Coping with Semantics in XML Document Management

Thomas Kudrass
Leipzig University of Applied Science,
Department of Computer Science and Mathematics, D-04251 Leipzig
kudrass@imn.htwk-leipzig.de

This paper tells the story of a war waged unknowingly by many system architects dealing with XML documents and databases. We discuss the semantic ambiguities and deficiencies of XML that explain many mismatches we encounter when mapping XML documents to databases. As a remedy, we show the benefits of using RM-ODP viewpoints to structure an XML document specification and an XML management system as well. We discuss the usage of generic relationships, as they are introduced in the General Relationship Model (GRM) for the specification of the invariants in different document views. Moreover, we discuss the realization issue for the storage of XML documents in databases. We give some examples how viewpoints and specification/realization may occur concurrently in one XML document which proves the need for semantic models in different document views.

1. Motivation

XML is emerging as the dominant standard for representing data on the internet [GP00]. It is called the successor to HTML and is considered the lingua franca for data exchange in distributed applications. XML was originally designed as a universal media-independent publishing format that supports all kinds of document users at every level. Today much effort is being placed in combining XML and database technology to develop full-fledged document management systems.

In spite of the enthusiasm generated by XML, but on the other side there are many misconceptions regarding the semantics of XML. Some users refer to XML as "semantic markup" contrary to HTML. Others say "XML is just syntax - no semantics!". XML has merely the potential to improve the document semantics through markup because with XML as language, it is possible to introduce new tags that represent some meta information. Like any other specification language, it can be used to express the semantics of documents and their components to a certain extent. Our projects are dealing with the development of XML management tools on top of commercial DBMSs. Therefore, we have to analyze the limitations of XML regarding semantics before implementing such a system. To capture of the various semantic aspects we need a structure that helps to understand the XML/DB problem from different viewpoints.

The remainder of this paper is structured as follows: Section 2 introduces the basic problem of XML regarding semantics. Section 3 discusses the main issues of XML-to-database mapping implied by the lack of XML semantics. Section 4 outlines the RM-ODP concepts of viewpoints and generic relationships applied to document modeling. Their usability for XML document specifications is shown in section 5. Section 6 stresses the separation of concerns in the realization of XML based systems. Finally, the conclusions sketch how to proceed when building those systems with databases.

2. About XML: A Semantic Perspective

HTML is just one of several markup languages on the market whose popularity has grown due to its importance as the standard format for web documents. HTML defines a standard set of tags with standardized meanings and standardized rules of use. So it is possible to define a logical document structure with a predefined set of building blocks. By using Cascading Style Sheets (CSS) it is also possible to separate the physical presentation of the document's building blocks from the overall logical structure.

On the one hand, XML is a subset of SGML, the international standard for the exchange of text documents, like HTML. On the other hand, XML belongs to the abstraction layer of SGML, i.e., it is a language to define specialized markup languages for different purposes. The main reason for the increasing popularity of XML is that all the various languages defined in XML can be parsed by a single standardized processor.

User-defined markup can structure the character data of a document and explain what they are through the use of *names*. Naming has to be considered very carefully. The ISO standard RM/ODP [ISO95a] states that a name is a term that refers to an entity in a given naming context. XML Namespaces are not appropriate. Instead, interoperable systems require access to a common sets of object semantics [BB00]. The need for shared ontologies has already been recognized in some industries where XML can help in building open systems, for example in electronic commerce.

Even if we would have a solution for the naming problem there is no standard behavior defined for the tags in an XML document. Specification of the behavior has to come from somewhere else. Some XML proponents say, the treatment of an element in an XML documents has to be supplied programmatically (with scripts) or declaratively (with style sheets). Many authors refer to presentation issues when discussing the behaviour of XML elements. This is where Extensible Stylesheet Language (XSL) comes in. It provides means to transform an XML document into a document with another logical structure. With XSL it becomes possible to produce different logical layouts needed by various users. On the other hand, it is very flexible in generating the presentation of a document by combining the power of XML with the idea of a style-property vocabulary. The notion of a "stylesheet" provides only one type of processing semantics (see CSS, XSL/Formatting Objects). But the term "stylesheet" reveals, that it is restricted to the presentation of document elements without considering the integrity of the stored information. XML processors can easily check the well-formedness of an XML docu-

ment and also the validity with respect to a given Document Type Definition (DTD). However, XML processors do not have an understanding of the document object semantics for which almost no predefined concepts are available in XML.

Document-Centric vs. Data-Centric XML Documents

It is one of the main characteristics of XML that it provides a syntax for serializing any kind of structured data that can be processed by standardized ubiquitous tools. The exchange of data among systems as a document would be just a special case of using XML as a universal media-independent publishing format. Therefore, we can distinguish between two types of documents with different requirements: *document-centric* and *data-centric* documents. Data-centric documents are documents that use XML for the data transport. Although XML is human-readable, data-centric documents are designed for machine consumption. Usually the data are only temporily stored as XML documents during the transport. Examples of such documents are sales orders, stock quotes, flight schedules. Data-centric documents are characterized by a fairly regular structure, fine-grained data, i.e. the elementary information unit are PCDATA elements or attributes, and mostly no mixed content. The order, in which sibling elements and PCDATA data occur, is generally not significant. Also prose-rich documents can be classified as data-centric, if the text has a highly regular structure with parts common to all documents. Document-centric documents are documents that are designed for a human reader, such as books, journal articles, emails. They are characterized by a less regular structure, coarse-grained data and lots of mixed content. The order, in which sibling elements and PCDATA occur, is almost always significant, particularly when the document is read serially by a human being.

Some documents can be considered hybrids of document-centric and data-centric documents. Looking at them in more detail, they can be specified as compositions of different types of documents, e.g. medical documents contain discrete pieces of data such as patient data, findings, prescriptions, procedures.

As we will see later, the distinction between data-centric and document-centric documents is a very simple requirements analysis with large impact on the implementation of a technical system that manages XML documents. Management refers to all issues of handling the information, for example store documents, retrieve whole documents, extract document parts, update documents.

3. XML - A Database Perspective

Round-Trip Problem

When storing XML document, there is one important requirement, called *round-tripping* [Bo00]. That is, an XML document is stored in a database (or somewhere else) and retrieved as the "same" document back again. This is important for XML applications that need to retrieve exactly the document with exactly the same layout which includes things in XML like CDATA sections, character entities, comments, and processing instructions. It is also vital to

many applications which are required by law to keep exact copies of documents. Round-tripping is less important to data-centric applications which care about the content of the document represented in elements, text, and attributes. The order of sibling elements or attributes may be important in some cases of data-centric applications. Take, for example, the position of items on sales orders: you typically list first the PC, followed by RAM, cables, peripherals, etc. However, as independent items in a database the order is immaterial.

Schema Definition

Document Type Definitions (DTDs) describe the structure of XML documents and are like a schema for them. A DTD specifies the structure of an XML element by specifying the names of its sub-elements and attributes. The sub-element structure is specified using the operators * (set with zero or more elements), + (set with one or more elements), ? (optional), and | (OR). All values are assumed to be string values, unless the type is ANY in which case the value can be an arbitrary XML fragment. There is a special attribute, ID, which can occur once for each element. ID uniquely identifies an element within a document and can be referenced through an (untyped) IDREF attribute in another element. XML Schemas are extensions to DTDs. One of the main differences between those and DTDs is that they allow typing of values and set size specifications (like cardinality constraints in associations). Although they provide some constructs such as hierarchy, sequence, choice, attributes, and opaque references, DTDs or XML Schemas are ill-suited to express the semantics of the *content* of the document elements. The reason is that they had been designed for serialization of data as a prerequisite for data exchange among systems. In that case the interacting systems are themselves responsible for the enforcement of the data integrity constraints that cannot be expressed with XML. If we want to store an XML document into a database we encounter all the problems caused by the semantic deficiencies of XML. The main issues and caveats are discussed subsequently.

From XML to Databases: Common Mapping Problems

Attributes vs. Element Text: Is the data stored in attributes or element text? Some authors recommend the extensive use of attributes in XML. They argue from an implementation point of view because using attributes instead of sub-elements reduces the number of nodes in the tree representation of the document which can be handled easier at the API (e.g., in DOM). Attributes can have a list of values and even a default value. On the other hand, attributes cannot be nested.

Meaning of Attributes: Without a semantic model, you may encounter some ambiguities regarding the interpretation of an attribute. Consider the example of a customer‘s order. Some components of an order could be expressed as sub-elements such as line items or customer number. Let’s assume we add an attribute to the order element `"expiry date 11/2001"`. Does that mean the order will expire in Nov. 2001 if the delivery is not possible until then? Or does it mean the information about the order can be thrown away in Nov 2001? Or is it just the expiry date of the credit card date, if another meaning to this name would be added? The ex-

ample underscores the need for an explicit information model because relying on "data names" may lead to serious problems.

Null Values: In the database world there is a concept of null values, different from a value of 0 (for numbers) or zero length (for a string). This has to be adequately expressed in an XML document considering the different semantics of different kinds of null values. XML supports null data through optional element types and attributes. In the XML Schema specification there is even a provision for null values in the sense of database null values. `xsl:null = 'true'` indicates that the element's text value is null, not the empty string. So far there is no concept of null for attributes. Following the standard requires to use element text for values that can be null. But this may conflict with other rules when to use attributes.

Comments, Processing Instructions: Most XML-to-relational mapping algorithms ignore some XML document components such as comments or processing instructions. Obviously, they are considered not to be content of the document.

Markup: Entities are placeholders for some piece of text or single characters. Thus, they are substituted in the physical presentation of the document. They are visible in a logical layout view as it is supported by XML editors. Among them markup characters that are not used for markup raise special problems. Take as an example the string `<foo/>` This is considered content and would be stored in exactly the same way. Query languages, such as SQL, cannot interpret column values as markup and don't understand entity usage. Therefore, a search for the string "<foo/>" would fail in a database that is not XML-aware because it does not consider the logical structure of the document.

Links: There are some constructs to express associations among document elements. They can be specified by links (XLink, XPointer) or by attribute values (ID: identifier value / IDREF: "foreign key" value). Their usage is up to the creator of the document. Hence, there can be more hidden associations in the document. For an association specified by ID/IDREF it is impossible to derive a non-ambiguous representation in a relational database. At first glance, you would map this to primary key/foreign key relationships. In this case the behavioral semantics would be added arbitrarily during the database design. Note that the link concept had been originally designed for documents and document fragments, e.g., XPointers can easily point to document subtrees using XPath. From a content point of view there may be some more associations and constraints among the data stored in the document. This applies particularly to data-centric XML applications. For them the XML link mechanisms are not adequate, and it would be more convenient to use another language to express the constraints (such as SQL DDL).

Ordered Composition: When mapping XML documents into databases, the problem arises how to deal with sibling orders. Data-centric applications don't require a strict order of the elements provided that they can be identified properly. Therefore, the effort to introduce artificial numbers for them in a database can be discarded. Sibling elements might by arbitrarily ordered, as there is no defined order among tuples of a relation. XML does not provide means to specify this aspect. The reason might be the implicit assumption that the whole text has to pre-

serve the order in which it is stored. Yet, this applies only to document-centric documents whose elements may be an ordered composition of sub-elements (cf. section 4, GRM).

Other Invariants: Take identity constraints as an example. With XML it is nearly impossible to specify that an attribute value must be unique across all objects of the same type within the document. The reason is that constraints can only be defined on the level of instances, not schemas. Collective state (invariants) and collective behavior cannot be directly handled in this manner, so that possibly some artificial objects ("performers") ought to be created. Therefore, a set of all concerned objects has to be constructed (e.g. using XPath expression), before the constraint can be enforced. Both the specification and the implementation of an identity constraints can be done better in a database environment. Identity constraints cannot be treated properly in XML, once the data is transformed into an XML document. They also represent a view on the document content that goes far beyond what XML was originally intended to do.

4. Using RM-ODP for Document Management

Viewpoints

In order to describe an XML document it has to be analyzed from different viewpoints, as it is good practice to separate concerns. An XML document can be discussed from three different viewpoints each representing relevant document properties to different users [KC95]:

- physical presentation view
- logical layout view
- content view

A *physical presentation* view considers the presentation of a document that can be very different, dependent on the media. A document can be seen as a composition of characters with some properties such as font, size, style. Regarding hypermedia documents also bitmaps can be components of a document. Any placeholders such as XML character entities have been resolved in the presentation view.

A *logical layout* view considers the logical layout. An XML document, like every other kind of document, can be logically structured. Thus, a document can be interpreted as a composition of prose components (paragraphs, sections, lists, list items) and other objects (e.g., frames, code sections). In case of document-centric documents this composition is mostly ordered for a human reader.
A document with a given logical layout has many possible physical presentation views that depend on the media, screen size, or paper size. Therefore, the logical layout plays an important role for the presentation of human-readable documents because the components can be presented in a uniform and consistent way. The idea can be found in any text processor, XML uses so-called Formatting Objects for the same purpose.

A *content view* considers intellectual content. In a content view a document can be interpreted as a composition of information objects suchs as title, author, abstract, body, bibliography. The content can be organized in a hierarchical structure as in "conventional" documents or can be flat as in relational databases.
The typical approach is to start with the document content and to map it into a logical layout that has to be transformed into a physical presentation. Today's XML systems usually do not support all three of the viewpoints and restrict themselves to the logical layout or the physical presentation.

RM-ODP defines five basic viewpoints of a system and its environment: enterprise viewpoint, information viewpoint, computational viewpoint, engineering viewpoint and technology viewpoint [ISO95a]. In this paper we argue mainly with the information viewpoint and the technology viewpoint. The information viewpoint focuses on the semantics of information and information processing. The information semantics can be specified by an invariant schema using the General Relationship Model (GRM) [KR94,ISO95]. The technology viewpoint focuses on the choice of technology for a system.

A viewpoint can be expanded into the specification of a new system at a different abstraction layer with the basic viewpoints. Therefore, we can look at the physical presentation view and describe an information model with some invariants for it. The model could be very simple, just a composition of characters. An invariant schema in the presentation view could specify character set, font information, or page size. The presentation medium can be specified in the technology viewpoint, e.g., paper, electronic file, screen. Each viewpoint has to be described in a viewpoint language. For example, the information viewpoint is typically described through data or object modelling, whereas the computational viewpoint may be specified by interaction diagrams. We apply the idea of viewpoints to understand the problems of dealing with content in XML document management as it will be discussed in the next section.

General Relationship Model (GRM)

The GRM can be used to describe relationships among things like documents and their elements. It provides some generic relationships with a well-defined behavioral semantics that apply everywhere (composition, subtyping, reference). Moreover, there exist other (non-generic) relationships that are defined in the same manner, for example the "realization" relationship [Kil99].

We have to focus on the composition relationship that can be encountered very often in document modeling. According to RM-ODP, a composition of objects is defined as a "combination of two or more objects yielding a new object at a different abstraction level of abstraction. The characteristics of the new objects are determined by the objects being combined and by the way they are combined" [ISO95a]. Besides the composition the reference relationship is needed. It means that the referencing object determines properties of the referenced object that can exist independently from the referencing object.

5. Separating Concerns in the Specification of XML Documents

Content View vs. Logical Layout

First, we have to understand the boundaries between content and logical layout which is a prerequisite for the lossless storage of XML documents in a database. A document reference model based on RM-ODP is described in [KC95]. Accordingly, a document is a composition of document content elements, e.g., abstract, body, legal disclaimers. The document reference model proposes an extendible set of subtypes of document content elements. XML is an acceptable language for representing document-centric documents in the information viewpoint of the content view. A user can define the necessary content elements in a DTD or in an XML Schema. These languages provide some restricted constructs to express structural constraints. The behavioral semantics is implicitly defined. For example, XML Schema enables to define cardinality constraints for the composition of a content element from other elements. The document becomes invalid by any violation of constraints defined in the schema. There are some implicit assumptions about content elements: they occur once only within a document - except from references- provided that they correspond with some large-grained information unit (e.g., paragraph section, picture). There is only a small amount of data that is common to all documents such as copyright notices, corporate addresses, or product logos. This amount is so small in relation to the total that some redundany in the content view can be tolerated.

The content view of data-centric documents differs significantly from document-centric documents with respect to the invariants that have to be specified because there may be large overlaps among documents or content elements. Take as an example an order system of a web shop that could be used - with minimal changes - for a bricks-and-mortar store as well. A sales order consists of header information, such as order number, order date, customer number, and one or more line items that contain a part number, quantity, and price. Because of the hierarchical approach inherent to XML, the header information can be stored in a single document as a parent element with multiple fixed children. If the header information is extended by customer data, such as name and address, then we have the problem to duplicate the customer information in each sales order. The same question applies to the many-to-many relationship between orders and parts in our example. This relationship can be referred to as a composition: non-hierarchical, assembly, non-ordered. In our application, there may exist more user-defined integrity constraints that correspond to invariants, for example:

- the overall value of an order must exceed a certain minimum
- a customer can submit at most five orders
- if a customer is deleted all of his orders have to be cancelled

As stated above, XML provides a small set of simple integrity constructs in XML schema to specify validity that can be checked by an XML parser. From an XML point of view it does not make sense to provide further means to specify more complex constraints because only a database engine could efficiently deal with them. Therefore, we should choose another language or tool for the specification of the information viewpoint for the content of data-centric documents. In data-centric documents there are lots of choices how to map the content model

to a logical layout. There are different choices how the resulting logical layout of the document could look like, similar to database views. Some examples are:

- list of customers each with a list of orders each with a list of items
- list of orders each with customer data in the header and a list of items
- list of items each with a list of assigned orders having the customer information

There is an alternative how to specify the association between customers and orders in the information viewpoint of the content view: The customer information would be stored separately and referenced by an XLink in the sales order document. With the link approach we would use an implementation concept for associations between document elements - analogous to pointers [Kil94]. We should avoid this to prevent a confusion of different basic viewpoints within the content view.
In a logical layout view the use of XLinks or XPointers is useful for the handling of common layout elements, such as, e.g., corporate logos, in document-centric documents.

Specification of Operations

Users in each view have different ways to manipulate the information contained in the document. Each viewpoint provides operations that are specific for it. Because XML does not explicitly use the three different document views it provides operations and languages that mix different viewpoints. For example, XPath provides all types of retrieval operations: value-based, structure-based, text-oriented, and metadata search. At first glance, it looks very powerful but the underlying document tree model makes set-oriented queries cumbersome. The same applies to other declarative query languages such as XQuery or XML-QL. It is hard to express invariants in the information viewpoint which has to be done at the level of types, not instances. There is some ongoing effort to specify an update language, XUpdate, that is based on XPath und inherits its weaknesses with respect to invariants because it has been designed from a different viewpoint.

The Document Object Model (DOM) provides a standard model and an application programming interface for XML documents. As a platform and language independent interface it enables to retrieve content, structure and metadata of a document and to update them. Because there is no explicit boundary between logical layout view and the content view in XML the DOM provides operations of both viewpoints on the level of instances, for example insert data, remove a child node. Whereas the handling of document-centric documents with a tree-based logical structure is supported by the DOM interface, the requirements of data-centric documents are not considered. In order to specify the information viewpoint within the content view, another language is needed. Set-oriented operations - as they are known in a relational data model - and complex constraints cannot be specified with the DOM interface. The same applies to content-based retrieval such as full-text search operations.

6. Realization of XML Document Management

Consider the implementation of a management system for XML documents. This can be seen as a realization relationship between a "source" activity (e.g., specification) and a "target" activity (e.g., design) [Kil99]. As already mentioned, we can apply the basic five viewpoints within each viewpoint of the document reference model. This can help us to understand the problems we have encountered in the past regarding the lossless storage of XML documents in a database.

Everything that can be defined in XML is both specification and realization. Compared to conventional database or software design, there is nothing for XML documents, apart from editors. XML does not provide an abstraction that would create a new semantic level in which one can be absolutely precise. Thus, the specification of XML documents is like programming without analysis.

The realization of the storage function for XML documents can differ significantly in all three viewpoints. The content view of a data-centric document can better be stored in a relational DBMS, because it provides SQL constructs to express semantic data constraints in the document (for examples see [FK99, STH+99]). There can be different logical layout views of data-centric documents. They may be stored separately as XML template files and combined with the content via placeholders (e.g., *rowset* markup). Different logical layouts can simply be produced using XSLT. There is one risk with respect to the effort of the W3C to enhance XML Schema and XML query languages. If XML Schema would provide all primitives to define what things are, how they are related and how to deal with them, then it would result in a full-fledged data model as it is defined in a DBMS. A DBMS has the knowledge to manage the data accordingly and provides all data-oriented functionality, such as integrity enforcement, data manipulation, retrieval, optimization. A well-established example how to separate concerns is to centralize these functions in a DBMS away from file-processing applications. Without considering this, we would re-invent DBMS functionality - possibly on top of a DBMS - by developing software modules, viz. XML processors, that deal with XML documents on behalf of other software module. All semantics would be back in application programs built by using XML technology. Not to mention the DBMS know-how regarding the optimization and physical data organization. That’s why we must keep the borderlines clear among different document views. XML processors can deal with the logical layout view or provide tools for the presentation, as for example XML-to-HTML transformation, but they should not focus on the content.

Native XML database systems are a better choice for document-centric documents from the technology viewpoint because they can enforce XML-specific constraints in a more natural way than traditional DBMS systems [SAG00]. In that case, XML is used as the language to describe the information viewpoint of the document as a composition of sub-elements (e.g., article, book).

7. Conclusions

Generic information modelling concepts promote the understanding of XML document management. The current XML technology provides lots of tools and languages, but there is almost no guidance for a precise semantic specification of the content or the logical structure of the document. New standards such as XML Schema or XLink aggravate these problems because they increase the number of syntactic alternatives how to specify the semantics of the XML document. Our conclusions can be seen as a roadmap how to proceed when building an XML system with databases.

- **Analyze the requirements first before building an XML system**

XML documents have different characteristics and can simply be classified into two categories, viz. data oriented and document-oriented. Each of these categories implies different business requirements to the XML system. The requirements comprise the functionality an XML system has to provide, i.e., queries, updates, import and export functions. There may be a broad range of queries from full-text search to set-oriented queries as in SQL. This has a huge impact on the choice of the technology that may be a combination of several platforms (for example: relational DBMS plus file system).

- **Think in viewpoints to understand the semantics**

An XML document can be discussed from three contextual views as they are already known for many years. Usually, they can be found in combination because XML languages do not really enforce a separation of concerns. Our examples have shown the mixed occurrence of the content view and the logical structure in an XML document. This has to be understood before the document will be stored in some other system. Therefore, a viewpoint can be expanded into the specification of a new system at a different abstraction layer with the complete range of the basic five RM/ODP viewpoints. This idea was very helpful to understand the problem of integrity maintenance in XML documents because there is a different set of invariants regarding the logical structure and the document content. The information viewpoint on the content view may require a different language to express the invariant schema in comparison with the information viewpoint of the logical layout view. This applies in particular to data-centric documents with "hidden" data semantics.

- **Use generic relationships for constraint modelling**

XML provides both a language and a restricted set of concepts to express constraints. Users who try to specify all invariants in an information viewpoint should not stick to the XML offerings. Instead they should use GRM concepts representing relationship patterns that always exist despite of the XML capabilities. Among them are all subtypes of composition and the reference relationship. Some cases have been discussed in the paper, for example, the different composition of document elements dependent on the document type and the contextual view. The generic relationships can be used in the information viewpoint throughout all document views provided they are clearly separated.

- **Beware of the difference between specification and realization**

The current XML standard does not consider the difference between specification and realization. The realization pattern can be encountered repeatedly because an XML database system consists of several design layers: XML document layer, logical database design (according to the mapping algorithm), physical database design. These layers have to be separated clearly in order to fulfil other non-functional requirements like storage und location transparency.

Acknowledgments

My thanks are due to Haim Kilov and Alex Buchmann who gave constructive comments. The work was supported by the Saxonian Department of Science and Art (Sächsisches Ministerium für Wissenschaft und Kunst) through the HWP program.

References

[BB00] C. Bornhövd, A. P. Buchmann: *Semantically Meaningful Data Exchange in Loosely Coupled Environments,* 6th International Conference on Information Systems Analysis and Synthesis, ISAS'00, Orlando, Fl., 2000.

[Bo01] R. Bourret: *XML and Databases*, http://www.rpbourret.com/xml/XMLAndDatabases.html

[FK99] D. Florescu, D. Kossmann: *Storing and Querying XML Data using an RDBMS.* Data Engineering, Sept. 1999, Vol.22, No.3.

[GP00] C.E. Goldfarb, P. Prescod: *The XML Handbook*, Addison Wesley, 2000.

[ISO95a] ISO/IEC JTC1/SC21: *Open Distributed Processing - Reference Model -Part 2: Foundations,*, IS 10746-2/ITU-T Recommendation X.902, 1995.

[ISO95b] ISO/IEC JTC1/SC21: *Information Technology. Open Systems Interconnection - Management Information Services - Structure of Management Information - Part 7: General Relationship Model*, 1995 (ISO/IEC 10165-7.2).

[KC95] H. Kilov, L. Cuthbert: *A model for document management*, Computer Communications, Vol. 18, No. 6, Elsevier Science B.V., 1995.

[Kil94] H. Kilov: *On Understanding hypertext: are links essential?*, ACM Software Engineering Notes, Vol. 19, No. 1, Jan. 1994.

[Kil99] H. Kilov: *Business Specifications - The Key to Successful Software Engineering*, Prentice Hall, 1999.

[KR94] H. Kilov, J. Ross: *Information Modeling: an Object-Oriented Approach*, Prentice Hall, 1994.

[SAG01] Software AG: *Tamino XML Database,* http://www.softwareag.com/tamino, 2001.

[STH+99] J. Shanmugasundaram et. al: *Relational Databases for Querying XML Documents: Limitations and Opportunities*., Proc. 25th VLDB conference, 1999.