which has an ID data type. Elements which are sub-elements of other elements can appear in some order (indicated by ,) or without any order (indicated by |). For example, for a book, its front must appear before its body and references, but both of parts and chapters can appear in the body of a book regardless of their ordering.

3.2. Semantic Schemas in Sem-ODM

There are two constructs, category and relation, that are used to describe a Sem-ODM. Categories are like *Entities* in an ER model except that the *Attributes* in an ER model are represented as relations in the Sem-ODM. A Category can either be a *Concrete Category* or an *Abstract Category*. Concrete Categories are categories like String, Number, and Boolean. Abstract Categories are composed of abstract objects. For example, categories like person and book are abstract categories. In a Semantic Schema there can be binary relations from an abstract category, which is called the *Domain* of the relation, to another category, which is called the *Range* of the relation. Relations from an abstract category to a concrete category correspond to attributes in an ER model. Relations from an abstract category to an abstract category are just like associations in an OO model.

With the help of relations in the Sem-ODM, the nested structure of XML can be represented in a Semantic Schema. Sem-ODM also has the concept of *Cardinality* of relations. There isn't any ordering concept in Sem-ODM. We would have to extend the relations in Sem-ODM with ordering to support this feature of XML. And we also add an XMLtype as a concrete category in the Sem-ODB to support XML-specific data types.

3.3. Mapping DTDs to Semantic Schemas

After describing the basic constructs of DTDs and Semantic Schemas, we show how to map DTDs to Semantic Schemas in this sub-section.

3.3.1. Simplifying DTDs As shown in Figure 2, elements in a DTD, especially composite elements that are composed of other elements, look much like regular expressions except that there is no ordering concept and explicit cardinality in regular expressions. A regular expression could be very complex, such as ((a|b*|c)*ab)*. In this sense, a DTD could also have very complex structure, for instance, <!ELEMENT e ((a,b,c)*|(d*,f))* >, though this might not be needed in practice. To ease the mapping process, we transform complex DTDs to simpler, but equivalent ones before performing the real mapping. This idea is adopted from [6][16].

3.3.2. DTD Constructs Classification While considering the mapping from DTDs to Semantic Schemas, we would usually start with mapping the constructs of DTDs to the constructs of Semantic Schemas. Before we get to the mapping algorithm, we need to make a classification of the information that we can get from a DTD.

The basic constructs of a DTD are elements and attributes, which are depicted in Figure 3 in the form of a Sem-ODM Semantic Schema. Therefore, mapping can be considered from these two perspectives, element-related and attribute-related mapping.

According to the content type of elements, elements can be classified into EmptyContent, AnyContent, MixedContent and ChildrenContent, as illustrated in Figure 4. Furthermore, MixedContent elements can be categorized as DataOnly, which only contains #PCDATA, and real Mixed, which contains both #PCDATA and sub-elements. A ChildrenContent element can either be a Choice element in which the ordering of sub-elements doesn't matter or sequence element in which the ordering of sub-elements matters.

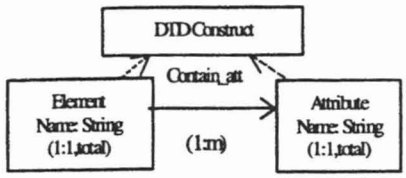

Figure 3. DTDComponent Schema

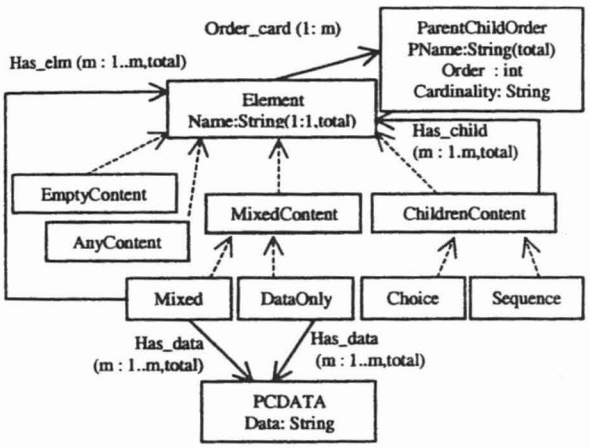

Figure 4. DTD Element Schema

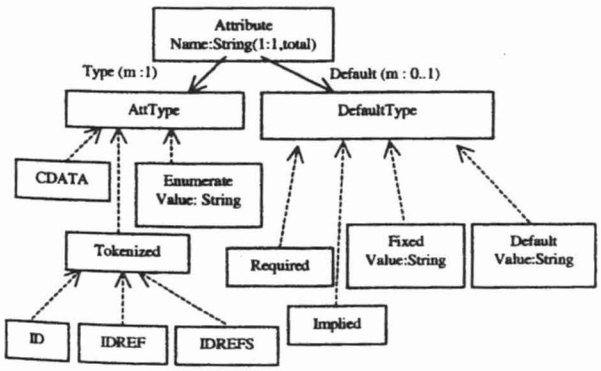

Figure 5. DTD Attribute Schema

Attributes in DTDs can be further refined according to their type and default, as shown in Figure 5. An attribute can have type CDATA, Tokenized, or Enumerate. Each attribute could have a FIXED, IMPLIED, DEFAULT, or REQUIRED default value.

3.3.3. Schema Mapping Based on the above categorization, we have come up with the following mapping rules related to elements.

(1) Every Element E of ChildrenContent, EmptyContent, and Mixed type is mapped to a Category E_C in the Semantic Schema. For example, the element book, which is of ChildrenElement type, is mapped to category Book in the Semantic Schema.
(2) For every Element E of AnyContent type,
- If E has parent element P (might be many), then E is mapped to a relation from the category E_P, which is the category corresponding to the element P in the Semantic Schema, to XMLType.
- If E doesn't have any parent element, E is mapped to Category E_C with a relation R from E_C to XMLType.

(3) For every Element E of DataOnly type,
- If E has parent element P (might be many), then E is mapped to a relation from the category E_P, which is the category corresponding to the element P in the Semantic Schema, to String.
- If E doesn't have any parent element, E is mapped to Category E_C with a relation R from E_C to String For example, the element first is mapped to a relation called first with category Author as its Domain and String as it Range.

To map relationships between elements and sub-elements, and relationships between attributes and elements in DTDs into relations in Semantic Schema, the following rules exist.

(1) Every sub-element E_i of element E of Choice type is mapped into a relation of its parent Category E_C, which is the category corresponding to E.
(2) Every sub-elements E_i of element E of Sequence type is mapped into a relation of its parent Category E_C, which is the category corresponding to E. In addition, we need pay special attention to the ordering of each sub-elements of E. We include this information in the corresponding relations of the Semantic Schema. For example, sub-elements front, body and references of element book are mapped to relations of Category book with ordering of 1, 2 and 3, respectively.

Another factor that should be considered is the cardinality of sub-elements. We represent cardinality of sub-elements in the corresponding relations as following:

(1) Cadinality * is mapped to 1:0..m
(2) Cadinality ? is mapped to 1:0..1
(3) Cadinality + is mapped to 1:1..m

(4) Any sub-elements appearing in their parent elements with no cardinalities specified are mapped as having 1:1 cardinality.

Attributes in DTDs are mapped to relations with the parent Category as the Domain and a Concrete Category as the Range. The mapping rules are generalized as follows.

(1) For <! ATTLIST E att CDATA>, where E is an element and att is its attribute of type CDATA.
Attribute att is mapped to a relation att: $E_C \rightarrow$ String (m:1).
(2) For <!ATTLIST E att eval1 |eval2 |eval3 >, where E is an element and att is its attribute of type Enumerate.
Attribute att is mapped to a relation att: $E_C \rightarrow$ Enumerate(m:1).
(3) For <! ATTLIST E att ID>, where E is an element and att is its attribute of type ID.
Attribute att is mapped to a relation att: $E_C \rightarrow$ String (1:1, unique).
(4) For <! ATTLIST E att IDREF>, where E is an element and att is its attribute of type IDREF.
Attribute att is mapped to a relation att: $E_C \rightarrow$ String (m:1).
(5) For <! ATTLIST E att IDREFS>, where E is an element and att is its attribute of type IDREFS.
Attribute att is mapped to a relation att: $E_C \rightarrow$ String (m:m).

For example, in <!ATTLIST part id ID #REQUIRED>, attribute id is mapped to a relation id: part→String(1:1)

As discussed in section 3.3.2, Attributes in DTDs **may** have default options. The mapping rules about the default options are generalized as follows.

(1) Attribute *A* with #REQUIRED default option: the corresponding relation is total,
(2) Attribute *A* with #IMPLIED default option: the corresponding relation is normal,
(3) Attribute *A* with #FIXED default option: no mapping for the #FIXED value,
(4) Attribute *A* with #DEFAULT default option: set default value for this relation in Sem-ODB.

For example, in the above attribute example, the attribute id of element part is mapped to a total relation id: part→String (1:1, total)

Based on the above mapping rules, we can transform the DTD in Figure 2 into the Semantic Schema shown in Figure 6.

Note that in Figure 6, the relations are enhanced with ordering which is denoted by the ordering number along with the relation name and + represents one or more cardinality.

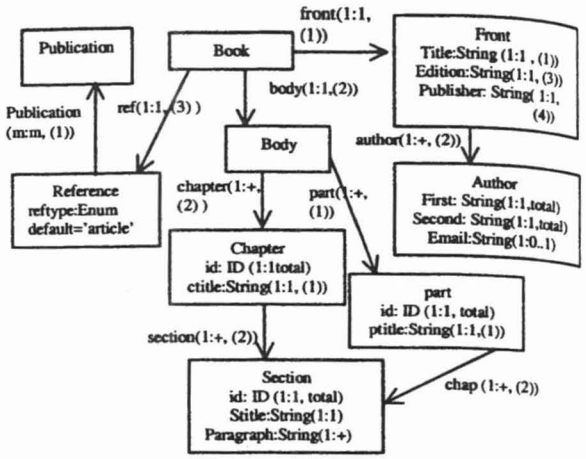

Figure 6. The Semantic Schema for the Running Example

3.3.4. The Mapping Meta-Schemas In the previous sections, we have discussed the rules for mapping DTDs to Semantic Schemas. During the mapping process, mapping information must be kept somewhere to facilitate XML document loading and querying in the subsequent steps. Because Sem-ODM is a powerful expressive data model capable of capturing advanced complex modeling constructs, we used Sem-ODB as the storage medium of the knowledge. We have successfully deployed a similar structure (called KnowledgeBase) in SemAccess [14] to help resolve the schema heterogeneity between relational and semantic databases. We applied this structure in XMLWrapper to efficiently resolve the mapping between DTDs and Semantic Schemas. The meta-schema that maps a DTD Construct to a Semantic Schema Construct is illustrated in Figure 7.

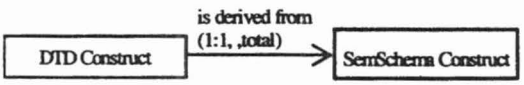

Figure 7. Mapping Information between Sem-ODM Meta-Schema and DTD

4. CONCLUSIONS AND FUTURE WORK

This paper proposes a systematic approach to store and retrieve XML documents using Sem-ODB, a database system with a powerful and expressive data model. By using Sem-ODB, we can avoid the common problem of fragmentation found when using a RDBMS to store XML. The system architecture of the XMLWrapper was presented and the mapping module of it was further discussed in detail. Storing mapping information between DTDs and Semantic Schemas in a Sem-ODB helps resolve schema heterogeneity and allows transformations from one to the other more to be made efficiently. This

technique has been proven in our previous development of SemAccess.

There are some issues that have not been addressed in this paper, such as mapping NMTOKEN, NMTOKENS, and ENTITY of XML documents, as well as the XML query translation and optimization and the XML document generator modules depicted in the system architecture. Our future work will focus on these issues.

5. REFERENCES

[1] Extensible Markup Language (XML) 1.0 (Second Edition) W3C Recommendation, October 2000, http://www.w3.org/TR/2000/REC-xml-20001006.

[2] J. Bosak, T. Bray, et. al., "W3C XML Specification DTD," http://www.w3.org/XML/1998/06/xmlspec-report.html

[3] Jennifer Widom, "Data Management for XML: Research Directions", IEEE Data Engineering Bulletin 22(3): 44-52(1999)

[4] Alon Levy, More on Data Management for XML,http://www.cs.washington.edu/homes/alon/widom-response.html

[5] Gerti Kappel, Elisabeth Kapsammer, S. Rausch-Schott, Werner Retschitzegger, "X-Ray - Towards Integrating XML and Relational Database Systems", 19th International Conference on Conceptual Modeling, Salt Lake City, Utah, USA, October, 2000.

[6] J. Shanmugasundaram, et al., "Relational Databases for Querying XML Documents: Limitations and Opportunities", Proceedings of the 25th Int. Conf. On Very Large Data Bases (VLDB), Edinburgh, Scotland, UK. 1999

[7] M. Klettke, H. Meyer, "XML and Object-Relational Database Systems – Enhancing Structural Mappings Based on Statistics", Proceedings of the Third International Workshop on the Web and Databases, WebDB 2000, Dallas, Texas, USA, May, 2000.

[8] D. Florescu, D. Kossmann, "Storing and Querying XML Data Using an RDBMS", IEEE Data Engineering Bulletin, Special Issue on XML, Vol. 22, No. 3, September, 1999

[9] V. Christophides, S. Abiteboul, S. Cluet, M. Scholl, "From Structured Documents to Novel Query Facilities", Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data, Minneapolis, Minnesota, May, 1994

[10] R. Goldman, J. McHugh, and J. Widom, "From Semistructured Data to XML: Migrating the Lore Data Model and Query Language", Proceedings of the 2nd International Workshop on the Web and Databases (WebDB '99), Philadelphia, Pennsylvania, June 1999

[11] Tamino XML Database Home Page, http://www.softwareag.com/tamino/

[12] Rishe N., Sun W., Barton D., Deng Y., Orji C., Alexopoulos M., Loureiro L., Ordonez C., Sanchez M., Shaposhnikov A., "Florida International University High Performance Database Research Center". In *SIGMOD Record*, 24 (1995), 3, pp. 71-76.

[13] Naphtali Rishe, Database Design: The Semantic Modeling Approach, McGraw-Hill, 1992.

[14] Naphtali Rishe, Jun Yuan, Rukshan Athauda, et al., "SemanticAccess: Semantic Interface for Querying Databases", Proceeding of the VLDB conf. , Cairo, Egypt, 2000.

[15] Shu-Ching Chen, Xinran Wang, Naphtali Rishe, and Mark Allen Weiss, "A high-Performance Web-Based System Design for Spatial Data Access", Eighth Symposium of ACM GIS, Washington D.C., USA.

[16] A. Deutsch, M. Fern'ndez, and D. Suciu, "Storing Semistructured Data with STORED", Proceedings of the ACM SIGMOD International Conference on Management of Data, Philadelphia, Pennsylvania, USA June, 1999.

ISBN: 980-07-7541-2