XML Content Management based on Object-Relational Database Technology

B. Surjanto, N. Ritter, H. Loeser
Computer Science Department, University of Kaiserslautern,
P.O. Box 3049, 67653 Kaiserslautern, Germany
{surjanto, ritter, loeser}@informatik.uni-kl.de
http://www.uni-kl.de/AG-Haerder

Abstract

XML (Extensible Markup Language) is a textual markup language designed for the creation of self-describing documents. Such documents contain textual data combined with structural information describing the structure of the textual data. Currently, products and approaches for document-oriented application domains focus mainly on the textual representation when processing and analyzing documents. Usually, they do not take advantage of the availability of structural information and only support some of the relevant aspects of content management. On the other hand, existing research approaches for structure-oriented application domains prefer very fine granularities and give less attention to operations revealing textual document contents.

We introduce XCoP (XML Content Repository) as a repository which is based on an object-relational database management system (ORDBMS) and improves content management of XML documents thereby exploiting their structural information. It allows users to reuse and process textual portions of document contents, called *fragments*, thus avoiding data redundancy and, as a consequence, update anomalies on replicated data. Moreover, it enables collaborative development of documents and facilitates synchronization of fragment modification and versioning. In contrast to the existing tools and approaches, the fragment granularity in XCoP is flexibly configurable. At the same time, XCoP also maintains the structural information of document contents and manages the relations between them.

Keywords: WWW, ORDBMS, Repository, XML, Content Management, Reuse

1 Introduction

As a new standard document markup language for the Web, XML recommended by the W3C (World Wide Web Consortium) [BPS98] has gained worldwide attention from both industry and research community, since it is assumed to play a key role for data representation and data exchange on the Web. XML was developed as a textual markup language designed for the creation of self-describing documents, i.e., XML documents contain textual data interspersed with structural information. The structural description may also be used to express specific document semantics in order to facilitate the organization of and search in large sets of documents.

Unfortunately, current products and approaches for document-oriented application domains such as web content management (WCM), document management (DM), document content management (DCM), or digital libraries, concentrate on the textual representation when processing and analyzing documents. They do not take advantage of the availability of structural information and do not adequately and completely support other relevant aspects of content management: content reuse, search based on structural information, integration with operational databases as well as other external data sources in order to facilitate dynamic content generation, ensuring link consistency within and between documents, and collaborative development of documents. In contrast, existing approaches for structure-oriented application domains, e.g., data integration, management of semistructured data, or search engines, prefer very fine granularity such as markup elements within a document in order to exploit the corresponding structural information and give less attention to operations analyzing and processing textual document contents. We are convinced that processing support to both, textual document contents and structural information, would be very useful for many application domains and could lead to substantial improvements when large sets of documents have to be organized, classified and searched.

In this paper, we introduce XCoP (XML Content Repository) which is based on object-relational database technology and manages textual XML document contents together with their structural information. XCoP allows the reuse of document contents by providing so-called *fragments*. The use of our fragment concept reduces many operational problems. It avoids data redundancy and, therefore, update anomalies on replicated document contents. Furthermore, it enables collaborative development of documents, since multiple users may work on different fragments of a document concurrently. Unlike existing tools and approaches, the fragment granularity in XCoP is flexibly configurable by the users. Depending on their needs, users may specify, split, and merge fragments at runtime in suitable units they want to share. Frequently reused fragments indicate data locality and, therefore, are stored by XCoP in the database in their entirety as units. Regarding document composition this approach provides better performance in comparison to tools or approaches decomposing document contents down to the level of markup elements and store them in different places. At the same time, XCoP extracts the structural information of document contents in order to make query facilities and other repository services more effective. These structural descriptions together with their corresponding relations to document contents are maintained.

With the advent of object-relational database management systems (ORDBMSs) [Sto98], developers can extend the database system by user-defined data types (UDT), user-defined routines (UDR), and user-defined index structures. Furthermore, ORDBMSs allow users to access external data sources via SQL (Structured Query Language), enable (limited) object-oriented data modeling, still keep the advantages of relational database management systems (RDBMSs), and coexist with legacy data which mostly resides in relational databases (RDBs). We develop XCoP as part of an integrated Web database framework, called *iWebDB*, which is built by our database research group using the extensibility features of an ORDBMS.

To the best of our knowledge, there are currently no approaches of XML repositories that manage documents at the content level, allow the flexible definition of content granularity, provide operations on both textual representation and structural information, and finally are based on an ORDBMS. In this paper, we deal with the content management of XML documents in XCoP and do not discuss other value-added repository services [BD94] provided by XCoP such as version control, configuration management, collaboration, context management as well as workflow control.

The remainder of this paper is organized as follows: Section 2 briefly introduces XML and its supplemental techniques considered here. Section 3 provides a list of requirements for the content management of XML documents. In sections 4 and 5 we describe our approach for managing XML document contents and its implementation on the top of an ORDBMS. Previous related work is then reviewed in section 6. Finally, section 7 gives concluding remarks and an outlook to future work.

2 XML

XML is a metalanguage for describing document markup languages. It is extensible, i.e., it allows users to specify their own markup elements and structures which may be customized to their needs, and enables consuming applications to perform validation checks w.r.t. the specified structures. An XML document is defined as a textual data object which must satisfy the so-called *well-formedness* constraint and may further be *valid*, i.e., fulfill certain validity constraints [BPS98]. An XML document consists of markup elements each represented by a start tag and an end tag and character data in between, e.g., `<TITLE>Hello World</TITLE>`. Obviously, tags are delimited by angle brackets and end tags additionally contain a slash appearing in front of the markup element name (which is `TITLE` in our example). A markup element may also be empty, e.g., `<DUMMY></DUMMY>` (equivalent to `<DUMMY/>`). Furthermore, a markup element

may have attributes composed of name-value pairs embedded in the start tag or empty tag, e.g., <AUTHOR AGE="50">. An XML document is well-formed, if all its markup elements and attributes are properly delimited and nested.

Structuring XML Documents

Optionally, XML documents can be structured both physically and logically using a DTD (Document Type Definition) which comprises a set of *markup declarations*. A document is physically composed of its parts, called *(general) entities*, that are stored separately. In the DTD a (general) entity is specified by a special markup declaration, called *(general) entity declaration*. The XML specification also includes a facility for separately storing any part of a DTD, called *parameter entity*, using *parameter entity declarations*. Entities that reside outside their corresponding document are named *external entities*, otherwise *internal entities*.

An XML document is logically structured by means of the markup declarations contained in a corresponding DTD, which are *element declarations* and *attribute list declarations*. Together they represent a formal set of syntactical rules determining which markup elements and attributes are available and how they are to be applied. Moreover, it is possible to define *references* between markup elements within an XML document. Such references are described using attributes of types `ID` and `IDREF` resp. `IDREFS`. Each referred element must have a unique identifier specified in its `ID` attribute and references are represented by specifying `ID` values of the referred elements in `IDREF` resp. `IDREFS` attributes of the elements that refer to them.

```
<?XML VERSION="1.0"?>
<!DOCTYPE PRODUCT-CATALOG [
    <!ELEMENT PRODUCT (NAME, (QA-NOTE | MFG-INSTR)*)>
    <!ATTLIST PRODUCT
        PROD-ID   ID #REQUIRED
        TYPE      (f-good | sf-good | raw-mat) "f-good"
        MAT-USED  IDREFS #IMPLIED>
    <!ELEMENT NAME (#PCDATA)>
    <!ELEMENT QA-NOTE (#PCDATA | QA-METH | TEMP)*>
    <!ELEMENT QA-METH (#PCDATA)>
    <!ELEMENT TEMP (#PCDATA)>
    <!ELEMENT MFG-INSTR (#PCDATA)>
]>
<PRODUCT-CATALOG>
    <PRODUCT PROD-ID="FL-20" MAT-USED="PP-60 MIX-750">
        <NAME> Flexilene </NAME>
        <QA-NOTE>
           Perform QA immediately after jumbo production,
           use <QA-METH>J-61</QA-METH> at <TEMP>45</TEMP>.
        </QA-NOTE>
        <MFG-INSTR>
           Can only be produced on Line 2 or 4.
        </MFG-INSTR>
    </PRODUCT>
    <PRODUCT PROD-ID="PP-60" TYPE="raw-mat">
        <NAME> Polypropylene 60 </NAME>
    </PRODUCT>
    <PRODUCT PROD-ID="MIX-750" TYPE="sf-good">
        <NAME> FlexiMix </NAME>
        <QA-NOTE>
           For european customers, assure mixing quality
           using <QA-METH>M-16</QA-METH>. Otherwise, use
           <QA-METH>M-30</QA-METH> with standard params.
        </QA-NOTE>
    </PRODUCT>
</PRODUCT-CATALOG>
```

Figure 1: An example of a valid XML document with references

An XML document is said to be valid if it has a DTD and meets the constraints defined in the DTD. Figure 1 shows an example of a valid XML document (containing references).

Linking Mechanism

In addition to the unidirectional hyperlinks of HTML (Hypertext Markup Language), XLink (XML Linking Language, [Dot99]) provides more sophisticated linking constructs. In XLink, link characteristics are described by the so-called *linking element*s. Basically, there are two types of linking elements in XLink: *sim-*

ple links and *extended links*. Like HTML hyperlinks, simple links are unidirectional and thus have only one *locator* used to locate a *resource* whose identification is based on the URI syntax (Uniform Resource Identifier, [BLF+98]) and specified in the reserved attribute `HREF`. In contrast to a simple link, an extended link can connect any number of resources using *locator elements* and may be *multi-directional*. An extended link can also be *out-of-line* meaning that all participating resources are remote (not parts of the extended link).

Addressing Parts of XML Documents

Based on the logical tree respresentation of an XML document, internal structures of a document can be addressed using XPointer (XML Pointer Language, [DDM99]). XPointer can be used to locate nodes, points as well as character strings, and also to specify ranges of document parts.

3 Requirements for the Content Management of XML Documents

As motivated in the previous sections, we think that XML is the key technology for the next generation of Web applications and will become the standard format for data representation and data exchange on the Web. Its simplicity, extensibility, and structuring capability make XML very attractive for various application domains (e.g., document management, knowledge management, e-commerce, intelligent agents, and many more). The potential impact of XML is obvious. Using XML as a single language for specifying the huge amount of information produced and consumed over the Web would provide better interoperation among information providers as well as easier handling of documents by applications. Additionally, the capability of RDF (Resource Description Framework, [LS99]) to describe metadata (representing semantics) of Web resources makes the Web more appealing for both information providers and consumers. Furthermore, XML and XSL (Extensible Stylesheet Language, [DAB+99]) together offer a solution for a media-independent publishing that is achieved through the separation of the content and presentation format. Content creators can therefore concentrate on producing their document contents and information consumers may choose their preferred views. Obviously, we will have to be prepared to deal with a very large number of XML documents and their contents will have to be managed adequately and efficiently. As we think, a corresponding management system has to fulfill the following requirements:

- **Content Reuse**
  Sharing information for reuse must be considered being an essential property of a content management system when many information producers work with a large number of documents containing lots of information. The main benefits of information reuse are obvious: decrease of information replication and avoidance of update anomalies. In order to achieve these benefits, content management systems should not only allow information reuse at a coarse and fixed level of granularity, e.g., documents, but should be more flexible and allow dynamic determination of the appropriate granule of reuse.
- **Exploitation of Structural Information**
  An XML document contains different kinds of structural information, such as markup elements and attributes used, hierarchies of markup elements, or document structure information optionally defined in a corresponding DTD. Also XLink contains this kind of information, i.e., link information. Structural information is a kind of metadata that apparently is very useful for improving the search quality on document contents and enabling automatic content consistency checking (see next point).
- **Content Consistency Enforcement**
  The physical and logical structure of documents defined in the DTD as well as links between documents need to be handled adequately by enforcing their consistency, i.e., avoiding links to non-existing docu-

ments or resources. Actually, all kinds of structural information can effectively be used for consistency enforcement purposes.

- **Integrated Management of XML Documents and Other Resources**
  XML documents are by nature interconnected with other XML documents and other kinds of resources like various types of image files, a variety of word processing file types, etc. Thus, functions for an integrated management of XML documents and other resource types are required.

- **Utilization of DB Techniques**
  Due to the considerable amount of XML data anticipated in future applications, approved DB technology, e.g., for efficient secondary storage management, query processing, concurrency control, transaction management, recovery, etc., is likely to be exploited for efficient content management.

- **Dynamic Content Generation**
  The delivery of static XML documents or document contents is not sufficient for many application domains. Highly structured business data stored in DBS are often needed to be combined with XML data dynamically. Furthermore, dynamic generation of XML documents based on existing data is wanted. Thus, functions for accessing external data sources and document contents as well as for processing document templates have to be provided.

- **Repository Functionality Support**
  In order to manage several states of XML contents and to support the overall development life cycle, basic repository functions such as version control, configuration management, collaboration, context management, and workflow control are required. This additional functionality completes an XML content management system and leads to what we call an *XML content repository.*

4 XML Content Management in XCoP

In this section, we describe our approach for managing XML document contents. The textual representation of XML document contents, called *textual content*s, and the corresponding structural information are taken into consideration. DTDs, documents as well as (logical) document parts belong to textual contents. XCoP manages both textual contents and the related structural information of XML documents. For that purpose, a conceptual model and a set of operations that support the processing of textual contents and structural information are provided. Everything that is managed by XCoP is conceptually modeled as repository objects. They are further characterized by their attributes (e.g., owner, creation date, last modification date, etc.) and structurally organized into various repository object types such as resources, documents, fragments, elements, etc.

4.1 Managing Textual Contents

Although XML supports content reuse through external entities, it lacks functions for maintaining and ensuring the consistency of document parts belonging together. Indeed, physical structures in XML need to be maintained manually using entity declarations specified in DTDs. Changes in the physical structure of a document require manual updates in the DTD. In contrast, XCoP permits users to logically split document contents into parts without having to change DTDs, maintains these parts and ensures consistency automatically. Moreover, splitting document contents enables collaborative work since users can work on their respective parts independently.

Definition of Fragment

Comparable with the terminology used in the XML Fragment Interchange [GV99], we introduce the concept *fragment*. A fragment consists of textual contents and may further comprise a set of other fragments (child fragments) but not itself. A fragment is well-formed, if its textual contents match the production [43] `content` of XML 1.0 and all its child fragments are well-formed. Apparently, the *overall textual contents* of a well-formed fragment resulting from a proper integration of the textual contents of all direct and transitive child fragments is well-formed. A well-formed fragment is called a *root fragment*, if it has exactly one markup element at the top of its element hierarchy. A DTD fragment contains a set of markup declarations which are specified properly as defined in the XML specification and may also be composed of other DTD fragments. A DTD fragment is called a *DTD root fragment*, if it contains text delimited by brackets ("[...]") as demanded by the production `doctypedecl` of XML 1.0. Consequently, we define an XML document being well-formed, if it consists of exactly one root fragment, and being valid, if it has a DTD (with only one DTD root fragment) and exactly one root fragment whose textual contents meet the DTD.

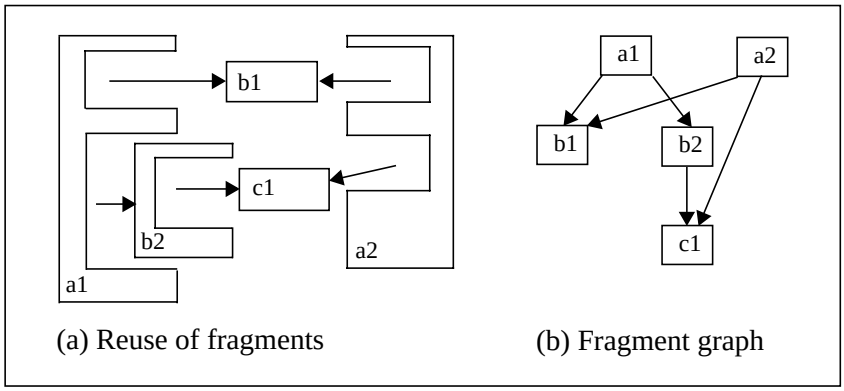

Figure 2: An example of fragment reuse and its fragment graph

Based on the fragment concept, we can flexibly determine portions of XML document contents as well as DTDs, so that they can be reused for building new (DTD) fragments. Figure 2 illustrates an example of fragment reuse and the corresponding structuring relationships spanning a so-called *fragment graph*. Fragment graphs are acyclic and directed. Their graphical representation uses rectangles as vertices for fragments and arcs to describe the *aggregration* relationships between parent and child fragments. Arcs are ordered according to the appearance of the child fragments in their parent fragment.

Determining Fragment Granularity

One of the main strengths of XCoP is that it permits users to specify the fragments they want to reuse in arbitrary size and in a flexible way. We provide two simple techniques, named explicit fragmentation and implicit fragmentation, for determining (DTD) fragments by the help of XCoP-specific PIs (*processing instruction*s), called *markup PIs*, that act like markup tags.

Explicit fragmentation happens by explicitly enclosing that part of a given document's or fragment's textual content, which is supposed to become a new fragment, by a pair of markup PIs (`<?XCOP TAG=<EXPL-FRAGMENT>?>` and `<?XCOP TAG=</EXPL-FRAGMENT>?>`) especially provided for that purpose. Implicit fragmentation happens by marking an excerpt of a DTD or DTD fragment. Corresponding markup PIs are `<?XCOP TAG=<IMPL-FRAGMENT>?>` and `<?XCOP TAG=</IMPL-FRAGMENT>?>`. The marked DTD excerpt may contain several element declarations. The meaning is that each element occurrence of an element declaration contained in the marked DTD excerpt is always to be treated as an own fragment. Both techniques allow users to share and reuse fragments without having to take care of DTD updates. By default entire document contents (without DTDs) are treated as fragments, if there are no fragmentation instructions. Thus, based on their needs users may freely determine a fixed fragment granularity (the spectrum reaches from the entire document down to a single markup element), decide for variable frag-

ment size, or just mix both. In addition, specifications of implicit fragmentation can be modified at any time. In this case, the affected fragments and their relations to each other are reorganized accordingly.

Figure 3 shows how the markup PIs are applied to specify implicit and explicit fragmentation. Based on the given implicit fragmentation, each PRODUCT element together with its textual contents will be handled as a fragment. Additionally, according to the explicit fragmentation specification the PRODUCT element with PROD-ID="MIX-750" has a child fragment containing a QA-NOTE element and its textual contents.

```
<?XML VERSION="1.0"?>
<!DOCTYPE PRODUCT-CATALOG [
    <?XCOP TAG=<IMPL-FRAGMENT>?>
    <!ELEMENT PRODUCT (NAME, (QA-NOTE | MFG-INSTR)*)>
    <?XCOP TAG=</IMPL-FRAGMENT>?>
    ...
]>
<PRODUCT-CATALOG>
    <PRODUCT PROD-ID="PP-60" TYPE="raw-mat">
        <NAME> Polypropylene 60 </NAME>
    </PRODUCT>
    <PRODUCT PROD-ID="MIX-750" TYPE="sf-good">
        <NAME> FlexiMix </NAME>
        <?XCOP TAG=<EXPL-FRAGMENT>?>
        <QA-NOTE>
            For european customers, assure mixing quality
            using <QA-METH>M-16</QA-METH>. Otherwise, use
            <QA-METH>M-30</QA-METH> with standard params.
        </QA-NOTE>
        <?XCOP TAG=</EXPL-FRAGMENT>?>
    </PRODUCT>
</PRODUCT-CATALOG>
```

Figure 3: An example of fragmentations

Conceptual Model for Managing Textual Contents

Figure 4 illustrates a basic conceptual model for textual contents, i.e., XML documents, DTDs, fragments and other kinds of resources. In our model, a resource is represented by a repository object which can be identified by a URI. Resources are either binary resources (e.g., images or executable files) or textual resources (e.g., ASCII text files). The latter are refined in DTDs, documents, and fragments. A fragment may consist of several child fragments, which are ordered and may be shared with other parent fragments. Fragments include markup elements and are further specialized in DTD fragments that consist of markup declarations. They are well-formed resp. valid according to the definitions given previously.

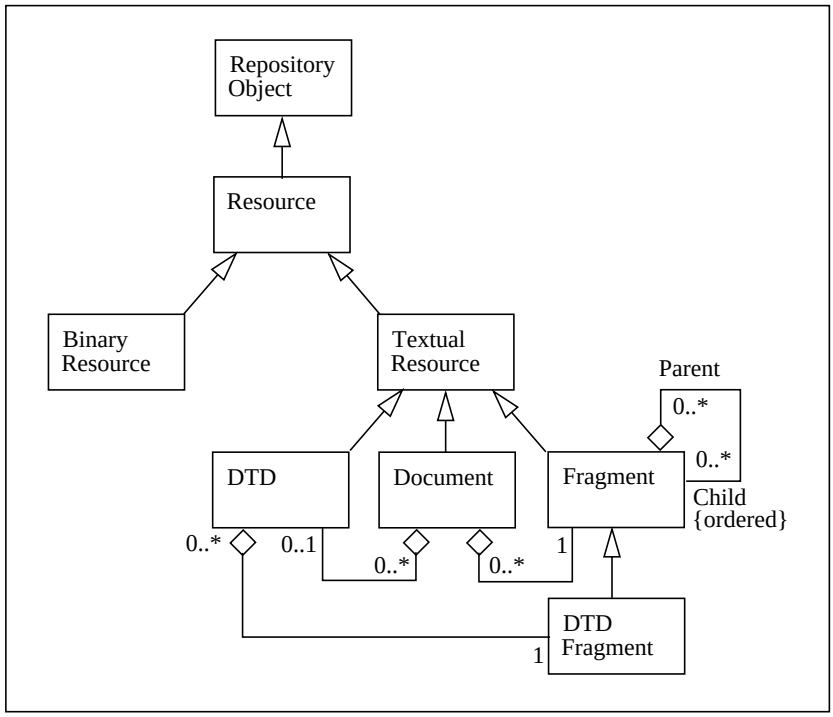

Figure 4: Conceptual model for managing textual contents

4.2 Exploiting Structural Information

Currently, we use the following kinds of structural information in order to support automatic enforcement of document contents consistency and to provide appropriate search facilities:

- *Well-formed XML document information*:
  markup elements, attributes, references between markup elements, *composition* relationships resulting from the content model of markup elements, and the appearance order of markup elements, as well as composition relationships between attributes and elements;
- *DTD information*:
  entity declarations, element declarations, attribute list declarations, composition relationships between element, element ordering as well as composition relationships between elements and attribute lists;
- *XLink information*:
  various types of linking elements (e.g., simple links, extended links, locator elements, etc.) together with XLink-specific attributes such as `type`, `href`, `role`, `arc`, etc.

Conceptual Model for Managing Structural Information

Figures 5 and 6 show conceptual models for managing structural information. Figure 5 illustrates that a fragment has an ordered, non-empty set of markup elements. Each of them may have one or more attributes. We further divide attributes into attributes of type `ID` and attributes of type `IDREF(S)`. On one hand, a markup element with `ID` attribute can be referred by one or more other element(s) via `IDREF(S)` attributes. On the other hand, an element with an `IDREF(S)` attribute refers to exactly one resp. many elements each with an `ID` attribute. A simplified conceptual model of XLink is presented in Figure 5. Various kinds of linking elements in XLink (i.e., simple links, extended links, and locator elements) are modeled as types of markup elements. An extended link has an non-empty, ordered set of locator elements. Finally, a simple link or a locator element refers to exactly one resource.

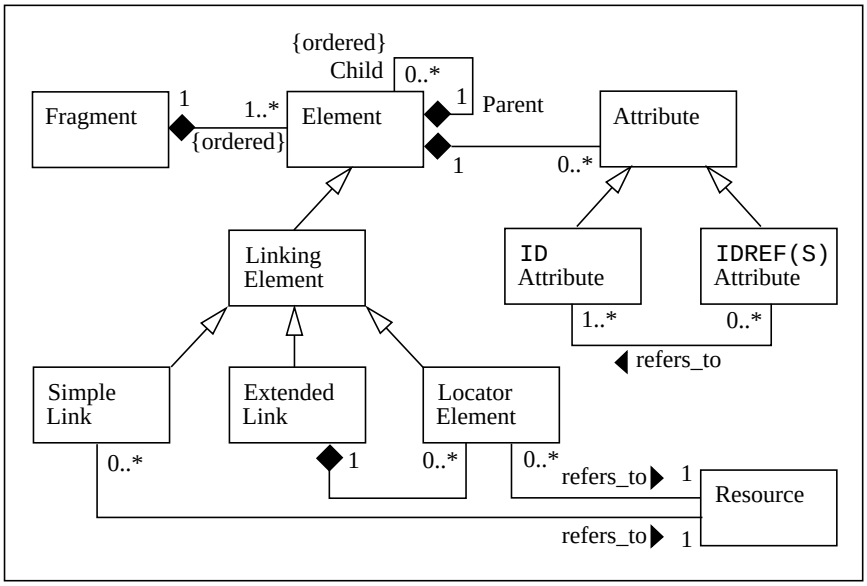

Figure 5: Modeling structural information within fragments

Figure 6 shows that markup declarations, markup elements, and attributes are also modeled as repository objects. Markup declarations are further classified into element declarations, attribute declarations, and entity declarations, whereas an element declaration may contain one or more attribute declaration(s) and consist of an ordered set of other element declarations. DTD fragments comprise an ordered and non-empty set

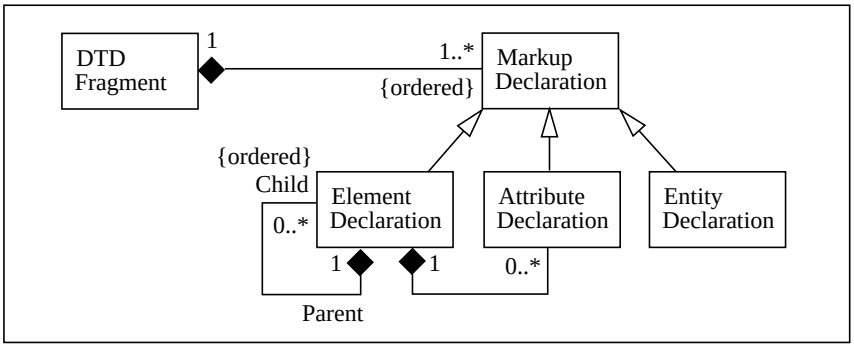

Figure 6: Modeling structural information within DTD fragments

of markup declarations. Due to simplicity, we ignore parameter entities and general entities as well as their relationships to entity declarations in the models given in this paper.

4.3 Processing Repository Objects

This section gives an overview of the operations provided to support the processing of repository objects, particularly those that are related to textual contents and structural information. Operations are defined specifically for each repository object type. Although our actual modeling approach is object-oriented, we give a more generic description in the following in order to keep our explanations short and simple. We distinguish four categories: manipulation operations on repository objects, manipulation operations on relationships, retrieval operations, and specific operations. Well-formedness resp. validity constraints are checked during operation executions whenever required, especially for such operations that effect content changes.

Manipulation Operations on Repository Objects

This category contains operations for the creation, modification, and deletion of repository objects.

- `createObject (initTextualContents: String): Object`
  This operation accepts an input string `initTextualContents` representing the initial textual contents of the repository object to be created. After its successful execution, the created repository object is delivered. When creating a document object, the corresponding root fragment and DTD (if any) are identified and created (again by calling `createObject`) and associated with the document object (see `CreateRelationship` operation below). Likewise, a create operation for a DTD leads to the creation of the related DTD root fragment and the corresponding relationship. In case of the creation of a (DTD) fragment object, fragmentation markups within the textual contents of the fragment are recognized and the `createObject` operation is recursively performed for each child fragment. Afterwards, the corresponding aggregation relationships are created. Similarly, structural information of a (DTD) fragment, i.e., markup declarations resp. markup elements and attributes, are extracted and the corresponding objects and relationships are created.
- `updateObject (newTextualContents: String): Boolean`
  An update operation requires the new textual contents supposed to replace the existing textual contents of a repository object. Update operations are usually performed by calling deletion and creation operations on subordinated data structures.
- `deleteObject (): Boolean`
  This operation deletes a repository object. An object can only be deleted if it does not share (aggregation) subordinated objects with other objects of the same type. Similar to the operations mentioned previously, a delete operation may cause cascaded deletions.

Manipulation Operations on Relationships

The operations of this category are used explicitly to establish and delete relationships between repository objects.

- `createRelationship (partnerObject: Object [, relType: RelationshipType,`
  `posExpr: PosExpr]): Boolean`
  A relationship to another repository object (e.g., from a document to a DTD, from a parent to a child fragment, from a fragment to an element, etc.) is established using this operation. The optional parameter `posExpr` specifies the relationship position and is needed only in case of an ordered relationship type.

For example, a position of a composition relationship between a parent and a child element is given by a non-negative integer number specifying the order position of the child element in its parent element, whereas an XPointer expression is used to specify a position of an aggregation relationship between a parent and a child fragment. If both participating repository objects are elements, then the relationship type `relType` to be taken into account can additionally be restricted (composition or reference). In contrast, when creating a relationship between two elements, an element and an attribute, or two markup declarations, changes in the corresponding fragment are performed accordingly.

- `deleteRelationship (partnerObject: Object): Boolean`
  This operation just deletes a relationship existing to the partner repository object. The corresponding objects remain in the repository.

Retrieval Operations

Retrieval operations allow us to get the textual contents of a repository object and provide a navigational access mechanism for repository objects based on their relationships to other repository objects.

- `getObjectContents ([level: Integer]): String`
  By this operation, the textual contents of a repository object are retrieved. If applied to a repository object, which is not an attribute objects, a non-negative integer number `level` is required representing the number of the hierarchy levels (of the same object type) to be examined. This operation delivers the textual contents which result from integrating the textual contents of all child objects up to the specified level.

- ```
  getObjects (targetObjectType: ObjectType [, relType: RelationshipType,
              targetObjectRole: ObjectRole, level: Integer]): ObjectList
  ```
  This operation delivers a list of objects of type `targetObjectType` that are connected to a repository object. A non-negative integer number `level` representing the number of hierarchy levels to be examined and the role of the target repository object `targetObjectRole` (i.e., child or parent) can optionally be specified if origin and target object types are the same. If both are elements, then the relationship type to be taken into account can additionally be restricted (composition or reference).

Specific Operations

Unlike the operations of the previous three categories, specific operations are exclusively applicable to certain repository object types. In the following, the specific operations on fragments are explained:

- `splitFragment (targetRangeExpr: XPtrExpr): Fragment`
  A (textual) excerpt (specified by the XPointer expression `targetRangeExpr`) of a fragment object is extracted to become a new child fragment.

- `mergeFragment (role: FragmentRole, allowReplication: Boolean): Fragment`
  This is the reverse operation to the previous one. Depending on the `role` (i.e., `child` or `parent`) of a fragment object, its textual contents are merged with its parent resp. child fragment(s). If shared fragments are involved it depends on the flag `allowReplication`, whether these shared fragments are replicated or the merge is denied.

- `matchFragment (targetExpr: XPtrExpr): Boolean`
  This operations checks whether the textual contents of a fragment object contain parts that match the text specified by `targetExpr`.
  ```

5 Implementation

In this section, we give an overview of iWebDB, our integrated Web database framework [LR99], and briefly describe how it can be used to implement XCoP. We want to start with a short introduction to object-relational database (ORDB) technology, our implementations are based on.

5.1 Useful ORDB Technology

Current activities in developing ORDBMSs [SBM98] aim at integrating object-oriented concepts into the relational model. The *extensibility* property of ORDBMS can be considered to be extremely beneficial for efficiently implementing repository functionality, since it allows for:

- adding UDTs consisting of specially designed data structures and corresponding operations (UDRs) to the database schema;
- capturing arbitrary formats within UDTs by exploiting large objects (LOBs) types;
- linking externally stored data with database entries and controlling access to these (externally stored) data by database mechanisms;
- exploiting pre-defined extensions for accessing the database via the web, for managing data of various formats (text, HTML, video, audio, image, etc.);
- dynamically transforming externally stored data into relations, which may be incorporated into SQL processing.

The availability of these features was the reason, why we decided for ORDBMS. In the following, it will be outlined how we used these features.

5.2 iWebDB

iWebDB is implemented based on an ORDBMS and exploits OR extensibility features such as UDT, UDR and external data access. As depicted in Figure 7, iWebDB currently consists of eight modules, five of them being DBS extensions, so-called DataBlades (Informix), Extenders (IBM), or Catridges (Oracle): Doc, ED (External Data), eXtract, DG (Document Generator), and XCoM (XML Content Manager). In addition, two client applications, SM (Site Manager) and XCoEx (XML Content Explorer), exist. The module

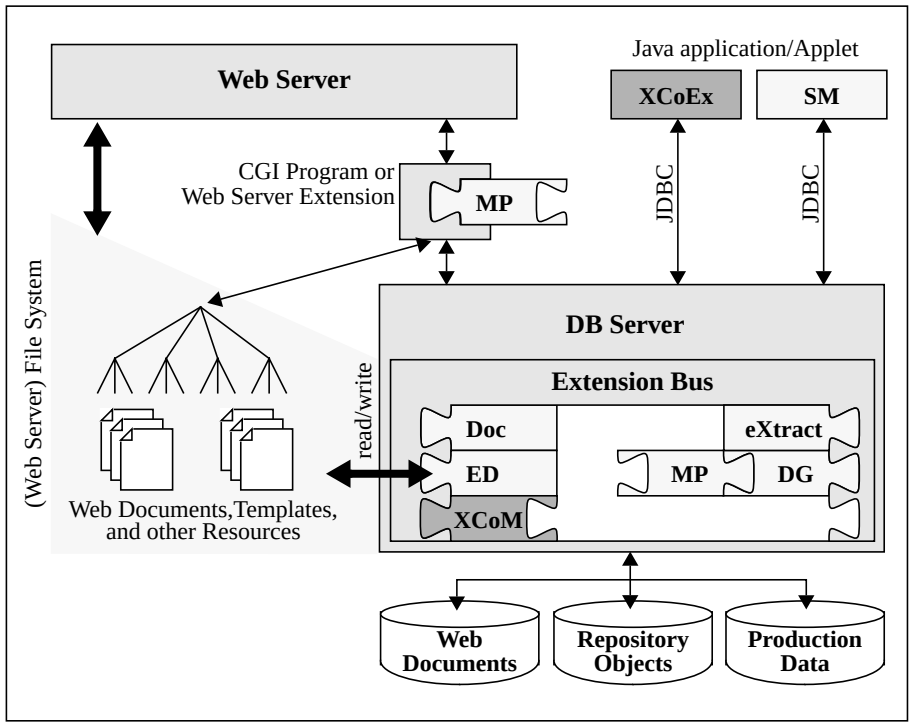

Figure 7: Architecture of iWebDB

MP (Macro Processor) is a function library that can be used for client-side as well as for server-side applications. XCoM and XCoEx are specific components of XCoP.

- **iWebDB/Doc**
  All base types and tables for WCM, e. g., types for storing XML, HTML, Postscript, and other text documents as well as images of different formats are provided by iWebDB/Doc. Each document is stored entirely. In addition to document storage, iWebDB/Doc provides functionality for managing, querying, and analyzing the (HTML) content, e. g., functions to extract the title, headers, hyperlinks, etc.

- **iWebDB/ED**
  This module helps to simplify Web site administration by allowing the integration of externally stored files into iWebDB. It is based on the abstract table facility of SQL/MED (Management of External Data, [JTC99]). File system data accessible via Web protocols, such as "`file:`", "`ftp:`", and "`http:`", can be made available in relational tables, the file contents in document formats provided by iWebDB/Doc. Available data can be queried using SQL. Thus, external data can seamlessly be integrated into the system. Users and other iWebDB components can access documents stored in a file system as well as documents managed by the DBS in the same way by using SQL. iWebDB/ED is used by the document generator module to write generated documents to the file system and, in case of document templates provided in the file system, to read them.

- **iWebDB/eXtract**
  In order to enhance the search capabilities provided by iWebDB/Doc, iWebDB/eXtract provides additional functions for analyzing documents. Furthermore, functionality is provided for storing extracted data in a special index structure being exploited by a search engine.

- **iWebDB/SM**
  This module is the iWebDB tool for site management needed by all so-called information providers. Based on a graphical user interface (GUI), all tasks concerning installation and maintenance of Web documents and directories, user and group administration as well as en-/disabling document generation can easily be performed.

- **iWebDB/MP**
  This module is a function library designed to be the foundation for building gateway programs for Web-based DB access. It offers an extensible macro processor for embedding special tags into Web documents. Among other things, it provides functionality for the execution of SQL statements, invocation of Java functions, and the composition of documents. The library can be used to realize a fully functional client- or server-side Web database gateway (see Figure 7). In addition, it serves as a basis for iWebDB/DG (see below). The macro processor offers an API to access the context manager, so that namespaces and variables can be created as well as values be read or set. Using this API, applications based on iWebDB/MP can create their own namespace, define variables, and set values.

- **iWebDB/DG**
  This module provides a document generator based on iWebDB/MP. Using SQL triggers, static Web pages stored in the Web server file system are automatically refreshed whenever DB data the pages are based on has been modified. Thus, except for Web applications, Web information systems can avoid dynamically generated documents to offer up-to-date information.

5.3 XCoP

After the overview of the iWebDB architecture and most of its components, we want now outline the two XCoP-specific components, XCoEx and XCoM.

- **XCoEx**
  This module provides a GUI for manipulating and retrieving repository objects, resp. fragments. A text editor for editing textual contents is included. In addition, administration tasks related to repository objects and their mapping to the (web server) file system as well as user and group access privileges is also supported. XCoEx is implemented as a Java application using JDBC (Java Database Connectivity).
- **XCoM**
  This module is responsible for the server-side management of repository objects. The implementation of the conceptual models and the corresponding operations on the top of an ORDBMS is realized in XCoM.

XCoM will be detailed in the following subsections by showing how the conceptual models and operations on repository objects can easily be mapped into an OR data model based on IDS/UDO (Informix Dynamic Server with Universal Data Option, [Inf97]). Current limitations of IDS/UDO related to modeling capabilities are also mentioned.

Mapping XCoP's Conceptual Model to OR Data Model

The repository object types (as described in Section 4) are captured within an object-relational DB schema as follows (see Figure 8 for examples). Each object type is mapped to a so-called *named row type*, and each subtype relationship to a corresponding *sub-class* relationship type (see keyword UNDER in Figure 8(a)). Each subtype inherits the attributes (i.e., data fields) as well as operational properties (i.e., routines, aggregates, and operators) of its supertype. Based on the defined type hierarchy, a table hierarchy can be specified (also illustrated in Figure 8(a)). The primary key constraint defined in the supertable `repository_object` is inherited by all subtables.

With IDS/UDO, *routine overloading* can be accomplished as shown in Figure 8(b). Routine overloading allows users to assign a single name (e.g., `create_object`) to multiple routines having different signatures.

IDS/UDO does not support reference types and provides only limited set operation capability on collection types. Because both techniques can be very useful for the implementation of relationship type semantics (e.g., aggregation), we have to employ referential integrity constraints instead as a workaround (see Figure 8(c)).

Considering that a fragment represents the smallest unit of information specified by users and indicates data locality, it is advantageous to also store it as a unit. Fragments are therefore stored as rows of a single table `fragment`. Obviously, this storage approach achieves better performance when constructing a document from its fragments as compared to other approaches that compose a document from its elements. Moreover, concurrent work on fragments can be directly controlled by using the row locking mechanism offered by most (O)RDBMS vendors. Aggregation relationships between fragments are mapped to a separate table (`fragment_graph`). The `order_pos` column in the `fragment_graph` table reflects the order position of a child fragment within its parent fragment.

Storing structural information, especially elements and attributes, within relational tables is a delicate matter as reported in [FK99, STH+99, DFS99]. Based on the mapping schemes given in [FK99], we use the *attribute approach* for storing structural information. Moreover, [FK99] describes techniques for translating

queries posed in an XML query language (e.g., XML-QL [DFF+98]) into SQL queries for the alternative mapping schemes. Indeed, search quality on document contents is improved significantly by exploiting their structural information which is extracted and stored in tables. Furthermore, dependencies between a fragment and its structural information are automatically controlled by triggers. Changes in a fragment will activate a trigger which accordingly maintains the corresponding structural information, and vice versa.

Other extensions of iWebDB, particularly MP, DG, and ED, allow dynamic content generation of documents (and fragments) integrating existing legacy databases or external files by using special macro tags and templates. In our first prototype, we store the textual contents as LOBs, since IDS/UDO does not offer mechanisms comparable to the *IBM DataLinks* [Dav99] allowing to handle links to externally stored files. Finally, we again utilize referential integrity constraints and triggers for automatic enforcement of link and reference consistency.

```
CREATE ROW TYPE repository_object_t (
    oid VARCHAR(8) NOT NULL, owner VARCHAR(20), cdate DATE);
CREATE ROW TYPE resource_t (
    uri VARCHAR(255), contents BLOB) UNDER repository_object_t;
CREATE ROW TYPE textual_resource_t (
    description VARCHAR(255)) UNDER resource_t;
CREATE ROW TYPE document_t (
    content_status VARCHAR(8), dtd_oid VARCHAR(8), root_fragment_oid VARCHAR(8)
    ) UNDER textual_resource_t;
CREATE ROW TYPE dtd_t (
    name VARCHAR(60), root_fragment_oid VARCHAR(8)) UNDER textual_resource_t;
CREATE ROW TYPE fragment_t (
    is_root BOOLEAN) UNDER textual_resource_t;
CREATE ROW TYPE dtd_fragment_t (
    ) UNDER fragment_t;

CREATE TABLE repository_object OF TYPE repository_object_t (PRIMARY KEY (oid));
CREATE TABLE resource OF TYPE resource_t UNDER repository_object;
CREATE TABLE textual_resource OF TYPE textual_resource_t UNDER resource;
CREATE TABLE document OF TYPE document_t UNDER textual_resource;
CREATE TABLE dtd OF TYPE dtd_t UNDER textual_resource;
CREATE TABLE fragment OF TYPE fragment_t UNDER textual_resource;
CREATE TABLE dtd_fragment OF TYPE dtd_fragment_t UNDER fragment;
```

(a) Definition of type hierarchy and typed table hierarchy

```
CREATE FUNCTION create_object (document) RETURNING VARCHAR(8);
CREATE FUNCTION create_object (dtd) RETURNING VARCHAR(8);
CREATE FUNCTION create_object (fragment) RETURNING VARCHAR(8);
CREATE FUNCTION create_object (dtd_fragment) RETURNING VARCHAR(8);
```

(b) Routine overloading

```
ALTER TABLE document ADD CONSTRAINT (
    FOREIGN KEY (dtd_oid) REFERENCES dtd (oid),
    FOREIGN KEY (root_fragment_oid) REFERENCES fragment (oid));

ALTER TABLE dtd ADD CONSTRAINT (
    FOREIGN KEY (root_fragment_oid) REFERENCES dtd_fragment (oid));

CREATE TABLE fragment_graph (
    parent_oid VARCHAR(8), child_oid (8), order_pos SMALLINT,
    PRIMARY KEY (parent_oid, child_oid),
    FOREIGN KEY (parent_oid) REFERENCES fragment (oid),
    FOREIGN KEY (child_oid) REFERENCES fragment (oid));
```

(c) Implementation of relationships between repository objects

Figure 8: A sample OR schema

6 Related Work

With XCoP we contribute an integrated solution approach to content management requirements of XML documents as presented in section 4. Unlike other tools and approaches that are either based on an RDBMS or an object-oriented database management system (OODBMS), the implementation of our approach is based on an ORDBMS. By exploiting the extension capabilities of an ORDBMS many tasks concerning content management can be delegated to the DBMS in an integrative way.

While XCoP supports content reuse extensively, iWebDB/Doc [LR99] only offers basic storage options at the document level. An implicit fragmentation technique for structured document contents was introduced in HyperStorM [BAN+97], a DB application framework designed for managing SGML (Standard Generalized Markup Language) documents and built on the top of an OODBMS. We have adapted this technique in order to apply it to XML documents in XCoP. Beyond the possibilities provided in HyperStorM, we allow users to modify implicit fragmentation specifications at run-time and additionally offer the explicit fragmentation method. HyperStorM does not exploit structural information of fragments and lacks content management functionality, e.g., content reuse, external data sources integration, etc. Recently, Lore [GMW99], a DBMS specifically designed to handle semistructured data, has been modified in order to manage XML data. Lore focuses on processing fine-grained structural information rather than coarse-grained textual content. It also lacks sophisticated content management functionality.

Fragmentation of XML documents has been proposed to W3C in XML Fragment Interchange [GV99]. This proposal only specifies a mechanism for processing parts of an XML document without having to take the entire document into account. Issues concerning fragment reuse and collaboration among concurrently working users are beyond its scope.

There are a number of commercial products for DM resp. DCM, e.g., *DynaBase* [Dyn99] of *Inso Corporation*, *Documentum 4i* [Doc99] of *Documentum*, *BladeRunner* [Bla99] of *the e-content company*, and *POET CMS* [Poe99] of *POET Software*. All products use a full-text search engine but offer limited search capabilities w.r.t. structural information. Furthermore these systems support versioning and configuration management, collaborative works, and workflow functionality. Integration with external data sources residing in file systems are not supported by all products. Most of them rely on OODBMS except for DynaBase and Documentum that use RDBMS. DynaBase and Documentum 4i are DM products and thus use files as the smallest content unit, while others enable content reuse in variable size. Finally, BladeRunner and POET CMS do not support dynamic content generation.

Recent research approaches [FK99, STH+99, DFS99] propose techniques for storing fine-grained XML data in an RDBMS in order to leverage relational technology. We apply two techniques described in [FK99] for storing both fine-grained structural information and textual contents in variable granularity, while commercial products such as *eXcelon* [Exc99] of *eXcelon Corp.* and *Tamino* [Tam99] of *Software AG* exclusively support processing of fine-grained XML. This, indeed, may lead to unnecessary fragmentations, complicates the content management of XML documents and, therefore, does not support content reuse adequately.

7 Conclusion and Outlook to Future Work

In this paper, we have presented XCoP, a repository approach designed for content management of XML documents. XCoP integrates many content management services by utilizing ORDB technology. To the best of our knowledge XCop is the first approach exploiting ORDB technology for these purposes. Unlike exist-

ing products and approaches, processing both textual contents as well as corresponding structural information of XML documents are supported. XCoP facilitates content reuse through fragmentation which obviously avoids content redundancy and enables collaborative work on documents. With our fragmentation methods users can logically decompose an XML document into fragments and dynamically determine the granularity of the units they want to share. Two fragmentation methods have been introduced: implicit and explicit fragmentation. An implicit fragmentation is applied to markup declarations of a DTD and an explicit fragmentation to document contents. Furthermore, conceptual models for the management of textual contents and structural information have been presented and corresponding manipulation operations have been introduced. Finally, implementation techniques of XCoP based on the ORDBMS IDS/UDO with its current limitations have been discussed. The mapping of the conceptual models to an ORDB schema has been exemplary presented and ORDBMS features used, e.g., UDT, UDR, constraints, locking mechanism, and trigger, as well as those that are desirable, i.e., reference types and set operation support on collection types, have been discussed.

Currently, we are embedding structural information contained in RDF and working on version control, configuration management, and collaboration support. In contrast to WebDAV [CK00], a protocol for distributed authoring and versioning applying at the resource level (i.e., documents), our approach aims at controlling the management of document fragments. Furthermore, context management and workflow control functionality are planned to be accommodated in XCoP.

Acknowledgments

The authors thank T. Härder, M. Flehmig, U. Marder, and H.-P. Steiert for fruitful discussions and helpful comments on earlier versions of this paper.

References

[BAN+97] Böhm, K., Aberer, K., Neuhold, E. J., Yang, X.: *Structured Document Storage and Refined Declarative and navigational Access Mechanisms in HyperStorM*, The VLDB Journal, Vol. 6, pp. 296-311, 1997.

[BD94] Bernstein, P. A., Dayal, U.: *An Overview of Repository Technology*, Proc. of 20th VLDB, pp. 705-713, Santiago, Chile, September 1994.

[Bla99] *BladeRunner 1.5*, the e-content company, 1999. `http://www.xmlecontent.com/`.

[BLF+98] Berners-Lee, T., Fielding, R., Irvine, U. C., Masinter, L.: *Uniform Resources Identifiers (URI) - Generic Syntax, Network Working Group RFC 2396*, IETF, August 1998. `http://www.ietf.org/rfc/rfc2396.txt`.

[BPS98] Bray, T., Paoli, J., Sperberg-McQueen, C. M.: *Extensible Markup Language (XML) 1.0*, W3C Recommendation, February 1998. `http://www.w3.org/TR/REC-xml-19980210`.

[CK00] Clark, J., Kaler, C.: *Versioning Extensions to WebDAV*, Internet Draft, IETF Delta-V Working Group, February 2000. `http://www.webdav.org/deltav/protocol/`.

[Dav99] Davis J. R., : *DataLinks - Managing External Data with DB2 Universal Database*, IBM Corporation, February 1999. `http://www-4.ibm.com/software/data/pubs/papers/.`.

[DAB+99] Deach, S., Adler, S., Berglund, A., Caruso, J., Milowski, A., Zilles, S.: *Extensible Stylesheet Language (XSL) Specification*, W3C Working Draft, April 1999. `http://www.w3.org/TR/WD-xsl`.

[DDM99] DeRose, S., Daniel Jr., R., Maler, E.: *XML Pointer Language (XPointer)*, W3C Working Draft, December 1999. `http://www.w3.org/TR/WD-xptr`.

[DFF+98] Deutsch, A., Fernandez, M., Florescu, D., Levy, A., Suciu, D.: *XML-QL: A Query Language for XML*, W3C Note, August 1998. http://www.w3.org/TR/1998/NOTE-xml-ql-19980819/.

[DFS99] Deutsch, A., Fernandez, M., Suciu, D.: *Storing Semistructured Data with STORED*, SIGMOD'99, pp. 431-442, Philadelphia, 1999.

[Doc99] *Documentum 4i*, Documentum, 1999. http://www.documentum.com/.

[DOT99] DeRose, S., Orchard, D., Trafford, B.: *XML Linking Language (XLink)*, W3C Working Draft, December 1999. http://www.w3.org/TR/xlink.

[Dyn99] *DynaBase*, eBusiness Technologies, Inso Corporation, 1999. http://www.ebt.com/.

[Exc99] *eXcelon*, eXcelon Corporation, 1999. http://www.odi.com/.

[FK99] Florescu, D., Kossmann, D.: *A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database*, Technical Report, INRIA, France, 1999.

[GMW99] Goldman, R., McHugh, J., Widom, J.: *From Semistructured Data to XML: Migrating the Lore Data Model and Query Language*, Proc. of WebDB'99, pp. 25-30, Philadelphia, Pennsylvania, June 1999.

[GV99] Grosso, P., Veillard, D.: *XML Fragment Interchange*, W3C Working Draft, June 1999. http://www.w3.org/TR/WD-xml-fragment.

[Inf97] *Informix Dynamic Server with Universal Data Option 9.1.X*, Informix Software, Inc., 1997. http://www.informix.com/.

[JTC99] ISO/IEC JTC1/SC32: *Information Technology - Database Language SQL - Part 9: Management of External Data*, ISO, 1999.

[LR99] Loeser, H., Ritter, N.: *iWebDB - Integrated Web Content Management based on Object-Relational Database Technology*, Proc. of IDEAS'99, Montreal, Canada, pp. 92-97, August 1999.

[LS99] Lassila, O., Swick, R. R.: *Resource Description Framework (RDF) Model and Syntax Specification*, W3C Recommendation, February 1999. http://www.w3.org/TR/REC-rdf-syntax.

[Poe99] *POET Content Management Suite 2.0*, POET Software, 1999. http://www.poet.com/.

[STH+99] Shanmugasundaram, J., Tufte, K., He, G., Zhang, C., De Witt, D., Naughton, J.: *Relational Databases for Querying XML Documents: Limitations and Opportunities*, Proc. of 25th VLDB, pp. 302-314, Edinburg, Scotland, 1999.

[SBM98] Stonebraker, M., Brown, P., Moore, D.: *Object-Relational DBMSs - Tracking the Next Great Wave*, 2nd edition, Morgan Kaufmann Publishers, 1998.

[Tam99] *Tamino*, Software AG , 1999. http://www.softwareag.com/.