Running head: Storage and Access Control Issues for XML Documents

# Storage and Access Control Issues for XML Documents

George Pallis
Department of Informatics
Aristotle University of Thessaloniki
Greece

Konstantina Stoupa
Department of Informatics
Aristotle University of Thessaloniki
Greece

Athena Vakali
Department of Informatics
Aristotle University of Thessaloniki
Greece

## Abstract

XML documents management is becoming an area of great research value and interest, since XML has become a popular standard for data communication and knowledge exchange over the Internet. Therefore, new issues have emerged in terms of storage and access control policies for XML documents. Concerning the storage issues, the majority of proposals rely on the usage of typical database management systems (DBMSs) whereas, XML documents can also be stored in other storage environments (such as file systems and LDAP directories). It is important to consider storage and access control together since these issues are essential in implementations for XML documents management. Moreover, the chapter focuses on the recent access control models which guarantee the security of the XML-based data which are located in a variety of storage topologies. This chapter's goal is to survey and classify existing approaches for XML documents storage and access control and at the same time, highlight the main differences between them. The most popular XML database software tools are outlined, in terms of their storage and access control policies.

# Storage and Access Control Issues for XML Documents

## Introduction

Internet is currently the core media for data and knowledge exchange. XML (eXtensible Markup Language)[1], a subset of SGML (Standard Generalized Markup Language), is introduced by the World Wide Web Consortium (W3C) to complement and enhance HTML (Hypertext Markup Language) in electronic data representation and exchange on the Web. XML is becoming wide spreaded and is a text-based markup language (like HTML) but it supports a richer set of features. The main advantage of using XML is that an XML document (differently from an HTML document) can be written once and visualized in a variety of ways. Therefore, XML is currently the most popular standardization effort in Web documents representation and is rapidly becoming a standard for data representation and exchange over the Internet. As a result, large amounts of XML documents are being generated and their efficient management has become a major necessity. Researchers in both industry and academia have focused on efficiently storing, manipulating, and retrieving XML documents.

The main XML related research issues refer to the XML data accessing, storing, querying and exchanging. Indeed, even if XML lends its power to its ease-of use and extensibility, it is this structure of XML that results in a controversial fact. From one point of view, this structure characterizes XML as an ideal building block on high-speed applications, whereas from another point of view, it is this structure that makes XML unsuitable for usage under pre existing data management environments. Most implementations rely on the usage of typical database management systems (DBMSs), whereas others are based on specific systems (providing ad-hoc functionalities). Moreover, since XML can be used over various application platforms, different management approaches have to be devised depending on the type of the considered XML documents (structured vs unstructured), the platform type (DBMS vs file-based systems), and their main usage. Whatever is the chosen solution, a crucial point in efficiently managing XML documents is devising efficient storage and accessing control techniques. Among data management issues, storage and securing techniques have a particular importance, since the performance of the overall XML-based Web information system relies heavily on them.

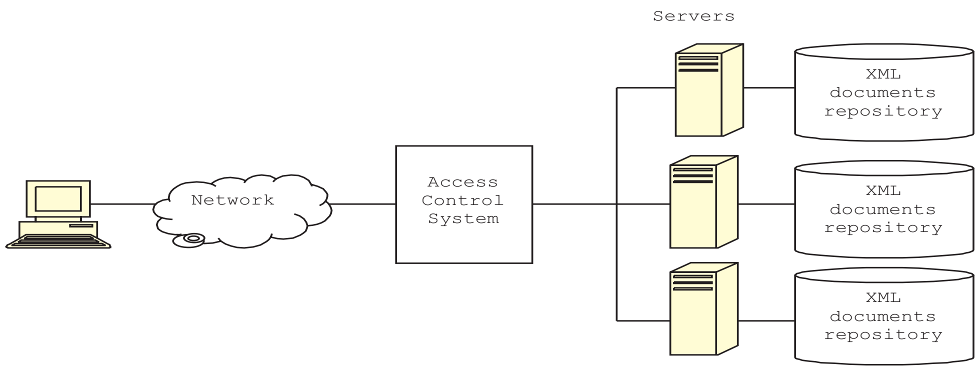
Several solutions for storing XML have been proposed both in the scientific literature and in commercial products, such as flat files, relational database management systems, object oriented database management systems, native XML database systems and LDAP directories. In this framework, the majority of storage and retrieval approaches are based on the usage of existing DBMSs or on specialized system implementations. Furthermore, these approaches can be classified

[1] http://www.w3.org/TR/1998/REC-xml-19980210/

with respect to the type of the system on which they rely and on the used XML document representation model.

Access control is essential in guaranteeing the security of such storage approaches. Several types of access control models have been introduced so far, ranging from the conventional ones (appropriate for centralized systems with low workload) to the most recent and flexible ones (such as the role-based). The implementation of models controlling storage and access to such documents has become a major research issue since hypertext documents are the most common form of information exchanged through the Internet.

**Figure 1.** Architecture of the considered topology



This chapter presents a survey for XML documents storage and access control issues and aims at contributing in identifying the most important policies for storage and accessing in Web-based information systems (which use the XML as their data representation format). Therefore, the goal of this paper is to survey and classify such approaches and at the same time, highlight the main differences between them. Moreover, the aim of this chapter is to provide a survey of the currently proposed storage and accessing approaches for XML documents, categorized with respect to qualitative parameters and focusing on the applicability and the efficiency of their structures. The architecture of the system discussed in this chapter is depicted in Figure 1 where a group of servers (supporting XML document repositories) are protected against unauthorized access (through an access control mechanism). More specifically, XML is used both for XML documents storage in repositories and XML structured access control.

Moreover, the chapter discusses about the storage and access control policies applied in the most popular XML based software tools, which primarily consider assuring authorized access and protecting documents (located in web-accessible databases). Finally, we will focus on the recent role-based control models, since most XML-based storage systems use roles to manage the requested accesses to the protected resources.

This chapter covers the storage and access control issues concerning XML documents. The whole discussion is based on a common case study. The whole chapter is organized into two parts: in the first part  XML documents storage issues are overviewed and in the second one a description of the main functions of XML-based access control and authorization models is given and emphasized by he use of various examples. Moreover, the most well-known software tools for XML documents storage and access control are presented and their approach for storage and access control is highlighted and identified. Finally, an outline of the current research trends concludes the chapter.

## XML Data Representation

In Each XML document may be based on a structural description of its content which is specified either by Document Type Definitions (DTDs) or XML Schemas. More specifically, the DTD defines the document structure with a list of legal elements (which describe the rules for associating tags with their content). The main purpose of the DTD is to provide a definition of the proper structure of an XML document. A DTD can be declared as an embedded object in the XML document, or as an external reference. An alternative to DTDs is the definition of scheme by a more sophisticated language, the *XML Schema* (which is more extensible and flexible than conventional DTDs). More specifically, the XML schema provides a means for defining the structure, content and semantics of XML documents. On the other hand, the DTD (or the XML schema) structure contributes in facilitating (for one application) the use of an XML document (created by any application) and improves the data communication over the Internet. Unfortunately, the syntax of XML Schemas has not yet been standardized and currently, the W3C organization works on version 1.1 of XML Schemas.

For reasons of uniformity we will use a case study which will be extended and referenced in the following sections. We refer to a library containing book catalogs in digital form. These catalogs include books whose authors are authorized to modify them. Subscribers to this digital library are able to read such catalogs. An example of such an XML document, which describes a fragment of a book catalog, is depicted in Figure 2. In addition, Figure 3 shows the XML Schema that conforms with the previous XML document while Figure 4 presents the DTD of the XML document. As shown in Figure 3, XML schema is a superset of DTDs and it is itself specified by XML syntax. In particular, the benefits that an XML Schema offers over DTDs can be summarized as follows:

- User-defined types are created.
- The text that appears in elements are constrained to specific types (such as numeric types in specific format).
- Types are restricted in order to create specialized types (e.g. specifying minimum and maximum values).
- Complex types are extended by using a form of inheritance.

However, the cost that is paid for these features is that XML Schema is significantly more complicated than DTDs.

**Figure 2.** An Example of an XML document for a book catalog

```
<?xml version="1.0"?> <!DOCTYPE BOOKS SYSTEM "books.dtd">
<catalog>
   <book bookID="bk101">
   <authors>
      <person perID="P101">
          <fname>Angappa</fname>
          <lname>Gunasekaran</lname>
      </person>
      <person perID="P102">
          <fname>Omar</fname>
          <lname>Khalil</lname>
      </person>
      <person perID="P103">
          <fname>Syed Mahbubur </fname>
          <lname>Rahman</lname>
      </person>
  </authors>
  <title>Knowledge and Information Technology Management</title>
  <category>Computer</category>
  <price>84.95</price>
  <publish_year>2003</publish_year>
  <publisher>Idea Group Publishing</publisher>
  </book>
  <book bookID="bk102">
      ...
  </book>
</catalog>
```

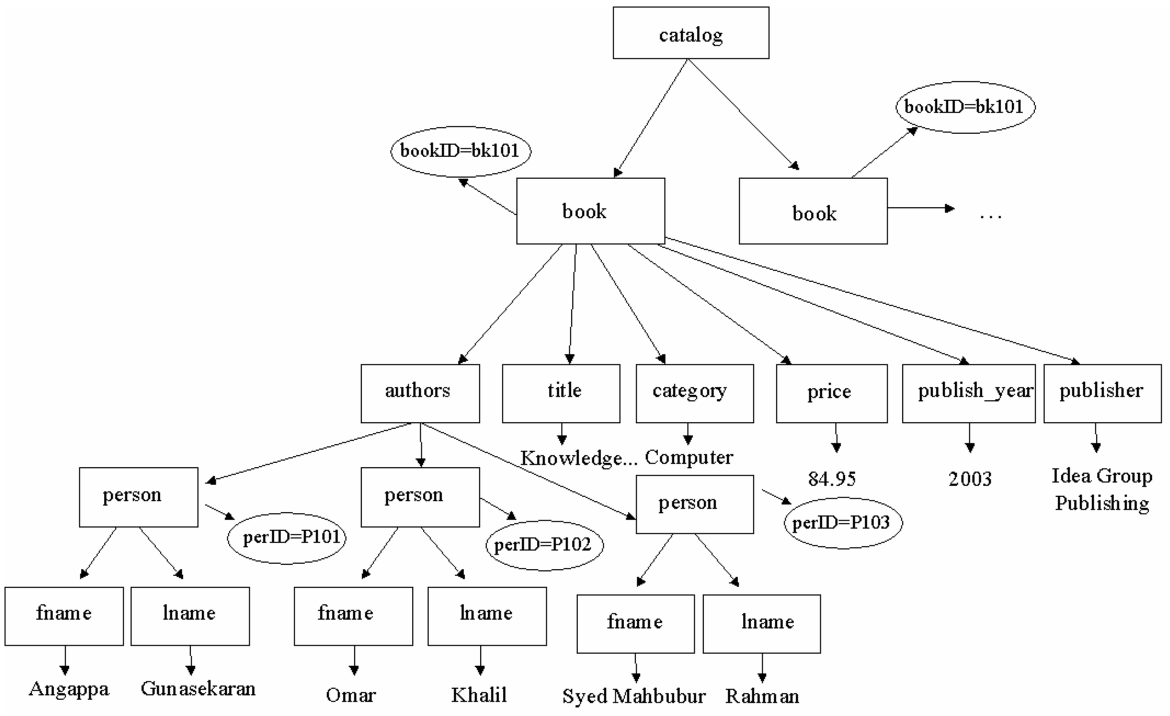
**Figure 3**: The XML Schema Definition for the document in Figure 2

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="catalog">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="book" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="authors" maxOccurs="unbounded">
                <xsd:compexType>
                  <xsd:sequence>
                    <xsd:element name="person">
                      <xsd:complexType>
                        <xsd:sequence>
                          <xsd:element name="fname" type="xsd:string"/>
                          <xsd:element name="lname" type="xsd:string"/>
                        </xsd:sequence>
                        <xsd:attribute name="perID" type="ID" use"required"/>
                      </xsd:complexType>
                    </xsd:element>
                    <xsd:element name="title" type="xsd:string"/>
                    <xsd:element name="category" type="xsd:string"/>
                    <xsd:element name="price" type="xsd:string"/>
                    <xsd:element name="publish_year" type="xsd:string"/>
                    <xsd:element name="publisher" type="xsd:string"/>
                  </xsd:sequence>
                </xsd:complexType>
              </xsd:element>
            </xsd:sequence>
            <xsd:attribute name="bookID" type="ID" use="required"/>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

**Figure 4**: The DTD for the Book Catalog Example: books.dtd

```
<!ELEMENT catalog(Book)>
<!ELEMENT book (authors,title,category,price,publish_year,publisher)>
<!ATTLIST book bookID ID default >
<!ELEMENT authors (person+)>
<!ELEMENT person(fname,lname)>
<!ATTLIST person perID ID default >
<!ELEMENT fname (#PCDATA)>
<!ELEMENT lname(#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT category (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT publish_year (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
```

**Figure 5**: A Tree-like Structure of the XML Document



The most common abstract data representation model for XML documents is a tree-like structure. Here, we will use a simplified tree form, where the nodes represent only elements, attributes and data. The tree for the document presented in Figure 2 is depicted in Figure 5, where the rectangles represent the elements of the XML document, the ellipses the attributes, and the edges the relationship(s) between an element and its sub-elements (or its attributes).

There are three typical processing steps in manipulating XML documents. Firstly, the XML document is parsed by using an XML parser. Secondly, the document is processed (this step depends on the chosen XML parser) and finally, the data are interpreted and a report is produced.

In order to process the XML documents effectively, two application program interfaces (APIs) have been proposed: SAX (Simple API for XML) and DOM (Document Object Model). The first is based on the textual processing of XML documents. The second is based on the tree representation of XML documents. These APIs are the de-facto standards for processing XML documents.

- **SAX**[2]: An event-based API for XML. SAX is based on a parser where the users provide event handlers for parsing various events. More specifically, the SAX parser sends events (as it parses the whole document), supporting a (LIFO) stack-oriented access for handling these events. SAX processes the XML documents as a stream. Experiments have shown that SAX is suitable in two cases: when the system's memory is limited or when only parts of documents are required.
- **DOM**[3]: The DOM API (developed by W3C) follows a tree-like structure and XML documents are parsed into a tree representation. More specifically, the elements have parent-child relations with other elements. The parser builds an internal structure such that an application can navigate it (in a tree-like fashion). DOM allows an application to have random access to the tree-structured document (at the cost of increased memory usage). In this context, a variety of functions for traversing the DOM tree have appeared. Comparing with SAX, DOM is suitable when processing XML documents for multiple times whereas its disadvantage is that loading is needed and parsing in every step.

## XML Documents Storage Policies

XML data storage policies are related with locating XML documents in an effective manner, on persistent memory. Several approaches have been developed for high performance storage and retrieval of XML documents. Here, we categorize placement approaches with respect to the corresponding underlying storage environments. In particular, a typical approach is to store the XML documents in a relational DBMS, (flat files are converted to relational representation) inheriting both the benefits and drawbacks of the relational databases. In a second approach XML documents are stored in non-relational databases, whereas a number of Object-Oriented, Object Relational and Native DBMSs have been proposed. Finally, in a third approach, XML documents can be stored in other XML storage environments such as file systems and LDAP directories.

### XML Storage under Relational DBMSs

A relational DBMS uses a collection of tables to represent both data and relationships among these data. More specifically, in order to represent XML data by using tables, it is necessary to break down the XML documents into rows and columns. The tree-like structure of XML facilitates both

[2] http://www.saxproject.org/

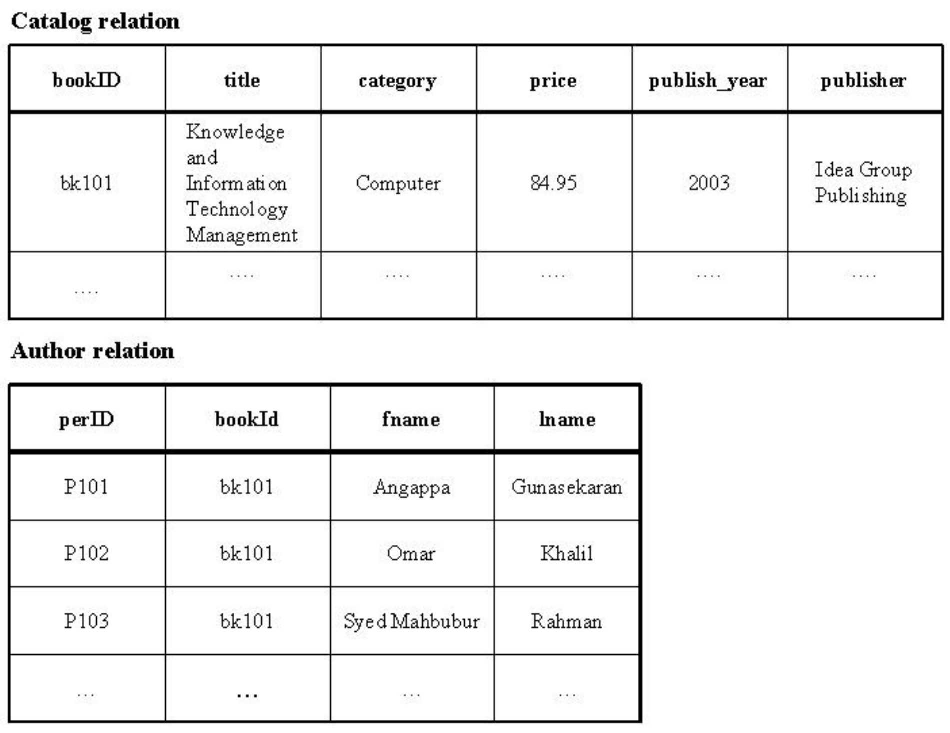
[3] http://www.w3.org/DOM/

their decomposition and storing in relational tables. However, this process is expected to cause some performance overhead mainly due to the continuous translation of trees to tables (and vice versa). In this context, an XML document (depicted in Figure 2), can be represented easily by relation tables, (as illustrated in Figure 6).

**Table 1**: XML DTD and Relational Database Schema Relationship

| **XML DTD** | **RELATIONAL DATABASE SCHEMA** |
|---|---|
| Element | Table |
| ID Attribute | Primary key |
| #REQUIRED | NULL |
| #IMPLIED | NOT NULL |

**Figure 6**: Relations for XML Data Representation



**Catalog relation**

| bookID | title | category | price | publish_year | publisher |
|---|---|---|---|---|---|
| bk101 | Knowledge and Information Technology Management | Computer | 84.95 | 2003 | Idea Group Publishing |
| .... | .... | .... | .... | .... | .... |

**Author relation**

| perID | bookId | fname | lname |
|---|---|---|---|
| P101 | bk101 | Angappa | Gunasekaran |
| P102 | bk101 | Omar | Khalil |
| P103 | bk101 | Syed Mahbubur | Rahman |
| ... | ... | ... | ... |

Due to its popularity, several models have been proposed to store XML documents in Relational DBMSs (e.g. (Shimura et. al., 1999; Silberschatz et. al., 2002; Tian et. al., 2000; Zhu and Lu, 2001)). In this framework, mapping relation is one of the most popular ways of storing XML documents in Relational databases. Existing XML to relational mapping techniques can be classified into the following two categories (Amer-Yahia and Fernandez, 2002):

- **Schema-driven Techniques**: These techniques require the existence of a DTD or XML schema (Florescu and Kossmann, 1999; Khan and Rao, 2001). In particular, XML elements (whose DTD or XML Schema is known) are mapped effectively to relations and attributes. This is done by using either a fixed or a flexible set of rules. More specifically, fixed mappings (using basic, shared and hybrid inlining techniques) are defined from DTDs to tables. When converting an XML DTD to relations, it is tempting to map each element in the DTD to a relation and map the attributes of the element to attributes of the relation. Table 1 summarizes the simulating between an XML DTD and a relational database schema. On the other hand, flexible mappings can be supported by using an XML Schema. In this case, more information might be captured than the previous one (in which a DTD is given). In general, XML schema has specific featured which are useful for storage. One can specify precise data types (e.g. strings, integers) instead of just text. For example, in the XML schema, the type information should be associated to attributes. The mapping approach described by (Bohannon et. al., 2002) is based on the principle of mapping groups in XML schema into tables in the relational schema. However, there are some common principles which are applied in both mappings (fixed and flexible). Therefore, sub-elements that can occur multiple times are mapped into separate tables. Non-repeated sub-elements may become attributes. The optionality is another principle which is handled using nullable fields. Finally, the choice is represented by using multiple tables or a universable table with nullable fields. Experiments have shown that the schema-driven techniques provide efficient QoS (Quality of Service). Furthermore, XML queries can be translated into SQL and executed efficiently. However, this approach cannot store all the XML documents. In particular, XML documents whose DTD (or XML-Schema) is unknown cannot be mapped to relations. In such a case, these documents can be stored as a tree representation or as strings. Another approach proposed by (Yoshikawa et. al., 2001) (called *XRel*) is based on such a mapping by which a fixed database schema is used to store the structure of all XML documents (based on path expressions).

- **User-defined Techniques**: These techniques have been proposed by commercial databases (such as IBM DB2, Oracle 8i and 9i, Microsoft SQL Server 2000). In this case, the user gives the underlying relational schema. Therefore, the mapping is provided either by using a declarative interface or programmatically through special purpose queries.

Moreover, a technique for automatic mapping of XML documents to relations under a relational DBMS is presented by (Khan and Rao, 2001), whereas a study on how XML data can be stored and queried under a standard relational database system is presented by (Florescu and Kossmann, 1999) . Furthermore, a data model and an execution process for efficient storage and retrieval of XML documents under a relational DBMS is presented by (Schmidt et. al., 2000). Overall, all existing storage methodologies in Relational DBMSs are categorized in a more general perspective, all the above models can be categorized in:

1. **XML Data as a Tree**: the XML documents can be easily represented as a tree, and node types in the tree are: element, attribute, and text (Kanne and Moerkotte, 2000). Therefore, XML data can be stored by using a pair of relations: *nodes* and *child*. In particular each element and attribute in the XML data is given a unique identifier (primary key). This representation has the advantage that all XML information can be represented directly in relational form, and many XML queries

can be translated into relational queries and executed inside the database system. The key issue for this approach is the mapping from the tree structure of an XML document to tuples in relational tables. In this context, several approaches have been proposed which model the XML documents as trees and store them by using various relations (Florescu and Kossmann, 1999; Kanne and Moerkotte,2000; Khan and Rao, 2001; Silberschatz et. al., 2002). More specifically, the existing relations are the following:

- **Nodes and edges**: XML documents, in this relation, can be stored using the following pair: nodes(id, type, label, value) and edges(parent-id, child-id, order). More specifically, each element/attribute is given a unique identifier. The type is either "element" or "attribute" and the label specifies the tag name of the element or the name of the attribute respectively. The value is the text value of the element. On the other hand, the relation "edges" indicates the parent-child relationships in the tree. The attribute order (which is optional) records the ordering of children. The main drawback of this relation is that each element gets broken up into many pieces, and a large number of joins are required to reassemble elements.
- **Nodes and values**: In this case, the pair of relations is the following: nodes(tag, docId, startPos, endPos, level) and values(word, docId, position). This relationship is based on partitions. The position indicates the word displacement within the XML document. The interval [startPos, endPos] determines the relationship between parent-child and ancestor-descendant. The drawback of this approach is that special types of joins are needed to evaluate path queries. Even simple queries require a large number of joins.
- **Nodes and paths**: XML documents can also be stored by using the following pair of relations: nodes(docId, pathId, tag, order, position, type, value) and paths(pathId, path). In this relation, each node has a unique path from the root of the document. For supporting this kind of relation, it is required to have implemented index structures on path expressions. Finally, the positions are used to find sub-trees.

Therefore, the tree structure is decomposed into relations (we can easily access and reuse) by the unit of logical structure and index structures can be used (such as B+ trees, R trees etc). These index structures are also provided in relational database systems in order to support the tree data model. The decomposition of XML documents is executed when they are parsed by using an application program interface (DOM or SAX). However, it has the disadvantage that each element is broken up into many pieces, and a large number of joins are required to reassemble elements.

2. **XML Data as a String**: A common way to store XML data in a relational DBMS is to store each child element as a string, in a separate tuple in the database (Kanne and Moerkotte, 2000). For

example, the XML document (Figure 2) can be stored as a set of tuples in a relation *elements(data)*, with the attribute *data* of each tuple storing one XML element (e.g. Title, Category, Price) in string form. An advantage of this approach is that as long as there are several top-level elements in a document, strings are small compared to full document, allowing faster access to individual elements. Furthermore, XML documents can be stored without DTD (or XML schema). A disadvantage of this approach is that we cannot query the data directly (since the database system does not have knowledge about the stored elements schema). A solution to this problem is to store various types of elements in different relations, and store the values of some critical elements as attributes of the relations to enable indexing. It is important to indicate that indexing techniques play a crucial role in improving the performance of storage systems. In fact, several database systems (such as Oracle 8i, 9i), support function indices, which can help avoid replication of attributes between the XML string and relation attributes.

## XML Storage under Object-Relational DBMSs

As Web applications manipulate an increasing amount of XML, there is a growing interest in storing XML documents in Object-Relational (O-R) DBMSs. In particular, several Relational DBMS vendors (such as Microsoft (MS) SQL Server 2000, Oracle 8i and 9i, IBM DB2, Informix etc) include Object-Oriented (O-O) features in their products, in order to offer more powerful modeling capabilities for storing XML documents. These products are discussed in detail in the next Sections.

In general, the XML documents, in O-R databases, are stored in a nested table, in which each tag name in DTD (or XML schema) corresponds to an attribute name in the nested table. In O-R DBMSs, the procedure for storing XML data to relation mapping is modeled by an O-R model. More specifically, each nested XML element is mapped into an object reference of the appropriate type. Then, several mapping rules are indirectly embedded in the underlying model. For the construction of XML document, the DBMS translates all the object references into an hierarchical structure of XML elements. For example, in an XML document (Figure 1), the elements *authors* and *person* are nested, and the latter one is the child element of the former one. Therefore, two objects are defined, namely *authors-obj* and *person-obj*, and the second one will make an object reference to the first one.

For mapping of an XML document into an O-R DBMS, it is required to traverse the XML document. For this reason, an XML DOM is usually used, to facilitate the construction of a tree structure in the main memory. In particular, this tree structure will contain the document's elements, attributes, text etc. It is important to recall that a DOM-based parser exposes the data along with a programming library -called the DOM Application Programming Interface (API)- which will allow data in an XML document to be accessed and manipulated. This API is available for many different programming languages (Java, C++ etc).

## XML Storage under Object-Oriented DBMS

Another option is to use O-O databases for storing the XML documents. In particular, XML documents are stored as collections of object instances, using relationships based on the O-O idea (Vakali and Terzi, 2000). Since O-O DBMSs have been designed to work well with object programming languages (such as C++, C# and Java). Inheritance, and object-identity are their basic characteristics. In particular, O-O DBMSs tend to store XML in a way approximate to the DOM, (which has already been presented). However, O-O DBMSs cannot easily handle data with a dynamic structure since a new class definition for a new XML document is needed and the use of O-O DBMSs for XML document storage is not as efficient and flexible.

For such reasons, the use of O-O DBMSs has shown very limited commercial success, (especially when compared to their relational counterparts). The most indicative O-O are:

- **Lore** (McHugh et. al., 1997): It is one such example that has been built to manage XML documents. The data model used for semi-structured data representation in Lore is the *Object Exchange Model* (OEM). This model can be thought of as a labeled directed graph. The vertices in the graph are *objects* and each object has a unique object identifier. This model is flexible enough to encompass all types of information, including semantic information about objects.
- **Extensible Information Server (XIS)**[4]: An O-O system which stores XML documents under eXcelon's ObjectStore O-O database, as DOM trees, stored in a proprietary, B-tree-like structure (for performance reasons) and can be indexed by using both value and structural indexes. In particular, XIS stores XML documents in a preparsed format in order to reduce the overhead associated with parsing on demand. Furthermore, XIS supports queries through XPath with extension functions and a proprietary update language. It also supports server-side functions, (written in Java) and can directly manipulate data in the database (through a server-side DOM implementation). Moreover, XIS provides a distributed caching mechanism for improving concurrent access and overall application performance.
- **SHORE (Semantic Hypertext Object REpository)** (Hess et. al., 2000): This system stores information extracted from XML documents whereas, an object-based XML data representation model for effective XML data placement. In particular, it stores information extracted from XML documents using a variant of R-trees and B-trees structure.

[4] http://www.exceloncorp.com/xis/

## XML Storage under Native XML DBMSs

Most recent advances in XML technology, have presented another approach - created specifically for XML - known as the "Native" XML database. Native XML databases satisfy the need for a more robust XML approach by offering a solution that is specifically designed to handle and optimize XML's unique structure. Using a relational database management system to store XML documents can create serious performance problems for large-scale applications, since data hierarchy, context and semantics are often lost (when XML documents are retrieved and processed with SQL). As an alternative, storing and indexing XML documents in their native form preserves the document structure, content and meaning and increase the performance of the underlying applications.

In this context, native XML DBMSs use XML as their basic data model. More specifically, a native XML database defines a (logical) model for an XML document and stores and retrieves documents according to that model. In order to store the XML documents on a native XML database, two basic steps are involved:

1. Describe the data via its structure (DTD or XML schema) and
2. Define a native database XML schema (or a data map) to use for storage and retrieval.

In this case, the XML document is the fundamental unit of (logical) storage, such as a relational database has a row in a table as its fundamental unit of (logical) storage. Therefore, it is not required to support any particular underlying physical storage model in native XML databases. For example, it can be built on a relational, hierarchical, or O-O database, or use a proprietary storage format such as indexed, compressed file. XML schemas are implemented in native XML databases to record rules for storing and indexing data and to provide data retrieval and storage information to the underlying database engines.

Furthermore, native XML databases can be categorized into two policies: text-based storage and model-based storage. The first one stores the entire document in text form, (such as a binary large object-BLOB in a relational database). The second one stores a binary model of the document in an existing or custom database. They are particularly suited to store, retrieve, and update XML documents. Typical examples include Natix (Kanne and Moerkotte, 2000), Tamino (Schoning, 2001), SODA, Ipedo, Xyleme etc. In fact, native XML DBMSs satisfy the need for a more robust XML storage approach by offering a solution that is specifically designed to handle and optimize XML's unique structure.

Several commercial native XML DBMSs have also been developed, but until now they have not become very popular. The main reason is that these systems (including physical distribution of data, the index mechanism, etc.) must be built from scratch. In general, native DBMSs differentiate based on their purpose (research or commercial implementations) and their storage structure is employed accordingly. More specifically, the research-oriented implementations support trees and/or sub-trees for the structural unit whereas the commercial-oriented tools use collections (like directories) as their main structural unit. The most indicative of them are given in Appendix A.

## Other Environments for XML Storage

Alternatively, XML documents can also be stored in other storage environments such as file systems and LDAP directories.

- **File System Storage**: Since an XML document is a file, a typical storage approach is to store it simply as a flat file. In particular, this approach uses a typical file-processing system, supported by a conventional operating system (as a basis for database applications). The wide availability of XML tools for data files results in a relatively easy accessing and querying of XML data (which are stored in files). By using a flat file for XML data, we have a quite fast storing (or retrieving) of whole documents. However, this storage format has many disadvantages, such as difficulty in accessing and updating (since the only way is to over-write the whole file) data (Silberschatz et. al., 2002). Furthermore, this approach encounters also security, concurrent access, atomicity and integrity problems.
- **LDAP Directories**: Currently, researchers have showed a steadily increasing interest in LDAP (Lightweight Directory Access Protocol) directories in order to effectively store XML documents (e.g. (Marrσn and Lausen, 2001)). Several commercial companies offer LDAP support in their browsers and operating systems, making directory services a remarkable alternative to more traditional systems for the storage and efficient retrieval of information. According to this trend, XML documents are stored in LDAP directories which can be considered as a specialized database (Johner et. al., 1998). Therefore, the internal storage model of this database system is defined in terms of LDAP classes and attributes[5].

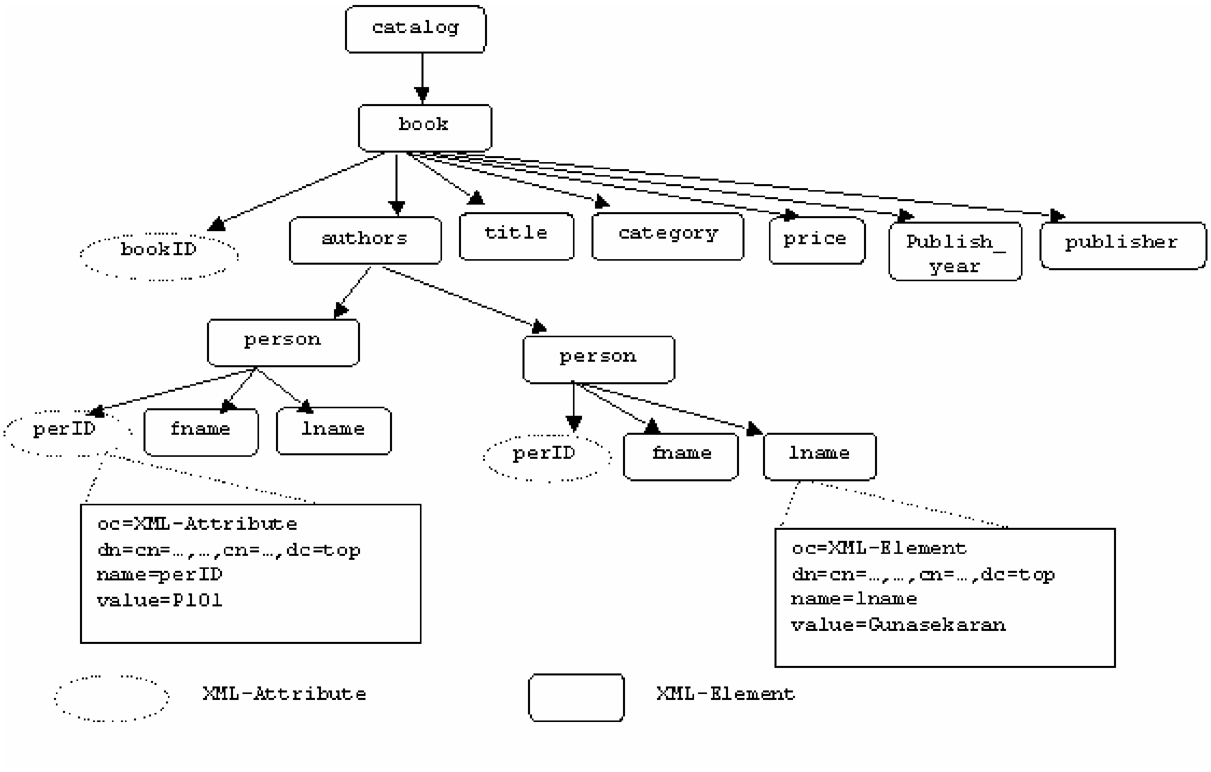
  The XML documents are organized in a hierarchical fashion, similar to the way files are organized in file system. In conjunction with the DSML (Directory Service Markup Language), which is a new standard for representing directory information as XML, the directory services can take advantage of XML's most powerful features. In particular, DSML bridges the gap between directory services and XML applications in a robust way. Today, several products support DSML, including Sun Microsystems, Microsoft, Oracle Corp., IBM, Hewlett-Packard etc.

  The main differences between directories and typical DBMSs are related with size and design issues. In fact, LDAP directories are generally smaller and less complex

[5] More details about the architecture of the LDAP model and protocol are discussed by (Howes et. al., 1999).

applications than DBMSs. More specifically, the LDAP directories are more widely distributed, more easily extended, replicated on a higher scale and have a higher read-to-write ratio than typical DBMSs. On the other hand, the LDAP is a protocol for on line directory services (storing and accessing heterogeneous entities over the Internet). It provides a standard model (the LDAP model) which has an hierarchical infrastructure. On the other hand, XML and LDAP, have many similarities since in LDAP, data are organized as a tree, (where each node can contain data value and can act as a namespace for other nodes). This is quite close to XML, since the XML data model is hierarchical in structure and usually implemented by considering the XML document as a tree structure. Therefore, the transformation from the XML data model to the LDAP data model is not a complex task. Figure 7 depicts the LDAP tree representation for our example XML document. Moreover, LDAP directories also provide several standard APIs that can be used for accessing the directory. Finally, in (Marrσn and Lausen, 2001) an LDAP-based system for storing their XML documents is proposed and the results of their work have shown that LDAP directories reduce the workload and provide efficient performance for storing and retrieving XML data.

**Figure 7**: XML Data Representation in LDAP



## Access Control Issues for XML documents

Even though much research effort has focused on implementing XML-based storage tools, security of such implementations is still in primary research steps. Security of databases is guaranteed through the satisfaction of several needs: (a) *authentication*, (b) *authorization*, (c) *confidentiality*, (d) *integrity*, and (e) *non-repudiation*. Cryptography and digital signatures was the first step towards authentication. Unfortunately, researchers have been engaged in the research of the rest of security issues (which are highly related to access control) only recently. Attacks against confidentiality, integrity and non-repudiation may result from misfunctional access control (or authorization) mechanism which fails to protect document repositories from unauthorized accesses.

The need for a functional access control mechanism has become really significant since XML-based (or hypertext) documents have contributed to the enlargement of repositories. Such a fact has led to the distribution of protected documents to several physical locations which should be secured against unauthorized access. Moreover, since XML is a language for representing documents distributed through Internet, they are stored in huge distributed databases which are connected via the World Wide Web. Although, that development has led to a worldwide exchange of knowledge, it has also increased the need for robust access control. Nowadays, web-based databases can be easily attacked by automated malicious software "traveling" through the Internet (such software owner may hide anywhere on the globe).

It is obvious that XML documents storage systems cannot be protected by conventional access control models, such as *Mandatory Access Control (MAC)* and *Discretionary Access Control (DAC)* since MAC protects only confined centralized environments (when a unique administrator is responsible for the protection of repositories from unauthorized users (Sandhu and Mason, 1993)) and DAC might be a primitive solution (since the owner of each document is responsible for its protection). Since it is unflexible, especially for huge heterogeneous repositories containing (numerous owners) documents.

Most recent authorization systems (and their XML-based storage environments) use mainly the idea of roles employed on user and groups. *User* is the individual connecting to the system, allowed to submit requests. *Group* is a set of users or other groups. *Role* is a named collection of privileges needed to perform specific activities in the system. The important benefit of role-based models is that they are *policy neutral*, i.e. they may enforce multiple policies and they are not associated with a particular security policy.

Role-based access control has been extensively studied by (Osborn et. al., 2000; Sandhu et. al., 1996) where a description of RBAC is given. One or more roles are assigned to users and one or more permissions are assigned to roles.

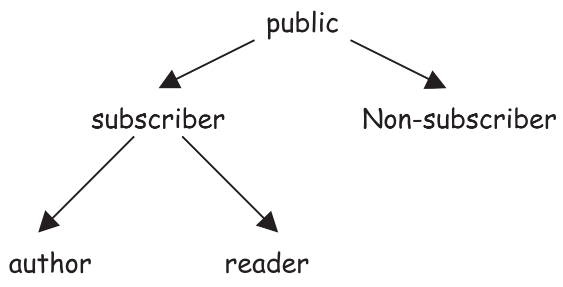
A more modern idea is the concept of credentials which are information concerning a user (Winslett et. al., 1997). This information is provided by the client (subject) when (s)he ubscribes to the system and it is needed by the access control mechanism. Credentials can be stored into different files or organized into groups.

As it will be discussed later, all of the well-known storing environments adopt role based access control which is employed by an access control system as shown in Figure 1. After the system grants access to the requested resource, the request is passed to the storage subsystem for further processing. An access control policy is governed by three issues: (a) *subjects* which are the entities requesting access to protected resources, (b) *objects* which are the protected resources and (c) *authorization* which are rules specifying which subject can access which object. An access control policy consists of several authorizations concerning a subject or an object. Next we will comment on role-based security by extending our case study.

## Authorization Subjects and Objects

A subject is usually identified by its identity or the location from which it sends the request for a document (Damiani et. al., 2001). Moreover, in order to simplify authorization administration and support more compact authorizations, subjects can be organized in hierarchies where the effects of a policy imposed to a general subject propagates to its descendants. While many approaches adopt subjects hierarchies (based on subject identities), some other research efforts support location-based hierarchies. Various XML-based access control models require XML-based presentation of the users' credentials since XML provides all the characteristics for organizing the unique features of subjects.

**Figure 8**: Subject hierarchy



We have already mentioned that most XML-based databases adopt the idea of roles for organizing subjects. With respect to our book catalog, a role hierarchy is the one shown in Figure 8, where the root role refers to all the users who visit that digital library site (public). These users are further classified into subscribers and non-subscribers, while the subscribers are in turn further classified into the authors and the simple readers. Of course, such a functionality demands the subscription of subjects to the system (something that is not required from all applications). XML can be used to define such hierarchies by exploiting the idea of DTDs[6], (as shown in Figure 9) which depicts a DTD describing a category of users. This DTD describes the required data about a

[6] We use DTDs instead of XML Schemas for their brevity

subscriber. The attributes of the core element subscriber is its ID and its *category* which can take two values: *author* and *reader.* A subscriber must have a name and a credit card number, (s)he may have an e-mail address and work for a company and (s)he must have at least one phone number.

Also, we can define a non-subscriber by an empty DTD as:

```
<!DOCTYPE BOOKS SYSTEM "non_subscriber.dtd">
<!ELEMENT   non_subscriber empty>
```

The most common way of identifying XML protection objects is by path expressions which can identify elements and attributes under protection. The exploitation of a standard language like XPath[7] may prove to be highly advantageous because the syntax and the semantics of the language are already known by the potential users. An example of such an expression may be /book//person for the example of Figures 3 and 4.

The objects under protection may be: (a) all instances of a DTD, (b) whole documents or (c) selected portions of a document (like elements or attributes). Referring to Figures 3 and 4 an object may be the whole document or a number of elements like, authors and title. Of course an object may also be the whole DTD such as the one in Figure 4. Authors in (Akker et. al., 2000) introduce the categorization of objects into sets in order to minimize the need of using multiple security policies for protecting them independently.

**Figure 9:** A Subject DTD, XML Schema and an instance

```
<!DOCTYPE BOOKS SYSTEM "subscriber.dtd">
<!ELEMENT subscriber (name,address,phone_number*,email?,company?,credit_card)>
<!ATTLIST subscriber credID ID #REQUIRED
          category (author|reader) #REQUIRED>
<!ELEMENT name (fname,lname)>
<!ELEMENT fname (#PCDATA)>
<!ELEMENT lname (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT phone_number (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT credit_card (#PCDATA)>


<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="subscriber">
   <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="name">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="fname" type="xsd:string"/>
              <xsd:element name="lname" type="xsd:string"/>
            </xsd:sequence>
          </xsd:complexType>
       </xsd:element>
       <xsd:element name="address" type="xsd:string"/>
       <xsd:element name="phone_number" type="xsd:string" minOccurs="1"/>
       <xsd:element name="email" type="xsd:string" minOccurs="0"/>
       <xsd:element name="company" type="xsd:string" minOccurs="0"/>
       <xsd:element name="credit_card" type="xsd:string" minOccurs="0"/>
```

[7] http://www.w3.org/TR/xpath

```
        <xsd:element name="category">
          <xsd:simpleType>
           <xsd:restriction base="xsd:string">
             <xsd:enumeration value="author"/>
             <xsd:enymration value="reader"/>
           </xsd:restriction>
          </xsd:simpleType>
        </xsd:element>
      </xsd:sequence>
      <xsd:attribute name="credID" type="ID" use="required"/>
   </xsd:complexType>
  </xsd:element>
</xsd:schema>

<subscriber="12345">
<subscriber category="author">
   <name>
      <fname>Omar</fname>
      <lname>Khalil</lname>
   </name>
   <phone_number>011111111</phone_number>
   <email>okhalil@csd.auth.gr</email>
   <credit_card>5555555</credit_card>
</subscriber>
```

## Authorizations

The basic issues in relation to the authorizations for XML documents include:

### *Policies*

The policy is the core security issue in access control systems. An authorization policy consists of a list of authorizations defining which subject can take which action (over which object). A principle that should guide every access control system is the distinction between policy and mechanism. Policy is the logic which rules an access control system, while the mechanism implements (or enforces) that logic.

By XML-based access control systems we refer to tools where policies protecting XML-based repositories are XML-defined and organized, and to tools where only the protected resources are organized by using XML features. The first version is more flexible and modern since it exploits the discussed features of XML.

Policies are most commonly divided into: (a) *positive* where permissions are defined, and (b) *negative* where denials are specified. Modern access control tools combine these two categories in favor of flexibility and expressiveness (e.g. Bertino et. al., 2001a; Damiani et. al., 2000; Castano and Ferrari, 2003).

Since in XML-based systems all of the protected resources are organized according to DTDs (and therefore in hierarchical format) another core issue concerning policies is their propagated effect. A great attention should be given to this feature since its inappropriate execution may lead to conflicts. The implicit propagation rule adopted by all the XML-based access control tools is that all DTD-level policies propagate to all DTD instances. Explicit propagation rules are based on the hierarchical organization of objects. Thus, most tools allow (or deny) the propagation of the effects of a policy under some condition. The approach introduced by (Kudo and Hada, 2000) also allows the propagation to the top where propagation takes effect to the parent elements. Of course, the occurrence of conflicts is inevitable and at this point *conflict resolution mechanisms* are triggered.

Since an authorization policy can be employed for the protection of grouped objects, it can also have effect over grouped subjects. This feature is allowed where subjects are organized in hierarchies (as shown in Figure 8). Therefore, a policy referring to the root of a hierarchy propagates to all of its children down to the leaves. For example in Figure 8 if a policy refers to "subscriber", then its effects propagate to both authors and simple readers by default, unless another policy prohibits this propagation.

According to the above discussed issues, a policy may contain authorizations of the form *<subject, object, mode, type, propagation>*. *Mode* specifies the action the subject can exercise on the object and can take various values according to the implementation and the protected resources. For example, the most commonly used values for documents are those defined by (Kudo and Hada, 2000): *read, write, create* and *delete*. In our example we will adopt the following modes: read, write. The parameter *type* may take either the value "+" if the policy is positive or "-" if it is negative. Finally, the parameter *propagation* defines the type of propagation to be implemented. In order to simplify our example we will only adopt two types: *prop* (propagation) and *no-prop* (no propagation).

The following are examples of policies specified according to the above approach:

```
(1) <//subscriber.[category="author"]/name/[fname="Omar" and lname="Khalil"],
                    catalog.xml://person/[fname="Omar" and lname="Khalil"],
                     write, +, prop>

(2) <//subscriber.[category="reader"], catalog.dtd, read, +,prop>
(3) <*,catalog.dtd,read,-,prop>
(4) <*,catalog.dtd,write,-,prop>
```

Tuple (1) specifies an authorization policy granting write access to author "Omar Khalil" on his books. Moreover, the policy propagates to all of the elements of the documents. The policy encoded by tuple (2) grants all readers the right to read all books. The last two tuples deny read and write access respectively to the public (which is depicted by an *).

## *Provisional Actions*

Policies can be further supported with provisional actions. In (Kudo and Hada, 2000) four types of provisional actions have been introduced: (a) *log* where the session should be logged for the request to be satisfied, (b) *encrypt* where the returned resource view should be encrypted, (c) *verify* where the signature of the subject should be verified and (d) combination of the these.

As PUSH dissemination techniques (i.e. documents are distributed to users without prior request) have been gaining ground these years, a modern access control system should include such mechanisms so as to support this modern dissemination technique. Author-X (Bertino et. al., 2001a) is a Java-based XML access control system satisfying such a need by using the idea of cryptography with several keys. The authors in (Castano and Ferrari, 2003) try to express policies used by Author-X.

*Document View*

The presentation of the requested object to the subject is another important activity performed by every access control model. After the object components accessible by the subject have been identified a mechanism *prunes* the document tree so as to contain only those parts (and sometimes links) that the requesting subject is allowed to access. The resulting document is referred to as *document view*. A problem in such an approach is that a view may not be a valid document as it may not follow the DTD associated with the document from which it is derived (Damiani et. al., 2000). A solution to this problem could be a loosened DTD where the definition of the elements may be defined as optional and not as required. Such an approach prevents the subject from realizing whether some parts are missing or simply do not exist in the original document. According to (Bertino et. al., 2001a, 2001b; Castano and Ferrari 2003; Kudo and Hada, 2000) the resulting document may be encrypted if such a security action has been requested by the user.

## Implementations

Some of the most popular database vendors (like *IBM, Microsoft* and *Oracle*) have developed database tools for the storage of XML documents, and several storage techniques have been adopted (in order to maximize functionality and performance). Moreover, for reasons of integrity these indicative database systems are supported by access control mechanisms. Currently, the most widely adopted technology to enforce the security guarantees of the XML-based databases is the Microsoft .NET platform. Microsoft .NET has been recently proposed and, it has been adjusted to support XML Web Services. Its security mechanism is adopted by Oracle 9i, XIS and DB2 which have been designed to cooperate with Microsoft .NET technology.

### The Microsoft .NET platform

Microsoft .NET[8] is a technology for connecting people, systems and resources. The driving force that has led the Microsoft's researchers attempts to this direction was the need to build and use XML Web Services securely.

The increasing complication of some core tasks, (like security, data management, data storing) has dictated their decomposition into a number of more specialized functions. These "simple" functions are executed by XML Web Services which Microsoft .NET technology fights to integrate. These XML Web Services may originate from various sources residing in distributed places all over

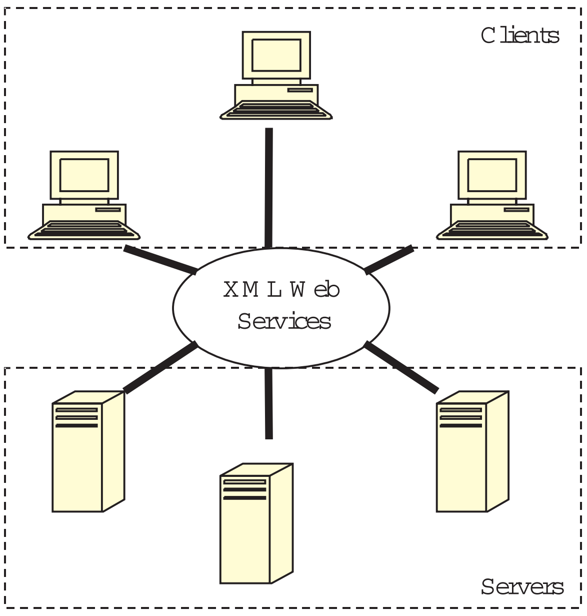
[8] http://www.microsoft.com/net/

the globe. Microsoft provides clients and servers with its own XML Web services but it is possible to combine them with other, through the .NET platform. XML Web Services are characterized by:

- XML Web Services may be differently implemented and they may be placed in various locations but they can cooperate through the use of a common communication protocol, (e.g. SOAP).
- XML Web Services allow the definition of an interface for the communication of the user with them. The description of the steps needed to build interface applications are explained in an XML document called a Web Service Description Language (WSDL) document.
- XML Web Services are registered using Universal Discovery Description and Integration (UDDI) so that users can easily find them.

A general architecture of a .NET-based system showing its XML-based nature is presented in Figure 10.

**Figure 10**: The basic components of a .NET architecture



Such a technology would be totally incomplete if it did not offer guarantees. Therefore, Microsoft .NET contains software for covering authentication, authorization and cryptographic needs. As authentication is beyond the scope of this chapter, we will focus on the second issue. As most modern access control systems, .NET employs the idea of roles as subjects. After a user has logged on the system, (s)he is assigned roles. Authorization policies expressed in .NET framework (i.e. the multilanguage used to build and execute XML and other web services and applications in .NET) define which role can perform which operations over which resources. Developers can sum up XML to express such policies or they can simply tie their system (e.g client or server) with existing Windows authorization mechanisms.

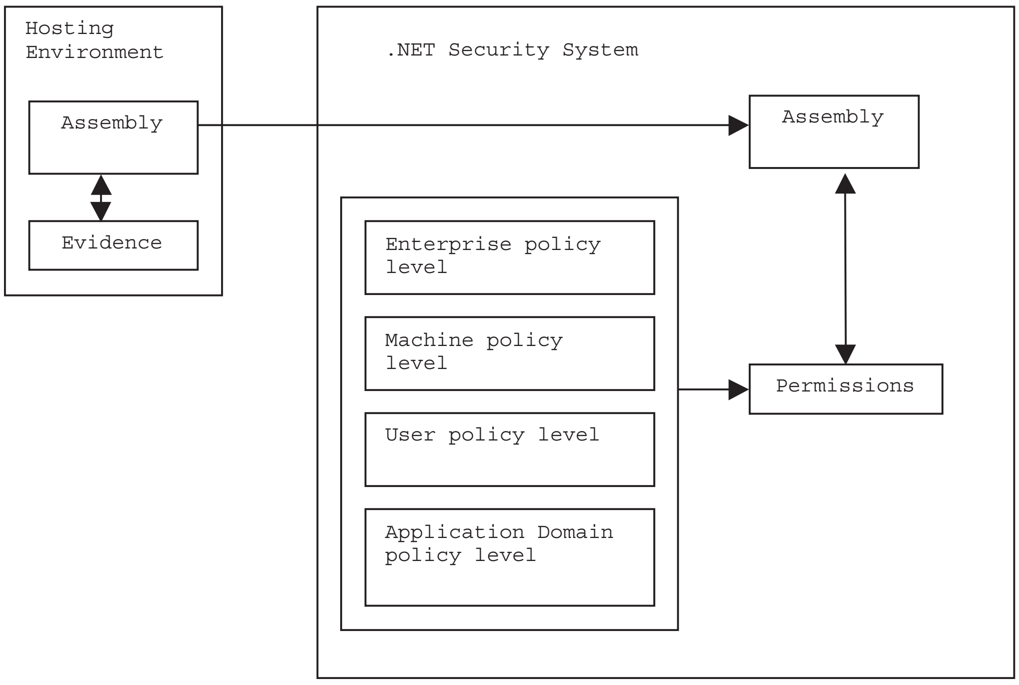
Moreover, the .NET technology employs evidence-based and code access security which allows local administrators to protect resources from malicious code. In such a case the subject is code and is identified using certain features, (like the directory it resides, the Internet location originating from, its has value, etc). Code access security disallows the execution of untrusted code even if the user executing it is trusted. Furthermore, such a functionality allows the developer to define permissions for mobile code which can be downloaded and executed by any user unknown to the developer at the designing phase.

A code access security policy system is governed by three core issues: (a) *evidence*, (b) *permissions* and (c) *security policy*. *Evidence* is the information about the identity of code (called assembly). Evidence is connected with every assembly and it is presented to the security system whenever an assembly is loaded into the runtime. *Permissions* represent authorizations to conduct a protected action (e.g. file or network access). Permissions may be grouped to form *permission sets*. Therefore, whenever an assembly requests access to a protected resource, the security system grants a permission set to it according to its evidence. Finally, a *security policy* integrates the two above issues by defining which permissions are given to an assembly according to its evidence. Policies are categorized into four levels:

- Enterprise policy level
- Machine policy level
- User policy level
- Application Domain policy level

The evidence combined with the policies of each level results to a permission set. The finite permission set arises through the intersection of the previous sets. Thus, in Figure 11 the function of the security system is depicted.

**Figure 11**: How permissions are produced



## XML Storage under DBMSs

Over the last years many of the major database vendors (such as *IBM, Microsoft* and *Oracle*) have developed database tools to support XML documents. The goal of these tools is to provide a secure data transferring from XML documents to databases and vice versa. More specifically, XML database tools involve a set of processes for accessing the XML data and such software can either be built into the object relational DBMSs, or into a new data model (Native XML DBMSs). In this framework, the most well-known software tools that employ XML documents storage and access control can be categorized in the following types of databases:

- **XML-enabled DBMSs:** They provide various interfaces for extracting structured data from XML documents and then to be stored in DBMSs (according to their model). In this case, XML documents are stored in conventional DBMS internally in the database. XML-enabled DBMSs support also services that make the reversible task, producing an XML document. This is done by using a query that can be expressed by the DBMS query language.
- **Native XML DBMSs**: They have been designed especially to store XML documents. XML plays a basic role in Native XML DBMSs since it is the only interface for storing the data. XML documents are stored in XML internally in the database. Therefore, the documents in native XML DBMSs are parsed and stored (as parsed).

**Table 2**: Storage in XML-Enabled Databases

| **PRODUCT** | **DBMS MODEL** | **STORAGE** |
|---|---|---|
| *Oracle 9i* | Object-Relational | Data are stored as relational tables or as XML documents. |
| *MS SQL Server 2000* | Object-Relational | Each XML document is stored as relational table and an element is created for each row. |
| *IBMs DB2* | Object-Relational | Data are stored as relational tables or as XML documents. |

- **Microsoft's (MS) SQL Server 2000**[9]: A relational database management system which provides a variety of tools for managing all aspects of databases. In particular, it provides support for XML data schemas, the ability to execute XPath (a language for addressing parts of an XML document) queries, and the ability to retrieve and create XML data (Rys, 2001). MS SQL Server 2000 has also added many new features which are focusing on XML support. Its utilities offer more flexibility for storing and structuring the XML documents. Each XML document is stored as a relational table and an element is created for each row. Furthermore, the structure of data is transparent to users, who interact with the relational DBMS, using only XML-based technologies. A disadvantage of such an approach is the increased overhead that has been associated with mapping and translating XML structured data into relational data. Another disadvantage is also the hierarchical structure of XML documents.

  SQL Server is the only discussed database system containing its own access control system. As every modern tool authorizations are based on roles. After a user has logged on the system, (s)he is assigned a role which is granted some permissions. SQL Server support several types of roles which are:

  - *Public role*: every user is assigned this role. It is a default role which cannot be deleted. Moreover, the administrator cannot de-assign or assign a subject with this role. Such a role is necessary in case the administrator could not think of role during the design phase that would tie with an unknown user.
  - *Predefined roles*: it is about roles with predefined permissions which cannot be given to other roles. For example, the administrator who has some rights that no other subject should have.
  - *User-defined*: roles of this type are defined by an administrator controlling a single database. Of course, they are local to the database in which they are created and they do not have global effect.

[9] http://www.microsoft.com

  - *Application-defined roles*: these roles are assigned to applications making them able to substitute users and take over control.

  The access control policies, which are determined by the local administrators, define which role is granted which permissions over which protected resources.

- **Oracle9i**[10]: Under Oracle, the XML documents can be stored either in a relational DBMS or in a file system. Therefore, Oracle can store and retrieve entire XML documents as columns, can access XML stored in external files (or on the Web) and can map XML document elements to tables and columns in the database. In particular, its architecture has a specific XML layer that supports a set of tools for managing XML documents. Moreover, the manufacturers of Oracle9i have also developed a high-performance, native XML storage and retrieval technology which is called *Oracle XML DB*. XML DB provides a unique ability to store and manage the XML documents under a standard XML data model. It provides several features for managing XML storage such as XML schema, XML indexes, foldering (enable folders to map XML documents into database structure) etc. Furthermore, XML DB supports also access control policies by creating access control lists (for any XML object), and by defining the users' privileges in addition to the system-defined ones. Details for the Oracle XML data management system can be found in (Banerjee et. al., 2000).

  Oracle 9i can excellently cooperate with Microsoft .NET system and Windows 2000. Therefore, in order to achieve access control one can build .NET clients able to cooperate with Oracle 9i database.

- **IBMs DB2**[11]: IBM DB2 provides a variety of features for storing data. This implementation offers support via a DB2 XML Extender product, which provides new operations for facilitating the storage and manipulation of XML documents. The XML Extender product serves as a repository for the management of DTDs. More specifically, it stores an XML document as a single column or maps the XML document to multiple tables and columns. In order to provide the structure of the generated document, a DTD is mapped against the relational tables using Data Access Definition (DAD). Also, IBM adopts .NET platform for access control.

## Conclusions

This chapter has presented an overview for XML documents storage and access control. More specifically, the most important policies for storage and accessing of XML data and storage is studied under several typical database environments (e.g. Relational, Object-Relational, Object-Oriented etc)

[10] http://technet.oracle.com

[11] http://www-4.ibm.com

and non typical frameworks (such as LDAP). Then, we studied the main issues for the security of XML documents, since access control is essential in guaranteeing the security of such storage approaches. In particular, the most-well known access control and authorization models were presented through examples. Also, the chapter presented the most popular commercial tools for XML management with respect to their storage and access control techniques.

It is important to indicate that no definite guidelines have yet been proposed for selecting an optimal solution when storing and securing XML documents. In order to improve the management of XML documents, some issues should require further research. In particular, the storage of XML documents may be improved by using some data mining techniques  (e.g. specific clustering algorithms). Furthermore, the XML management techniques should further extend existing access control policies, in order to improve the security in XML documents accessing. Finally, the commercial DBMSs should be extended to support more sophisticated storage and access control techniques such as integrated methods for locating and accessing of dynamic XML documents.

## References

Akker T., Snell Q. O. and Clemant M. J., "The YGuard Access Control Model: Set-based Access Control", Proceedings of the 6th ACM Symposium on Access Control Models and Technologies, Chantilly, Virginia, USA, 2001

Amer-Yahia S. and Fernandez M., "Techniques for Storing XML", Proceedings of the 18th International Conference on Data Engineering (ICDE 2002), San Jose, California, USA, Feb.\Mar., 2002.

Banerjee S., Krishnaprasad V., and Murthy R., "Oracle8i-The XML Enabled Data Management System", Proceedings of the International Conference on Data Engineering, 561-568, 2000

Bertino E., Castano S. and Ferrari E., "Securing XML Documents with Author-X", IEEE Internet Computing, pp. 21-31, May/Jun., 2001.

Bertino E., Castano S. and Ferrari E., "On Specifying Security Policies for Web Documents with an XML-based Language", Proceedings of the 6th ACM Symposium on Access Control Models and Technologies, Chantilly, Virginia, USA, pp. 57-65, 2001.

Bohannon P., Freire J., Roy P. and Simeon J., "From XML Schema to Relations: A Cost-based Approach to XML Storage", Proceedings of the 18th International Conference on Data Engineering (ICDE 2002), San Jose, California, USA, May/Jun., 2002.

Castano S., Fugini M., Martella G., and Samarati P., "Database Security", Addison-Wesley, Reading, MA, 1994.

Castano S. and Ferrari E., "Protecting Datasources Over the Web: Policies, Models, and Mechanisms", Web-Powered Databases:Chapter XI, Publisher Idea Group Inc., pp. 299-330, 2003.

Damiani E., Vimercati S. D. C., Paraboshi S., and Samarati P., "Securing XML Documents", Proceedings of the 7th International Conference on Extending Database Technology, Konstanz, Germany, 2000.

Damiani E., De Capitani di Vimercati S., Paraboschi S. and Samarati P., "Desing and Implementation of an Access Control Processor for XML Documents", Proceedings of the 9th World Wide Web Conference (WWW9), Amsterdam, Holland, 2000.

Damiani E., Samarati P., De Capitani di Vimercati S. and Paraboschi S., "Controlling Access to XML Documents", IEEE Internet Computing, pp. 18-28, Nov.\Dec, 2001.

Florescu D. and Kossmann D.,"A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database", INRIA Technical Report (3680), Rocquencourt, France, May, 1999.

Hess A., Schulz H. and Brossler P., "SHORE - A Hypertext Repository based on XML", Technical Report, sd&m Corporation, Software Design and Management, Southfield, USA, 2000.

Howes T. A., Smith M. C. and Good G. S., "Understanding and Deploying LDAP Directory Services", Macmillan Technical Publishing, USA, 1999.

Johner H., Brown L., Hinner F. S., Reis W. and Westman J., "Understanding LDAP", International Technical Support Organization, Ed. IBM, Jun. 1998.

Kanne C. C. and Moerkotte G., "Efficient Storage of XML data", Proceedings of the 16th International Conference on Data Engineering (ICDE 2000), San Diego, California, USA, Feb./Mar., 2000.

Khan L. and Rao Y.,"A Performance Evaluation of Storing XML Data in Relational DBMS", Proceedings of the 3rd ACM CIKM'01 Workshop on Web Information and Data Management (WIDM'01), Atlanta, USA, Nov. 2001.

Kudo M. and Hada S., “XML Document Security based on Provisional Authorization”, Proceedings of the 7th ACM Conference on Computer and Communications Security, Athens, Greece, 2000.

Marrσn P. J. and Lausen G., “On Processing XML in LDAP”, Proceedings of the 27th Conference on Very Large Data Bases (VLDB 2001), pp. 601-610, Roma, Italy, Sep. 2001.

McHugh J., Abiteboul S., Goldman R., Quass D. and Widom J., “Lore: A Database Management System for Semi-structured Data”, ACM SIGMOD Record, **26**(3), pp. 54-66, 1997.

Moyer M. J. and Ahamad M., “Generalized Role-based Access Control”, Proceedings of the 21st International Conference on Distributed Computing Systems, Mesa, AZ, 2001.

Osborn S., Sandhu R. and Munawer Q., “Configuring Role-Based Access Control to Enforce Mandatory and Discretionary Access Control Policies”, ACM Transactions on Information and System Security, **3**(2), pp. 85-106, 2000.

Rys M., “Bringing the Internet to your Database: Using SQL Server 2000 and XML to build Web and B2b Applications”, Proceedings of the International Conference on Data Engineering, 2001.

Sandhu R. S. and Mason G., “Lattice-Based Access Control Models”, IEEE Computer, pp. 9-19, Nov., 1993.

Sandhu R. S., Coyne E. J. and Feinstein H. L., “Role-Based Access Control Models”, IEEE Computer, pp.38-47, Feb., 1996.

Schoning H., “Tamino - A DBMS designed for XML”, Proceedings of the 17th International Conference on Data Engineering, Heidelberg, Germany, Apr. 2001.

Schmidt A., Kersten M., Windhouwer M. and Waas F., “Efficient Relational Storage and Retrieval of XML Documemts”, Proceedings of the 3rd International Workshop on the Web and Databases (WebDB 2000), Dallas, Texas, May, 2000.

Shimura T., Yoshikawa M. and Uemura S., “Storage and Retrieval of XML Documemts using Object-Relational Databases”, Proceedings of the 10th International Conference and Workshop on Database and Expert Systems Appllications (DEXA 1999), Florence, Italy, Aug.\Sep.1999.

Silberschatz A., Korth H. and Sudarshan S., “Database System Concepts”, Published by McGraw Hill Ed., 4th Edition, 2002.

Tian F., DeWitt D. J., Chen J. and Zhang C., “The Design and Performance Evaluation of Alternative XML Storage Policies”, ACM SIGMOD Record, **31**(1), pp. 5-10, 2002.

Vakali A. and Terzi E., “An Object-Oriented approach for effective XML Data Storage”, Proceedings. of the ECOOP Workshop on XML Object Tecnology, Cannes, France, 2000.

Winslett M., Ching N. Jones V. and Slepchin I., “Using digital credentials on the World Wide Web”, Journal of Computer Security, **5**(3), pp. 255-266, 1997.

Woo T. Y. C. and Lam S. S., “A Framework for Distributed Authorization”, 1st ACM Conference on Computer and Communications Security, Nov., 1993.

Yoshikawa M., Amagasa T., Shimura T., and Uemura S., “XRel: A Path-Based Approach to Storage and Retrieval of XML Documents using Relational Databases”, ACM Transactions on Internet Technology, **1**(1), pp. 110-141, 2001.

Zhu Y., and Lu K., “An Effective Data Placement Strategy for XML Documents”, Proceedings of the 18th British National Conference on Databases (BNCOD), Chilton, UK, Jul., 2001.

# Appendix A

In general, the most popular commercial native XML DBMSs can be classified into the following two categories with respect to their underlying storage model:

*Storage Model: Tree Structure*

- **NATIX** (Kanne and Moerkotte, 2000): It is the most well known native repository approach for storing, retrieving and managing tree-structured XML documents. The basic idea of NATIX is to represent and store XML documents based on their tree structure and it is oriented for research based implementations. The logical tree used for representing each XML document is splitted into sub-trees based on certain criteria. These sub-trees are the basic storage and query units.
- **SODA**[12]: It is another semi-structured DBMS tailored to manage XML information. In this system, the XML documents are stored in a single tree, which preserves all XML information and allows for efficient query and update operations along with optimizations that are XML-oriented and cannot be applied when conventional database schemes (like tables) are used. SODA is oriented for both research and commercial based implementations. Moreover, Soda provides also secured access control and authorization. The component which is responsible for preserving the security of XML documents in SODA database system is the access control manager.
- **Xyleme**[13]: A dynamic warehouse for XML documents of the Web. More specifically, Xyleme is designed to store, classify, index, integrate, query and monitor the XML documents. The performance of Xyleme heavily depends on the efficiency of its repository, which is the *Natix*. As we referred above, in Natix, the XML documents are represented by using an ordered tree structure. In order to store XML documents, Xyleme uses a combination of the following two approaches. In the first, the XML documents are stored in a conventional DBMS. In the second, the documents are stored as byte streams. Therefore, data are stored as trees until a certain depth where byte streams are used. The security of XML documents in Xyleme is guaranteed using access control lists. Access permissions are stored with each document, and users only get to view the documents they have rights to it. In addition, the top secure documents can also be stored in an independent partition.

[12] http://www.cse.unsw.edu.au/~soda/

[13] http://www.xyleme.com/

*Storage Model: Collection Structure*

- **Ipedo**[14]: It is a native XML database that allows its users to quickly and easily build XML-based information management solutions. The Ipedo architecture is composed of several components that make the Ipedo XML database accessible through standard Java programming interfaces. It provides both document-level as well as node-level access to XML and allows the users to organize XML documents by their schema. In particular, Ipedo supports an hierarchical storage engine which is highly optimized for XML information. The XML documents are organized into collections, which can be typed or un-typed. They are used to group related XML documents together. Typed collections contain a schema based on a DTD or XML Schema and all documents within that collection must conform to that schema. Un-typed collections can hold any number of XML documents regardless of the relationships between the schemas of those documents. Furthermore, Ipedo provides a sophisticated access control mechanism (security manager) in order to support a high-level security of XML documents. In particular, the security manager manages access to system resources by providing username and password authentication. Ipedo has also been designed to cooperate with Microsoft .NET technology.
- **eXist**[15]**:** It is an  Open Source native XML database which provides pluggable storage backends. According to this product, XML documents can be stored in the internal native XML database or an external relationship DBMS. XML can be  stored either in the internal, native XML database or an external relational DBMS. In eXist, XML documents the XML documents are stored using a DOM-tree (built from SAX-events) and are organized into hierarchical collections. Collections can be nested and are considered part of an XPath query string, so there is always a root collection. Indexes on collections may also be organized and thus the size of a collection may have a considerable impact on performance. As a disadvantage, eXist does not support direct manipulations of the DOM tree (like node insertions or removals). So, the XML documents should be deleted or updated as a whole. In order to ensure the integrity and compliance of XML documents, eXist supports an access control policy, which provides an interface to manipulate permissions and manage users. In particular, it organizes users in several groups, granting different permission sets for each one.
- **Tamino** (Schoning, 2001)**:** It is a modern database system that has been thoroughly designed for handling XML documents. In Tamino's database, the XML documents are stored in collections. More specifically, each XML document stored in Tamino resides in

[14] http://www.ipedo.com/html/products.html

[15] http://exist.sourceforge.net/

exactly one collection. In order to manage the XML documents effectively, Tamino supports a set of graphical tools. In addition, Tamino supports an XML-specific query language (Tamino-X-Query), which includes text retrieval facilities.

- **Xindice[16]:** It is an open source native XML database system, which is still evolving. In this system, XML documents are queried and updated using XML technologies, the first of which is W3C specification known as XPath. Using XPath it is possible to obtain a set of XML elements contained in a given XML document that conforms the parameters of the XPath query. In Xindice, XML documents are also organized using collections that can be queried as whole. A collection can be created either consisting of documents of the same type or a single collection can be created to store all documents together. Every XML document must be stored in at least one collection. While collections can be used strictly for organizational purposes, Xindice also allows for indexes to be created on collections to increase XPath performance. Moreover, Xindice supports a sophisticated mechanism for ensuring the integrity of XML documents. More specifically, Xindice provides an access control (on individual files or folders, by user based and/or group based) to XML documents.

[16] http://www.dbxml.org