Technical Report, No. TR-2002301, Dept. of Computer Science and Business Informatics,
University of Vienna, March, 2002

A Typed DOM for the Management of MPEG-7 Media Descriptions

Utz Westermann, Wolfgang Klas

Multimedia Information Systems

Department of Computer Science and Business Informatics

University of Vienna, Austria

{gerd-utz.westermann,wolfgang.klas}@univie.ac.at

Abstract

MPEG-7 is a very promising standard for the description of multimedia content. Certainly, means for the adequate management of large amounts of MPEG-7 media descriptions are needed in the near future. Essentially, MPEG-7 media descriptions are XML documents following media description schemes defined with an extension of XML Schema named MPEG-7 DDL. However, XML database solutions available today are not suitable for the management of MPEG-7 media descriptions. They typically neglect the type information available with media description schemes and represent the basic contents of media descriptions as text. But storing non-textual multimedia data typically contained in media descriptions such as melody contours and object shapes textually and forcing applications to access and process such data as text is neither adequate nor efficient. In this paper, we therefore propose the Typed Document Object Model (TDOM), a data model for XML documents that can benefit from available schema definitions and represent the basic contents of a document in a typed fashion. Through these typed representations, applications can access and work with multimedia data contained in MPEG-7 media descriptions in way that is appropriate to the particular type of the data. Thereby, TDOM constitutes a solid foundation for an XML database solution enabling the adequate management of MPEG-7 media descriptions.

1 Introduction

Recently, there have been considerable efforts to standardize the description of multimedia content. This has resulted in a variety of standards, such as the Dublin Core Metadata Element Set [10], Learning Object Metadata [21], VRA Core Categories [47], and the Multimedia Content Description Interface (MPEG-7) [24, 23, 36, 37]. Being an ISO standardization effort which is backed by prominent broadcasting companies, consumer electronics manufacturers, and telecommunication service providers and which has reached a mature state by the end of 2001, MPEG-7 receives considerable attention in the multimedia community. What makes MPEG-7 particularly attractive is that it is targeted at the description of multimedia content on a technical, feature-oriented level as well as on a semantic level. For instance, it is not only possible to describe the frequency spectrum of a song recording in an MPEG-7 media description. It is also possible to refer to the lyrics and the musical score, all within the same description.

The scope of standardization basically comprises two parts: a Description Definition Language (MPEG-7 DDL) [25] with which schemes for the description of media can be specified, and, defined via MPEG-7 DDL, a comprehensive set of media description schemes[1] that are useful for a variety of applications. The media description schemes standardized include schemes for visual media [26] and audible media [27] as well as schemes of general use [28]. Applications are not limited to these standardized media description schemes: new description schemes can defined with MPEG-7 DDL, either from scratch, or by extending or combining existing description schemes.

By the diversity of aspects with which content can be described and by the extensibility of the standard with new description schemes, MPEG-7 is expected to face wide-spread use

[1]The terminology of MPEG-7 originally distinguishes between descriptors, which are basic descriptive measures for media, and description schemes, which hierarchically combine descriptors and other description schemes to more complex descriptional units. However, descriptors do not just constitute simple, atomic measures but can also bear a complex structure. As they are furthermore specified in the same way as description schemes with MPEG-7 DDL, the distinction between both terms seems rather arbitrary. For the sake of simplicity, we therefore just talk about media description schemes in this paper but mean description schemes and descriptors at the same time.

in a broad range of applications: media archives, journalism, education, entertainment, etc. Therefore, means for the effective management of large amounts of MPEG-7 media descriptions are certainly needed.

Basically, MPEG-7 media descriptions are XML documents that are valid to a media description scheme expressed in MPEG-7 DDL. Thus, it seems natural to employ XML database solutions for the management of MPEG-7 media descriptions. Closer examination of current XML database solutions for their suitability for MPEG-7, however, reveals difficulties. One of the main problems with current solutions is that they typically represent the basic contents of XML documents, i.e., simple content of elements and the content of attribute values, as text. But textually representing multimedia data such as melody contours and object shapes often contained in MPEG-7 media descriptions is inadequate: textual representations of non-textual data consume unnecessary storage space, do not preserve the meaning of the data (e.g., with respect to indexing), and are inefficient and cumbersome to handle such that applications are forced to constantly translate the textual representations to data structures better suiting the particular data type at the cost of considerable processing power. Clearly, more adequate database solutions are required for MPEG-7.

In this respect, we make several substantial contributions in this paper: we motivate and present several basic but nevertheless essential requirements for the management of MPEG-7 media descriptions. Along these requirements, we analyze existing XML database solutions – native XML database solutions as well as extensions for relational systems, commercial systems as well as research prototypes – fortifying the demand for more suitable MPEG-7 database solutions. As a solid foundation for such a solution, we then propose the Typed Document Object Model (TDOM). TDOM is an object-oriented data model for XML documents specifically designed bearing the requirements for the management of MPEG-7 media descriptions in mind. The model's outstanding feature is that type information contained in media description schemes written in MPEG-7 DDL can be exploited to represent the basic contents of an XML document in a typed fashion and not just as text. In typed representation, simple element content and the content of attribute values is kept in data structures that are appropriate for the the respective content type and that come with type-specific operations to reasonably work with the content. Thereby, applications can process the multimedia data contained in MPEG-7

media descriptions more adequately and efficiently.

The remainder of the paper is organized as follows: Section 2 derives essential requirements for the adequate management of MPEG-7 media descriptions. Section 3 evaluates existing XML database solutions according to these requirements. Section 4 introduces and gives a thorough definition of TDOM. Section 5 concludes the paper with a summary and an outlook to current and future work.

2 Requirements for the management of MPEG-7 media descriptions

In this section, we briefly illustrate the nature of MPEG-7 media descriptions (2.1). We then derive a set of fundamental requirements for the management of such descriptions (2.2).

2.1 MPEG-7 media descriptions

MPEG-7 is strongly committed to XML and related standards. MPEG-7 DDL, the language used for the definition of media description schemes, is a superset of XML Schema [45, 2], a schema definition language for XML documents recently standardized by the W3C. Certain extensions to XML Schema were considered neccessary to better cope with the peculiarities of multimedia data. In particular, support for array and matrix data types as well as additional data types for time points and time durations were added to XML Schema. Regarded as an extended XML Schema, MPEG-7 DDL is thus just another schema definition language for XML documents. Since media description schemes are defined with MPEG-7 DDL, an MPEG-7 media description complying to a given media description scheme is consequently an XML document that is valid with respect to the schema definition given by the description scheme.

We would like to illustrate the concepts of media description schemes and media descriptions with an example. The MPEG-7 `Melody` media description scheme [27] is a representative description scheme that can serve as the basis for the realization of query-by-humming applications. A slightly simplified version of this description scheme expressed in MPEG-7 DDL is shown in Figure 1.

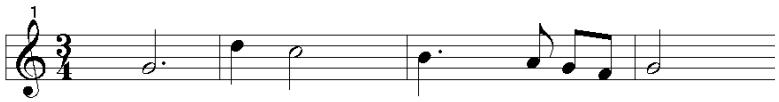
```
<schema xmlns="http://www.w3.org/2001/..."
  xmlns:mpeg7="http://www.mpeg7.org/2001/..."
  targetNamespace="http://www.mpeg7.org/2001/...">

<complexType name="MelodyType">
  <sequence>
    <element name="Meter"
      type="mpeg7:MeterType"
      minOccurs="0"/>
     <element name="MelodyContour"
      type="mpeg7:MelodyContourType"
      minOccurs="0"/>
  </sequence>
  <attribute name="id" type="ID"
    use="optional"/>
</complexType>

<complexType name="MelodyContourType">
  <sequence>
    <element name="Contour">
      <simpleType>
        <list itemType="integer"/>
      </simpleType>
    </element>
    <element name="Beat">
      <simpleType>
        <list itemType="integer"/>
      </simpleType>
    </element>
  </sequence>
</complexType>
```

```
<complexType name="MeterType">
  <sequence>
    <element name="Numerator">
      <simpleType>
        <restriction base="integer">
          <minInclusive value="1"/>
          <maxInclusive value="128"/>
        </restriction>
      </simpleType>
    </element>
    <element name="Denominator">
      <simpleType>
        <restriction base="integer">
          <enumeration value="1"/>
          <enumeration value="2"/>
          <enumeration value="4"/>
          <enumeration value="8"/>
          <enumeration value="16"/>
          <enumeration value="32"/>
          <enumeration value="64"/>
          <enumeration value="128"/>
        </restriction>
      </simpleType>
    </element>
  </sequence>
</complexType>

<element name="Melody"
  type="mpeg7:MelodyType"/>

</schema>
```

Figure 1: Simplified MPEG-7 `Melody` media description scheme

According to the media description scheme depicted, the melody of a song (see the declaration of the element type `Melody` in the lower right column) can be described by its meter and melody contour (element types `Meter` and `MelodyContour` declared in the complex type `MelodyType` in the left column). The meter of a melody is a fraction number consisting of a numerator and denominator (see element types `Numerator` and `Denominator` declared in the complex type `MeterType` in the right column). There is the restriction that the numerator must be an integer value in the interval from 1 to 128, while the denominator must be a power of two in the same interval. The melody contour consists of a contour and a beat (see element types `Contour` and `Beat` declared in the complex type `MelodyContourType` in the lower left column). The contour of a melody is a list of integer values giving a measure for the distance between every two consecutive notes of the melody while the beat is a list of integer values associating every note of the melody to its position in the beat.

A media description complying to the `Melody` media description scheme is an XML document that is valid to the schema definition presented. Figure 2 gives an example of such a



```
<!-- Melody description of 8 notes taken from “Moon River” by Henry Mancini -->

<Melody xmlns=“http://www.mpeg7.org/2001/MPEG-7_Schema”
        id=“moonRiver”>
        <Meter>
                <Numerator>3</Numerator>
                <Denominator>4</Denominator>
        </Meter>
        <MelodyContour>
                <!-- Distance between two notes -->

                <Contour>2 -1 -1 -1 -1 -1 1</Contour>

                <!-- Beat position of notes -->

                <Beat>1 4 5 7 8 9 9 10</Beat>
        </MelodyContour>
</Melody>
```

Figure 2: Example of an MPEG-7 media description

document describing a small fraction of the melody of the song “Moon River” by Henry Mancini (taken from [27], page 101).

There are some general observations that can be made on MPEG-7 media descriptions:

- MPEG-7 media descriptions are XML documents.
- MPEG-7 media descriptions comply to media description schemes expressed in MPEG-7 DDL, a schema definition language for XML documents.
- The set of available media description schemes is not fixed by MPEG-7. The standard ships with a multitude of predefined schemes such as our example `Melody` media description scheme but applications may create new description schemes with MPEG-7 DDL if desired.
- Much of the information encoded in XML documents that constitute MPEG-7 media descriptions is not of a textual nature. Large portions of the information consist of numbers and mathematical structures such as lists, vectors, and matrices – usually to describe rather technical aspects of media content. As a matter of fact, about 84% of the

media description schemes predefined by the standard for visual and audible content in [26] and [27] consist primarily of non-textual data.

2.2 Requirements

As we have observed, MPEG-7 media descriptions are XML documents. Thus, the problem of adequately managing MPEG-7 media descriptions can be curtailed to the problem of managing XML documents in principal. In literature, general requirements for XML database solutions have already been identified [38]. Among the desired features of an XML database solution are rich document modeling capabilities, the availability of a query language, support for document updates, the availability of index structures, the management of access rights, support for transactions as well as backup and recovery.

In the following, however, we want to specifically look onto the management of XML documents from the perspective of MPEG-7: we motivate and present four very basic but nevertheless critical requirements for the effective management of MPEG-7 media descriptions. Namely, the requirements are fine-grained representation of description structure, typed representation of description content, support for updates, and support for MPEG-7 DDL. As it will turn out in Section 3, current XML database solutions fail in fulfilling all of these basic requirements.

Fine-grained representation of description structure. With MPEG-7 DDL, schemes of arbitrary complexity for the description of media content can be defined, describing media content from possibly very different points of view. However, not every application working with media descriptions conforming to a complex description scheme can be expected to process the full scope of a description. Rather, applications will access only those parts of a description that are necessary to fulfil their particular tasks.

Enabling fine-grained access to the constituents of a media description is therefore essential for the adequate management of MPEG-7 media descriptions. This calls for the fine-grained representation of the structure of the media description. With a faithful reproduction of the hierarchy of the various nodes, i.e., markups, of which the description consists, applications can access exactly those parts that they are interested in. In contrast, if the description was represented as a single unstructured object, an application would always have to access the

complete description and decompose it into its constituents – even if the application is interested in just a small fraction.

Typed representation of description content. As we have already noticed by the means of the `Melody` media description scheme of Figure 1 as a typical representative of many description schemes predefined by MPEG-7, much of the information encoded in media descriptions consists of non-textual data like numbers and rather complex structures like lists. Since MPEG-7 media descriptions are XML documents and, as such, a form of text documents, these data are encoded as text.

This might be adequate for the platform-independent exchange of media descriptions. It is doubtful for several reasons, however, whether textual representation of non-textual data is also reasonable for the management of media descriptions within a database: textual representations of non-textual data typically consume more storage space than equivalent binary representations. Moreover, they are less efficient and cumbersome to handle. A good example for this point is the list of integer values being the content of the `<Contour>` element in the media description of Figure 2. The effort necessary to retrieve the 4th element of the list on the basis of the list's textual representation, i.e., through string operations, is significantly higher than the effort necessary for the same action on the basis of an adequate data structure, e.g., an array. Therefore, applications must usually translate textual representations of non-textual data to internal data structures more appropriate to the particular type of data before they can adequately work with the data – at the cost of considerable processing power. Finally, textual encoding of non-textual data does not necessarily preserve the semantics, e.g., with respect to ordering. For instance, the alphanumeric order of the textual representations of integer values differs from their numeric order hindering reasonable indexing.

Given these problems, a suitable database solution should represent the basic contents of an MPEG-7 media description – namely, simple content of elements and content of attribute values – in a typed fashion and not as text. With typed representation, we mean that these contents are kept in data structures that are adequate to the particular content type and come with type-specific operations to reasonably work with the content. To that end, a database solution should not only support the whole lot of simple data types predefined with MPEG-7 DDL

[25, 2] but also the variety of derivation methods for simple types coming with the standard – MPEG-7 DDL allows to flexibly derive new simple types from existing ones in a schema definition. Examples of such derived simple types are the list type and the range-restricted integer type defining the allowed contents of the `Contour` and `Numerator` element types in Figure 1: both are based on the predefined type integer.

Fine-grained updates. It is unrealistic to assume that MPEG-7 media descriptions are produced in one shot and then never touched again. Just like the media content they describe, media descriptions evolve and are constantly subject to change during the different phases of the content's lifecycle. Acknowledging that media descriptions are subject to change, we demand that a database solution should offer adequate means for updating media descriptions. Just like it is necessary to offer applications fine-grained access to the nodes of a possibly complex media description, applications should be allowed to perform fine-grained updates on any part of the description – having to unload the complete description from the database, to modify it outside the database, and to reinsert it into the database just to update a small fraction is definitely not efficient and prevents concurrent access.

Support for MPEG-7 DDL. For the suitable management of MPEG-7 media descriptions, it is of advantage to be capable of processing media description schemes expressed in MPEG-7 DDL. The exploitation of the information available in these schema definitions is on the one hand required to ensure the consistency of the database contents by validating whether an XML document constitutes a correct media description with respect to a given media description scheme. Typical occasions where such a validation is reasonable are during the import of a media description into a database and during the update of a media description to decide whether an update operation can be permitted without violating the description scheme. On the other hand, processing of type information contained in schema definitions is necessary to be able to infer the types and, by these types, appropriately typed representations of the basic contents of a media description.

One might argue that the use of media description schemes to type the basic contents of a media description could prohibitively increase the complexity of inserting new media descriptions into a database and the complexity of database updates such that the benefits of typed

representations are overweighed by performance problems. It is our experience, however, that this does not constitute a problem for most applications. If one admits that it is a vital task of a database solution to ensure the consistency of database contents, an MPEG-7 database solution must take the effort anyway to validate a media description against its associated media description scheme whenever the description is newly inserted into a database or updated.

During validation, the database solution already has to verify whether the textual representation of the simple content of an element or the content of an attribute value matches the type for the content as declared in the media description scheme. The additional effort necessary to bring the content into an appropriate typed representation after a successful verification is likely to be negligible for most applications, except for ones with very high update and insert frequencies and with harsh real-time constraints.

But for such applications, a database solution will already have to abstain from the validation of media descriptions after insertions and updates to noticeably save processing time – a time saving, the applications will presumably pay for later when they must ensure the consistency of media descriptions themselves and bring the basic contents of a description from their textual representations to internal ones more suitable for further processing whenever they access the description.

3 Analysis of XML database solutions

On the basis of the requirements elaborated in the previous section, we have examined current XML database solutions with regard to their suitability for managing MPEG-7 media descriptions. Figure 3 gives a summary of our analysis, showing whether or to what extent each of the solutions meets each of our requirements.

In the figure, two coarse categories of database solutions for XML documents are distinguished: native database solutions specifically designed for the storage of XML documents and solutions based on relational database management systems (DBMS). Both categories are further subdivided into products – commercial ones as well as open source projects – and research prototypes. In the following, we briefly discuss the results of Figure 3 for native XML database solutions (3.1) and for relational XML database solutions (3.2) before concluding this section

		Fine-grained Representation of Description Structure	Typed Representation of Description Content	Fine-grained Updates	Support for MPEG-7 DDL
Native Products	**dbXML [43]**	+	-	+	-
	eXcelon XIS [12]	+	Textual representation - interpretation as string or number can be explicitly specified for indexing purposes	+	Limited XML Schema support
	GoXML DB [50]	+	Though claimed [50] apparently not realized in Version 2.0.2	+	Limited XML Schema support
	Infonyte-DB [22]	+	-	+	-
	Tamino [42]	-	Textual representation - interpretation as string, real, or integer can be explicitly specified for indexing purposes	-	Limited XML Schema support
	TEXTML [29]	-	Textual representation - interpretation as string, date, or number can be explicitly specified for indexing purposes	-	-
	X-Hive/DB [48]	+	-	+	-
	Xindice [44]	+	-	+	-
Native Research	**Lore [17]**	+	Limited set of elementary simple types supported for attribute values only	+	-
	Natix [31]	+	Limited set of elementary simple types supported	+	-
	PDOM [19]	+	-	+	-
Relational Products	**IBM DB2 XML Extender [20]**	-	-	-	-
	Microsoft SQLXML [34]	-	-	-	-
	Oracle XML DB [18]	-	-	-	-
Relational Research	**Monet XML [39]**	+	-	+	-
	Shimura et al. [41]	+	-	+	-
	XML Cartridge [16]	+	-	+	-

Figure 3: Analysis of current XML database solutions (+ support, - no support, comment in case of partial support)

with a summary (3.3).

3.1 Native XML database solutions

Taking a look at Figure 3, we can see that the analyzed native XML solutions have strong deficiencies with respect to the typed representation of the basic contents of a media description: most of the native solutions investigated represent simple content of elements and the content of attribute values of XML documents as text – with all the incurring problems with regard to the appropriate processing of non-textual data often contained in MPEG-7 media descriptions. eXcelon XIS [12], TEXTML [29], and Tamino [42] somewhat alleviate the lack of typed representations by allowing to manually specify whether the content of an element or attribute value is to be interpreted as a string, number, or date for indexing purposes. Apart from the fact that support for strings, numbers, and dates does not come even close to the broad variety of elementary simple types and simple type derivation methods available with MPEG-7 DDL, the content still remains represented as text.

To some extent, three of the examined native XML database solutions, the commercial GoXML DB [50] and the research prototypes Lore [17] and Natix [31], address the issue of typed representations. GoXML DB claims that the basic contents of an XML document are represented in a typed fashion. Experiments with the current Version 2.0.2, however, have revealed that this feature has apparently not been implemented yet. Content of attribute values and simple content of elements is represented and interpreted as text. Compared to that, Lore and Natix offer some elementary simple types for the typed representation of basic document contents. However, both solutions support just small subsets of the simple types predefined by MPEG-7 DDL. Simple type derivation methods, such as lists and matrices, are not supported at all. Furthermore, Lore limits the use of typed representations to the content attribute values; simple element content remains represented as text.

Even if the native solutions sufficiently supported the typed representation of the basic contents of an XML document, they would not be able to infer adequate typed representations for the contents of an MPEG-7 media description: no solution investigated can fully process MPEG-7 DDL schema definitions. In fact, most systems do not make use of schema definitions at all for XML document storage. Just eXcelon XIS, Tamino, and GoXML DB support more or less limited subsets of XML Schema for the purpose of document validation not reaching the expressive power of MPEG-7 DDL.

3.2 Relational XML database solutions

Three principal approaches to storing XML documents in a relational DBMS can be distinguished. In the first approach, XML documents are directly stored in their textual format in a character large object (CLOB). Special stored procedures are provided to access the contents of a document from SQL, e.g., via XPath expressions [7]. Most XML database extensions of the major relational DBMS vendors support this approach, such as Oracle XML DB [18], IBM DB2 XML Extender [20], and Microsoft SQLXML [34]. With respect to the management of MPEG-7 media descriptions, however, it is obvious from Figure 3 that this approach is not adequate. As documents are stored as a whole in their textual format in a CLOB, there is neither a fine-grained representation of the structure of a media description nor a typed representation of the basic contents of a media description. Updates are only possible by replacing complete documents and any existing schema definitions are not exploited for the storage of XML documents.

The second approach to the management of XML documents in a relational DBMS keeps the nodes of a document and the hierarchical relationships between them in tables. There has been considerable research in this area (see [14] for an overview) and a lot of research prototypes have been developed. For our analysis, we have focused on three representative systems: Monet XML [39], Shimura et al. [41], and the XML Cartridge [16]. These systems have in common that they represent the structure of XML documents with a fine granularity and that they also allow fine-grained updates. Nevertheless, they are not suitable for the management of MPEG-7 media descriptions, since they do not make use of schema definitions to represent the basic contents a document in a typed manner but rather represent all contents as text.

The third approach to the management of XML documents in a relational DBMS maps the data conveyed in XML documents to application-specific database schemas. Even though this would in principal place all modeling capabilities available with the relational DBMS at the disposal for representing document content, we have decided to ignore this approach for the management of MPEG-7 media descriptions. The definition of an application-specific database schema as well as the specification of the mapping between an XML format to this schema are elaborate manual tasks. Bearing in mind that MPEG-7 allows to define arbitrary description schemes in excess to those predefined with the standard, the effort necessary to cope with

a media description following a previously unknown media description scheme is prohibitive. In literature, there are some approaches for the automatic derivation of relational database schemas from schema definitions for XML documents [40, 46, 11] and the automatic mapping between them. However, these are based on Document Type Definitions and whether they scale to the far more complex MPEG-7 DDL remains to be proven.

3.3 Summary

As we have seen, none of the XML database solutions examined, native as well as relational, is adequate for the management of MPEG-7 media descriptions. Their main limitations are the lack of typed representations for the basic contents of an XML document and the inability to process schema definitions expressed in MPEG-7 DDL to derive such typed representations. For the suitable management of MPEG-7 media descriptions, we therefore certainly see a need for a solution that focuses on exactly these points.

4 The Typed Document Object Model

As we have seen in Section 3, database solutions more suitable for the management of MPEG-7 media descriptions are needed. The heart of such a solution – just as with any other database solution – is a data model. The purpose of this data model is to provide a detailed, accurate, and adequate representation of the structure and contents of MPEG-7 media descriptions at a logical level. On the basis of such a logical representation, applications can gain access to the contents of media descriptions and process them more appropriately compared to the alternative of constantly working on the original textual format through parsing operations.

Furthermore, the data model can serve as the foundation for the physical storage scheme implemented by an MPEG-7 database solution. Closely orienting the physical storage scheme of MPEG-7 media descriptions in a database along the data model has the advantage that exactly those parts of a media description can be loaded from a database that are actually accessed during the processing of a description on the basis of the data model thereby minimizing I/O operations and main memory consumption. Storing media descriptions in their original textual format – which requires to constantly load the entire description into main memory whenever

it is accessed in order to parse it and bring it into the data model representation for further processing – is not very attractive as we could observe with relational XML database solutions like Oracle XML DB and IBM DB2 XML Extender in Section 3.

Since MPEG-7 media descriptions are XML documents, a suitable data model should be a model for XML documents that considers the basic requirements for the management of these descriptions that we have presented earlier in Section 2. In this section, we propose the Typed Document Object Model (TDOM), a conceptual object-oriented model for XML documents that we have created bearing exactly these requirements in mind. We begin by taking a brief look onto existing data models for XML documents unveiling their limitations concerning the representation of MPEG-7 media descriptions (4.1). Having thus fortified the need for a new data model, we then illustrate and give a thorough definition of TDOM (4.2).

4.1 Data models for XML documents

A variety of data models for XML documents have been proposed in literature, e.g., [30, 17, 16, 41, 39, 31]. Concerning their application for the representation of MPEG-7 media descriptions, however, these models suffer from mainly two weaknesses: firstly, they typically constitute variations of rather simple edge-labeled tree and graph data models. Though they provide fine-grained representations of the MPEG-7 media descriptions in principle, these models often ignore more subtle aspects of the descriptions' structure like the ordering the child nodes of an element, markup different from elements and attribute values such as processing instructions and comments, or the distinction between attribute values and elements. Secondly, they usually do not support typed representations: simple element content and the content of attribute values is typically represented as text hindering the reasonable processing of non-textual data on the basis of these models. Those few models that support typed representations (e.g., [31, 17]) only offer limited subsets of the elementary simple types predefined with MPEG-7 DDL; none of the models to our knowledge supports the simple type derivation methods of MPEG-7 DDL.

In addition to the models originating from research, several data models for XML documents have appeared in the context of standardization efforts. Prominent representatives are the XPath Data Model which constitutes the foundation for the XPath language [7], the DOM

Structure Model which is specified along with the DOM API by the Document Object Model (DOM) standard [32], and the XML Information Set [8]. These models generally offer detailed and accurate representations of XML documents. But their applicability for the processing of MPEG-7 media descriptions is limited, since they neglect the types of the basic contents of a description representing them always as text.

There are two recent developments with regard to standard data models for XML documents which are of particular interest for our aim to adequately represent MPEG-7 media descriptions, namely DOM Level 3 [33] and the XQuery 1.0 and XPath 2.0 Data Model [13]. The current DOM Level 3 standardization effort originally not only aimed at the fine-grained representation of XML documents but also at the fine-grained representation of the schema definitions to which the documents comply. In that context, Abstract Schemas [4] have been proposed as a schema-dialect-neutral model for the representation of schema definitions. Hence, Abstract Schemas might serve as a basis for the representation of media description schemes expressed in MPEG-7 DDL. However, Abstract Schemas do not reach the expressiveness of MPEG-7 DDL. It is furthermore noteworthy that work on Abstract Schemas has recently been canceled and that they will not be included in the final version of DOM Level 3.[2]

The XQuery 1.0 and XPath 2.0 Data Model is currently being defined as the foundation of the XQuery standardization effort for a common XML query language [3]. What makes the model interesting with regard to the representation of MPEG-7 media descriptions is that it supports the elementary data types predefined by XML Schema for the typed representation of simple element content and the content of attribute values. Nevertheless, difficulties concerning the use of the XQuery 1.0 and XPath 2.0 Data Model for MPEG-7 still remain: with the exception of lists, the current working draft does not support the far majority of the simple type derivation methods offered by XML Schema and MPEG-7 DDL for typed representations. Furthermore, the model is still in a very unstable state. For example, the paradigm followed for the specification of the data model has just been changed fundamentally compared to the previous working draft issued in April 2002. Instead of an open structural definition, the model is now opaquely defined similar to abstract data types. Finally, as the current abstract-data-type-based specification of the model does not include operations for changing a document, the

[2]See http://lists.w3.org/Archives/Public/www-dom/2002JulSep/0010.html.

data model can provide only a static view onto an MPEG-7 media description which does not permit fine-grained updates.

4.2 TDOM in seven points

On the way to an adequate MPEG-7 database solution, the deficiencies of the existing data models for XML documents fortify the need for a new model that pays more attention to the basic requirements for the management of MPEG-7 media descriptions. With the Typed Document Object Model (TDOM), we now propose a data model which does exactly that.

TDOM is an object-oriented model for XML documents that carries on traditional DOM [32] to allow an appropriate representation of MPEG-7 media descriptions. In the following, we provide a detailed and illustrated definition of TDOM which we present in seven points closely oriented along our requirements for the management of MPEG-7 media descriptions.

Since TDOM is an object-oriented model, we employ UML class diagrams for the definition of the various classes of the model and their interrelationships.[3] Whenever it is necessary to make formal statements about TDOM, we make use of the Object Constraint Language (OCL) [1].

1. TDOM is fine-grained.

Similar to traditional DOM, TDOM faithfully and fine-grainedly reproduces the structure of XML documents with an object-oriented model that allows access and manipulation at any required granularity. We have opted for an object-oriented model because object-oriented concepts are widely supported by potential implementation platforms for TDOM today, such as most programming languages, object-oriented and object-relational DBMSs. This promises a small gap between the model and its implementations.

The class diagram of Figure 4 introduces the classes of TDOM that are responsible for the representation of an XML document and for the faithful reproduction of its structure. The class `Document` represents XML documents. In the model, a document is identified by its storage location addressed with an URL. A document can optionally be characterized by

[3]For the purpose of clarity, the classes in the diagrams do not show any methods for accessing and manipulating attributes and associations.

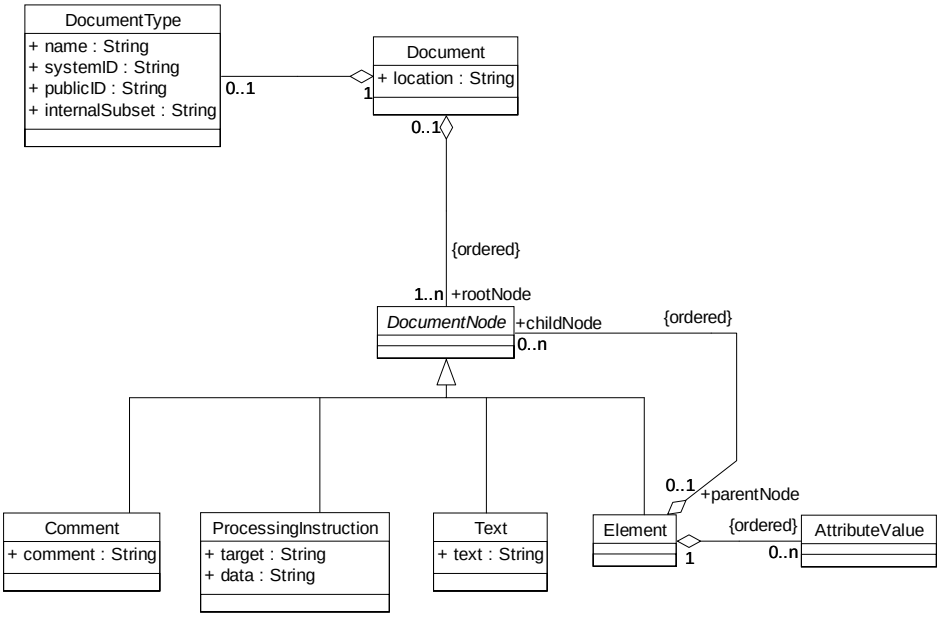


Figure 4: Representation of document structure (UML class diagram)

document type information (modeled by the class `DocumentType`) that might be conveyed in its `<!DOCTYPE>` section. As an entry point to its contents, each document refers to the sequence of root document nodes constituting the top level of its hierarchical structure, which is expressed by the aggregation between the classes `Document` and `DocumentNode`.

Being an abstract base class, `DocumentNode` subsumes one class each for the representation of the primal kinds of nodes of which an XML document may consist: `Comment` represents comments, `ProcessingInstruction` represents processing instructions together with their associated source and target declarations, `Text` copes with text interspersed with other document nodes in mixed content, and `Element` represents elements. Through elements, the hierarchical structure of a document is established – elements are the only kind of document nodes that may contain other nodes as their child nodes. This is expressed by the aggregation between `Element` and `DocumentNode`. Since elements can be further described by attribute values, TDOM introduces the class `AttributeValue` for their representation which is aggregated by `Element`.

The UML object diagram of Figure 5 exemplifies the structural representation of an MPEG-7 media description with TDOM using the example melody description of Figure 2. The descrip-

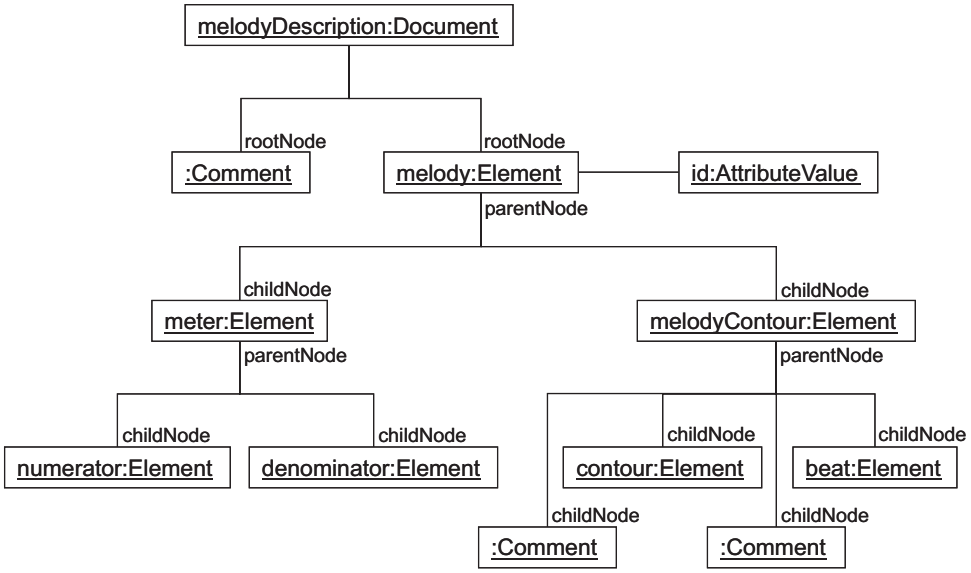


Figure 5: Structural representation of the example MPEG-7 media description with TDOM (UML object diagram)

tion itself is represented by the object of the class `Document` depicted at the top of the diagram; the document nodes contained in the description are represented by objects of the TDOM-classes corresponding to the particular kind of node. Via the references of the `Document` object to its root nodes and the references of the `Element` objects to their child nodes and attribute values, TDOM reconstructs the hierarchical structure of the example description. Outgoing from the `Document` object, an application can thus traverse the structure of the example media description and access and manipulate any desired document node at any granularity.

There are some limitations on the allowable structure of XML documents. There is the restriction that the attribute names and attribute namespaces of the attribute values associated with an element must be unique. This is formally expressed in OCL by Contraint 1. The constraint employs the shorthands `attNamespace` and `attName` to refer to the attribute namespace and attribute name of an attribute value. These shorthands will be defined later under point 3.

Constraint 1 (Unique attribute values)

context `Element`

inv: `attributeValue -> forAll(av1, av2 |`

```
av1.attNamespace = av2.attNamespace and
av1.attName = av2.attName implies
      av1 = av2)
```

There is the further limitation that there must be exactly one element among the root nodes of a document. Also, there must not be a text node among the root nodes. These restrictions are formally expressed by Constraint 2.

Constraint 2 (Root nodes)

```
context Document
inv:  rootNode -> one(e | e.oclIsTypeOf(Element))
inv:  not(rootNode -> exists(t | t.oclIsTypeOf(Text)))
```

The single element among the root nodes is called the root element of the document. The term root element is formalized by Definition 1.

Definition 1 (Root element)

```
context Document def:
let rootElement :  Element =
      rootNode -> any(e | e.oclIsTypeOf(Element))
```

2. TDOM is typed.

Traditional DOM represents the basic contents of an XML document as text prohibiting appropriate access to non-textual data. With TDOM, in contrast, it is our primary goal to exploit type information contained in media description schemes to which MPEG-7 media descriptions comply. The idea is to keep simple content of elements and the content of attribute values in a way that is appropriate for the particular content type. For this reason, we have made typed representations a central concept of TDOM.

In typed representation, elements and attribute values are tightly coupled to the element types and attributes declared in the schema definition accompanying an XML document. According to the class diagram of Figure 6 which unveils more details regarding the representation of elements and attribute values with TDOM, an element or attribute in typed representation

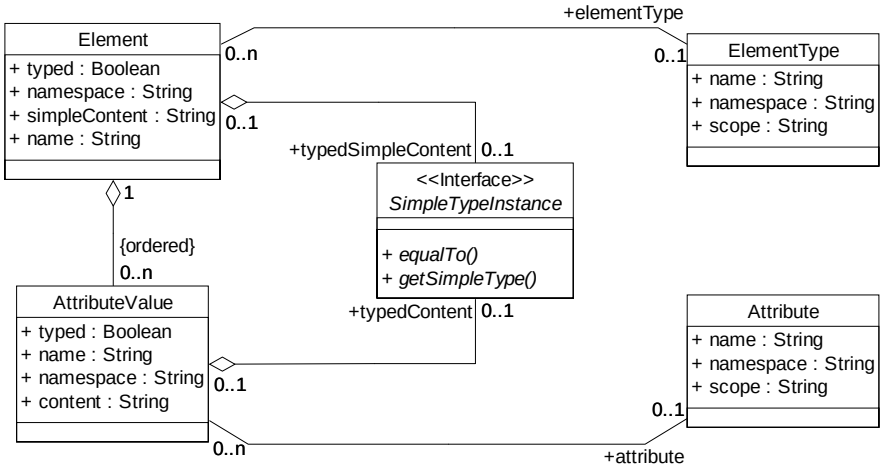


Figure 6: Representation of elements and attribute values (UML class diagram)

(indicated by the boolean attribute `typed` of the classes `Element` and `AttributeValue`) is explicitly associated with the respective element type or attribute it instantiates, i.e., it is valid to. This is expressed by the associations between the classes `Element` and `ElementType` and `AttributeValue` and `Attribute` respectively. Element types and attributes are characterized by their names and namespaces and an optional scope.

Furnishing the classes `ElementType` and `AttributeValue` with the `scope` attribute is a tribute to the fact that MPEG-7 DDL, just like other schema definition languages for XML documents, not only allows to declare element types and attributes that are globally visible but also those that are only visible within a certain scope, e.g., a complex type. In order to distinguish different element types and attributes with identical names and namespaces that might exist within different scopes of one and the same schema definition, the attribute `scope` contains a string uniquely describing the scope in which the element type or attribute is visible.

The explicit association of elements and attribute values in typed representation with their element types and attributes declared in the schema definition not only provides an index allowing to efficiently look up all instances of a certain element type or attribute in a document. It also opens up type information that is used to acquire an adequate representation of the content of elements and attribute values: elements with simple content and attribute values in typed representation do not keep their content as text, but rather encapsulate their content within an object. This is captured in Figure 6 by the aggregations between the classes `Element`

and `AttributeValue` and the interface `SimpleTypeInstance`, which the objects holding the content have to implement as a minimum (we will describe this interface in detail later under point 5). Inside these objects, the content is kept in a way adequate to the content type declared for the element type or attribute in the schema definition. The objects offer methods specific to the content type that allow applications to appropriately access and operate on the content.

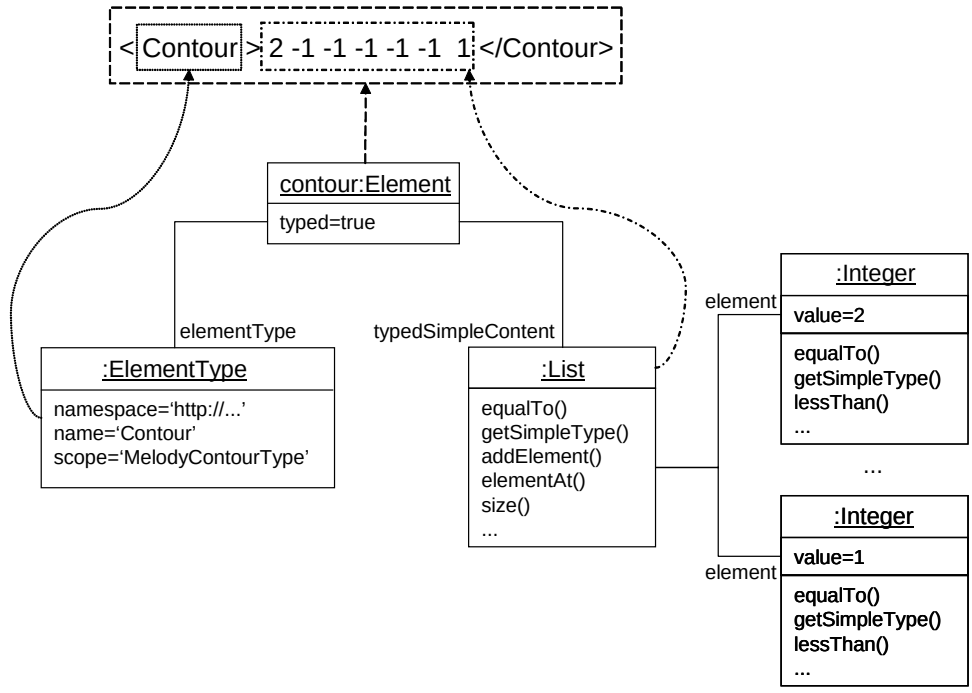


Figure 7: Typed representation of the example `<Contour>` element (UML object diagram)

In Figure 7, we give an example for a better understanding of typed representations. At the top of the Figure, the `<Contour>` element of our example MPEG-7 media description of Figure 2 is shown. Below the `<Contour>` element, the objects used for its representation are depicted in UML object diagram notation. A dashed arrow between an object and the `<Contour>` element indicates which part of the element is represented by the object. TDOM represents the whole element by an object of the class `Element`. In typed representation, an element is explicitly associated with the element type it instantiates. This is captured in the example by the reference from the `Element` object to the `ElementType` object representing the element type `Contour` that has been declared within the complex type `MelodyContourType` in the media description scheme of Figure 1. It is known from this element type declaration that the valid contents for elements of the type `Contour` are lists of integer values. As the example element is

kept in typed representation, its content is thus encapsulated within an object of a class that offers an implementation for lists with reasonable methods to work with them – the class `List`. Since the elements of the list are known to be integer values, they are encapsulated in objects of the class `Integer` providing an implementation for integer values.

With this representation of the `<Contour>` element at hand, an application can now reasonably operate on the element. E.g., an application can query the size of the list making up the content of the element and access its single elements, all by invoking the appropriate methods `size()` and `elementAt()` offered by the class `List`.

There are some constraints that have to be obeyed with regard to typed representations though. It must be ensured that the content of an element in typed representation is either simple, i.e., it is represented by an object implementing the interface `SimpleTypeInstance`, or complex, i.e., its content consists of further child nodes via the aggregation between `Element` and `DocumentNode` given in the class diagram of Figure 4, both not both. This is expressed by Constraint 3.

Constraint 3 (Typed element content)

```
context Element
inv:  typed implies
          (typedSimpleContent -> notEmpty() implies
               childNode -> isEmpty()) and
          (childNode -> notEmpty() implies
               typedSimpleContent -> isEmpty())
```

Moreover, it must be assured that the element type associated with a root element in typed representation is globally visible, i.e., it may not be scoped. This is expressed by the subsequent Constraint 4.

Constraint 4 (Typed root element)

```
context Document inv:
     rootElement.typed implies
          rootElement.elementType.scope = null
```

3. TDOM needs not to be typed.

Even though it is the central goal of TDOM to exploit type information available in schema definitions to infer an adequate, typed representation of elements and attribute values for appropriate access, there are nevertheless situations in which type information is not available. This might be the case, for example, if a media description scheme makes use of constructs that prohibit type inference for parts of an MPEG-7 media description. As an example, the constructs `<any>` and `<anyAttribute>` of MPEG-7 DDL state that an arbitrary element or attribute value is valid as the content of a certain element type respectively. This includes elements and attribute values for which no further schema information is available. Obviously, it will prove difficult to create a typed representation of such elements and attribute values.

As a fallback for such situations, TDOM offers the notion of untyped representations. In untyped representation, elements or attribute values are decoupled from the schema definition, not being explicitly associated with the definition's element types or attributes. They maintain the name and namespace of their respective element type or attribute as well as their content in the corresponding textual attributes of the classes `Element` and `AttributeValue` that are depicted in the class diagram of Figure 6 – with all the problems involved related to the appropriate access to the content.

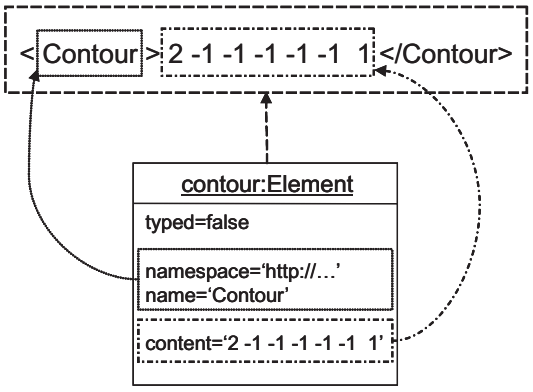


Figure 8: Unyped representation of the example `<Contour>` element (UML object diagram)

The UML object diagram of Figure 8 illustrates the concept of untyped representations. The diagram once more depicts the `<Contour>` element taken from our example MPEG-7 media description of Figure 2 – this time, however, in untyped representation. Again, dashed arrows indicate which parts of the object diagram correspond to which part of the element

shown at the top of the figure. The encoding of the list of integer values constituting the content of the element in the textual attribute `content` is especially noteworthy. There is no indication for an application that this string represents a list. Without further knowledge, the content of the element can thus only be processed as a string with doubtable usefulness. Even if the application had that knowledge, it would always have to parse the string and cast it to an appropriate internal representation before adequate access to the list of integer values could take place.

There are some contraints with regard to untyped representations. Just as with elements in typed representation, it must be assured that the content of an element in untyped representation is either simple or complex. This is the purpose of Constraint 5.

Constraint 5 (Untyped element content)

```
context Element
inv:  not(typed) implies
          (simpleContent <> null implies
                childNode -> isEmpty()) and
          (childNode -> notEmpty() implies
                simpleContent = null)
```

Moreover, elements and attribute values must be created in a consistent manner: an element or attribute value has to be either in typed or in untyped representation but not in an odd mixture of both. I.e., an element or attribute value in typed representation should not make use of the attributes of the classes `Element` and `AttributeValue` that are intended for untyped representations and vice versa. This is covered by Constraint 6.

Constraint 6 (Consistency of representations)

```
context AttributeValue
inv:  typed implies
          attribute -> notEmpty() and
          typedContent -> notEmpty() and
          name = null and namespace = null and content = null
inv:  not(typed) implies
```

```
            attribute -> isEmpty() and
            typedContent -> isEmpty() and
            name <> null and namespace <> null and content <> null

context Element
inv:  typed implies
            elementType -> notEmpty() and
            name = null and namespace = null and
            simpleContent = null
inv:  not(typed) implies
            elementType -> isEmpty() and
            typedSimpleContent -> isEmpty() and
            name <> null and namespace <> null
```

Finally, we are now able to provide the reader with the definition of the formal shorthands `attName` and `attNamespace` that we have used in Constraint 1 to address the name and namespace of the attribute to which an attribute value belongs:

Definition 2 (Attribute name and namespace)

```
context AttributeValue def:
let attName :  String =
      if typed then
            attribute.name
      else
            name
      endif
let attNamespace :  String =
      if typed then
            attribute.namespace
      else
            namespace
      endif
```

4. TDOM can be typed and untyped at the same time.

We have already mentioned before that MPEG-7 DDL offers constructs, e.g., `<any>` and `<anyAttribute>`, which permit the inclusion of elements and attribute values in an MPEG-7 media description for which no further schema information is available that could be used for the construction of typed representations. As a consequence, TDOM has to keep these elements and attribute values in untyped representation. Considering the advantages of typed representations, however, it is undoubtedly unattractive to keep all the description's other elements and attribute values for which schema information is available in untyped representation as well, just because of the existence of a few untypeable elements and attribute values.

For this reason, we explicitly allow elements and attribute values in typed and untyped representation to coexist in a single document. We leave it very well possible that an element in typed representation has attribute values and child elements in untyped representation among its constituents: the declaration of the element type to which the element refers in typed representation might allow arbitrary child elements and attribute values including those for which typed representations cannot be inferred due to the lack of type information.

On the contrary, we do not allow an element in untyped representation to contain child elements and attribute values in typed representation. In untyped representation, the exact element type of an element is not known (only its name and namespace) and with it the type's declaration. Without the declaration, the exact element types and attributes of the child elements and attribute values of the element are not known as well and therefore the child elements and attribute values cannot be in typed representation. This restriction is captured by Constraint 7.

Constraint 7 (Untyped representation of elements)

```
context Element
inv:  not(typed) implies
          not(childNode -> exists(e :  Element | e.typed)) and
          not(attributeValue -> exists(av | av.typed))
```

5. TDOM supports arbitrary simple types.

From the perspective of MPEG-7 DDL, an object representing the simple content of an element

or the content of an attribute value in typed representation constitutes an instance of a simple type. MPEG-7 DDL predefines a comprehensive set of elementary simple types whose instances may occur as the content of elements and attribute values in MPEG-7 media descriptions, as well as a variety of derivation methods for the definition of new simple types.

For the handling of simple types and their instances, TDOM provides a generic simple type framework. Using that framework, support for arbitrary simple types and their instances can be smoothly integrated with TDOM which keeps the model simple and extensible and relieves us from the need to anticipate and to hardwire all supported simple types into the model.

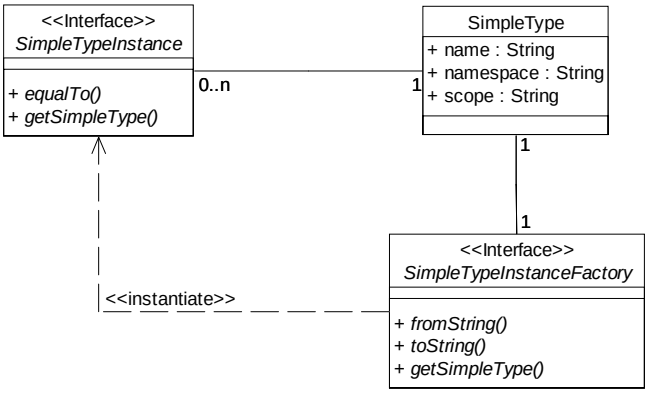


Figure 9: Simple type framework (UML class diagram)

The simple type framework of TDOM is presented in the class diagram of Figure 9. As shown in the diagram, the framework represents simple types by the class `SimpleType`. A `SimpleType` object serves to represent either an elementary simple type predefined by MPEG-7 DDL or a simple type specific to a certain schema definition that has been derived from a predefined simple type using the constructs for type derivation available with MPEG-7 DDL. TDOM attributes a simple type with its name, namespace and an optional scope in which it is visible in a schema definition.

TDOM represents the instances of a simple type as objects of a class offering a meaningful implementation for the instances of that type. Each of these objects encapsulates a suitable representation of the simple type instance and offers type-specific functionality that can be used by applications to appropriately operate on the instance. The simple type framework, however, abstracts from the concrete classes implementing a certain simple type. Instead, it

demands a minimal functionality that they have to provide which is specified by the interface `SimpleTypeInstance`. The interface `SimpleTypeInstance` consists of the methods `equalTo()`, which provides basic lookup functionality for simple type instances as it can be used to compare two simple type instances for equality, and `getSimpleType()`, which delivers simple type of the instance. Each simple type keeps track of its instances which is expressed by the association between `SimpleType` and `SimpleTypeInstance`.

Having provided a way to represent simple types and their instances, it must be possible to construct simple type instances from the textual representation in which they are conveyed in XML documents as well as to reconstruct that textual representation from a given simple type instance. For that purpose, each simple type references a factory for the production of its instances. TDOM demands a minimum functionality for each of these factories which is collected by the interface `SimpleTypeInstanceFactory`. The interface provides the methods `fromString()`, which produces an instance of the simple type to which the factory is related from the textual representation in which the instance is conveyed in an XML document, `toString()`, which returns a textual representation of a simple type instance, and `getSimpleType()`, which delivers the simple type whose instances are produced by the factory.

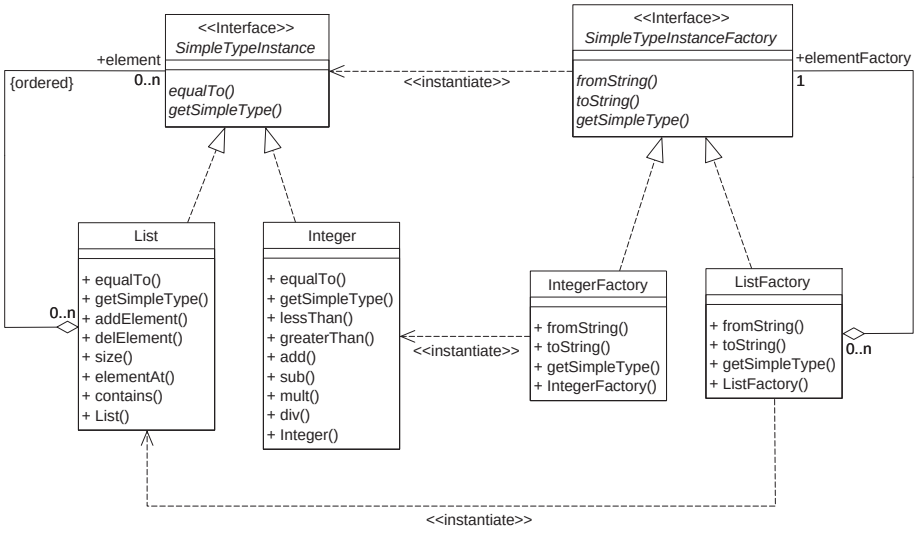


Figure 10: Example implementation of simple type support (UML class diagram)

Figure 10 gives an impression of how the simple type framework can be utilized to support

a set of simple types. In our example, support for the simple type integer and its instances is provided. This is achieved by defining the class `Integer` which, beyond type-specific methods, e.g., for adding and substracting, implements the interface `SimpleTypeInstance` so that its objects are usable as the content of elements and attribute values in typed representation. For the construction of `Integer` objects from the textual representations in which integer values are encoded in XML documents, the class `IntegerFactory` is supplied implementing the Interface `SimpleTypeInstanceFactory`.

Likewise, TDOM can accommodate derivation methods for simple types. Figure 10 exemplifies the integration of a list type with TDOM. Similar to other simple types, the classes `List` and `ListFactory` provide support for the instances of the list type and for their construction by implementing the interfaces `SimpleTypeInstance` and `ListFactory`, respectively. In contrast to elementary simple types such as integer, however, the construction of instances of a derived simple type typically includes the construction of instances of the base type. In our example, the construction of a list includes the construction of instances of the simple type of its elements. Therefore, the factory for the list type must refer to the factory of its base type, modeled by the aggregation between `ListFactory` and `SimpleTypeInstanceFactory`.

With these classes, we are able to adequately represent lists of integer values in TDOM and to construct them from the textual representation in which they are conveyed in XML documents; we can thus already build the typed representation of the example `<Contour>` element of Figure 7. The approach outlined for the implementation of simple types can be systematically followed to the extent where all the elementary simple types and simple type derivation methods coming with MPEG-7 DDL are supported.

In the following, we formally specify the semantics of the methods of interfaces `SimpleTypeInstance` and `SimpleTypeInstanceFactory` introduced by the simple type framework. Constraint 8 starts with the interface `SimpleTypeInstance`.

Constraint 8 (Simple type instance)

```
context SimpleTypeInstance::equalTo(SimpleTypeInstance sti) :
      Boolean
post: sti = self implies
            result = true
```

```
post: result = true implies
            self.getSimpleType() = sti.getSimpleType()

context SimpleType inv:
      simpleTypeInstance -> forAll(sti |
            sti.getSimpleType() = self)
```

The first postcondition of the method `equalTo()` ensures that a simple type instance is always equal to itself. The second postcondition states that, in order to be equal, two simple type instance must be of the same simple type. The invariant for the class `SimpleType` defines that the result of the method `getSimpleType()` on a simple type instance is the simple type associated with the simple type instance.

Constraint 9 describes the interface `SimpleTypeInstanceFactory` in more detail.

Constraint 9 (Simple type instance factory)

```
context SimpleTypeInstanceFactory::fromString(String s) :
            SimpleTypeInstance
post: result <> null implies
            result.getSimpleType() = self.getSimpleType()
post: result <> null implies
            self.simpleType.simpleTypeInstance -> forAll(sti |
                  self.toString(sti) = s implies
                        sti.equalTo(result))

context SimpleType
inv:  simpleTypeInstance -> forAll(sti1, sti2 |
            simpleTypeInstanceFactory.toString(sti1) =
                  simpleTypeInstanceFactory.toString(sti2) implies
                        sti1.equalTo(sti2))
inv:  simpleTypeInstanceFactory.getSimpleType() = self
```

The first postcondition of the method `fromString()` assures that an instance of a simple type successfully constructed from a textual representation refers to the simple type associated

with the factory. The second postcondition states that if an instance of a simple type is successfully constructed from a textual representation that is the result of the call of the method `toString()` on another instance of the same type, then both instances are equal. In other words, `fromString()` constitutes the inverse method to `toString()`.[4] In general, we can say that if calling `toString()` on two instances of the same simple type yields the same textual representation, then both instances are also equal to each other. This is formally described by the first invariant of the class `SimpleType` in the constraint above. Finally, the second invariant of `SimpleType` defines that the result of the call of the method `getSimpleType()` on a simple type instance factory is always the simple type to which the factory belongs.

6. TDOM facilitates flexible, fine-grained updates.

The basic characteristics of TDOM pave the way to sophisticated updates on MPEG-7 media descriptions. The model's fine-grained representation of an XML document's structure allows applications to access any part of the document and to perform modifications at any granularity. Moreover, the combination of the concepts of typed and untyped representation of elements and attribute values offer great flexibility with respect to updates.

To illustrate the benefit of having both typed and untyped representation available, we consider an update on our example media description of Figure 2. An application might want to replace the `<Beat>` element by a new one. A natural way to perform this task would be the deletion the `<Beat>` element followed by the insertion of the new `<Beat>` element as a child of the element `<MelodyContour>`.

Did TDOM only support typed representations, it would have to be ensured after every single update operation that every element and attribute value affected by the update is valid with respect to the declaration of the particular element type or attribute it is associated with in typed representation. This is very rigid. In our example, the deletion the `<Beat>` element already violates the validity of the `<MelodyContour>` element with respect to the element type `MelodyContour`, since, according to the schema definition of Figure 1, an element of that type

[4]The opposite need not to be true. For example, one and the same float value might be constructed from different textual representations (e.g., `123e-2` and `12.3e-1` represent the same float value `1.23`). However, calling `toString()` on the float value always yields just one of the possible textual representations which does not need to be the one from which the value has been constructed.

must contain exactly one element of type `Beat`. Thus, the deletion and thereby the whole sequence of update operations would have to be refused – even though the subsequent insertion of the new `<Beat>` element would restore schema consistency.

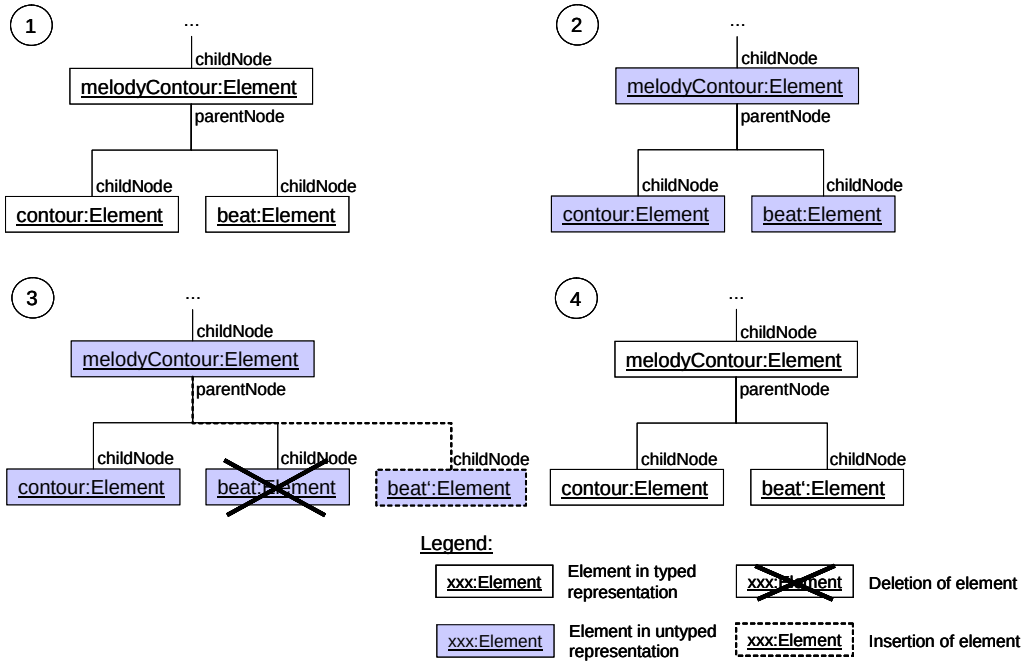


Figure 11: Switching between corresponding representations for an update

But having the additional means of untyped representations at hand (see Figure 11), applications can transform elements and attribute values in typed representation (1) that are affected by an update to a corresponding untyped representation (2). Thereby, they are decoupled from the element types and attributes of the schema definition. Any desired sequence of update operations can then be performed without being concerned with schema validity (3). After all update operations have been completed, the updated elements and attribute values can be brought back to corresponding typed representations (4) as long as the document is still valid with respect to the schema definition.

What do we mean exactly by the terms corresponding untyped representation and corresponding typed representation? A corresponding untyped representation should reproduce an element or attribute value that is kept in typed representation as faithful as possible with the means of untyped representation. Likewise, a corresponding typed representation should faithfully reproduce an element or attribute value in untyped representation by the means of typed

representation.

Definition 3 formalizes a natural notion of correspondence for attribute values. An attribute value `av'` in untyped representation constitutes a corresponding untyped representation of an attribute value `av` in typed representation (formally: `av.CUR(av')`), if `av'` refers to the name and namespace of the attribute associated with `av` and if the textual content of `av'` is a textual representation of the simple type instance forming the content of `av`. Conversely, we can also say that `av` constitutes a corresponding typed representation of `av'` (formally: `av'.CTR(av)`).

Definition 3 (Corresponding representations of attribute values)

```
context AttributeValue def:
let CUR(AttributeValue av) :  Boolean =
      typed and not(av.typed) and
      attribute.namespace = av.namespace and
      attribute.name = av.name and
      typedContent.simpleType.simpleTypeInstanceFactory.
            fromString(av.content) <> null and
      typedContent.simpleType.simpleTypeInstanceFactory.
            fromString(av.content).equalTo(typedContent)
let CTR(AttributeValue av) :  Boolean =
      av.CUR(self)
```

Definition 4 formally introduces a notion of correspondence for elements.[5] Following that definition, an element `e'` in untyped representation constitutes a corresponding untyped representation of an element `e` in typed representation (formally: `e.CUR(e')`), if `e'` refers to the name and namespace of the element type associated with `e`. If `e` has simple content, it is furthermore demanded that `e'` has simple content as well and the simple content of `e'` is a textual representation of the simple type instance forming the simple content of `e`. If `e` has complex content, however, it is demanded that `e'` also has complex content and the child nodes of `e'` are equal to the child nodes of `e` – with the exception of elements: the child elements of `e'` are expected to be corresponding untyped representations of the respective child elements

[5] In the definition, we assume the existence of the method `deepEqualTo()` to compare two objects for deep equality.

of e. Finally, every attribute value of e' must appear among the attribute values of e or be a corresponding untyped representation of an attribute value of e. With all these conditions fulfilled, we can conversely say that e constitutes a corresponding typed representation of e' (formally: e'.CTR(e)).

Definition 4 (Corresponding representations of elements)

```
context Element def:
let CUR(Element e) :  Boolean =
      typed and not(e.typed) and
      elementType.namespace = e.namespace and
      elementType.name = e.name and
      (typedSimpleContent -> notEmpty() implies
            e.simpleContent <> null and
            typedSimpleContent.simpleType.simpleTypeInstanceFactory.
                  fromString(e.simpleContent) <> null and
            typedSimpleContent.simpleType.simpleTypeInstanceFactory.
                  fromString(e.simpleContent).equalTo(typedSimpleContent)
      ) and
      (childNode -> notEmpty() implies
            childNode -> size() = e.childNode -> size() and
            Sequence{1..childNode -> size()} -> forAll(i :  Integer |
                  childNode -> at(i).deepEqualTo(e.childNode -> at(i)) or
                  (childNode -> at(i).oclIsTypeOf(Element) and
                  e.childNode -> at(i).oclIsTypeOf(Element) and
                  childNode -> at(i).CUR(e.childNode -> at(i))))
      ) and
      (attributeValue -> notEmpty() implies
            attributeValue -> size() = e.attributeValue -> size() and
            attributeValue -> forAll(av1 |
                  e.attributeValue -> exists(av2 |
                        av1.deepEqualTo(av2) or av1.CUR(av2))))
let CTR(Element e) :  Boolean =
```

```
e.CUR(self)
```

The construction of a corresponding untyped representation of an element or attribute value in typed representation is straightforward as the typed representation generally contains all the information that must be included with the corresponding untyped representation. An attribute value in typed representation keeps the name and namespace of the attribute with the attribute referred to by the attribute value in typed representation. Moreover, a textual representation of the content of the attribute value can be obtained from the simple type instance by employing the method `toString()` of the associated simple type instance factory.

Likewise, an element in typed representation keeps the name and namespace of the element type with the element type referenced. A textual representation of a potentially existing simple content can be derived from the simple type instance representing that simple content in typed representation via the associated simple type instance factory. Corresponding untyped representations of any child elements and attribute values of the element can be obtained recursively.

In contrast to the construction of a corresponding untyped representation, the construction of a corresponding typed representation of an element or attribute value in untyped representation is more complicated. This is due to the fact that elements or attribute values in untyped representation do not, apart from the name and namespace of their respective element type or attribute, convey type information that would allow the construction of a valid corresponding typed representations solely on the basis of the untyped representation. Additional information in form of a schema definition is needed. With the element types and attributes and the associated type information contained in a schema definition, the respective element type or attribute can be inferred to which an element or attribute value in untyped representation is valid. Based on the inferred element type or attribute and the associated type information, a corresponding typed representation can then be constructed straightforwardly.

For the inference of the respective element types and attributes of a schema definition to which the elements and attribute values of a document are valid, we have developed a formal mechanism called typing automaton. A typing automaton is an adaptation of a regular tree automaton [35, 5] for use with TDOM. We have to refrain from the detailed treatment of typing automata at this point, however, as this would exceed the scope of the paper.

7. TDOM takes account of MPEG-7 DDL.

TDOM has been designed to take advantage of media description schemes written in MPEG-7 DDL which accompany MPEG-7 media descriptions. These can be used to obtain typed representations of elements and attribute values such that the document's basic contents are kept in a fashion appropriate to the respective content type. To facilitate extensive construction of such typed representations for the basic contents that may occur in media descriptions, TDOM furthermore is capable of embracing the plenitude of predefined simple types and simple type derivation methods that come with MPEG-7 DDL via the simple type framework.

Since the purpose of TDOM is to effectively represent media descriptions and not the description schemes to which they comply, however, the detailed representation of an MPEG-7 DDL media description scheme coming with a media description has been left out of the scope of the model. Abstracting from the schema definition language, TDOM just presumes the existence of element types, attributes, and simple types in a schema definition for the modeling of typed representations.

The decision to abstract from the details of the schema definition language has the convenient side effect that it leaves TDOM, though primarily intended for MPEG-7, applicable to other application domains. In other domains, the typed representation of the basic contents of an XML document might also be desirable, but schema definition languages different from MPEG-7 DDL might play dominant roles. As an example taken from the domain of electronic data interchange, the structure of business documents following the XML Common Business Library (xCBL) [49] is defined with the schema definition languages SOX [9] and XDR [15].

In order to be able to construct typed representations of the basic contents of an MPEG-7 media description from a media description scheme expressed in MPEG-7 DDL, a database solution using TDOM as its data model must, of course, be able to process the description scheme and provide means for its detailed representation. These can base on data models for XML schema definitions such as Abstract Schemas [4] or, as it is the case with typing automata, on well-known formalisms for the validation and typing of XML documents such as regular tree automata [35, 5].

5 Conclusion

Starting out with essential requirements for the management of MPEG-7 media descriptions, we have analyzed current XML database solutions for their suitability for use in the context of MPEG-7. Facing the deficiencies of these solutions with respect to these requirements, we have realized the need for more adequate database solutions for MPEG-7 media descriptions. As a foundation of such a solution, we have introduced the Typed Document Object Model, a data model for XML documents specifically designed with the requirements for the management of MPEG-7 media descriptions in mind. We have highlighted TDOM's key features and given a thorough definition of the model.

We have fully implemented TDOM with Java on the basis of the object-oriented DBMS ObjectStore. Our implementation comes with a schema catalog that manages media description schemes expressed in MPEG-7 DDL in form of typing automata and uses these automata for the validation of MPEG-7 media descriptions and the automatic construction of appropriate typed representations of the contents of the descriptions. Furthermore, our implementation includes an indexing component supporting a variety of secondary access methods for indexing the basic contents of a media description including Hashtables, B-Trees, and R-Trees. We are currently providing a processor for XPath expressions [7] that exploits schema information and indexes available for an optimized query evaluation. The XPath processor forms the heart of optimizing processors for XQuery [3] and XSLT [6] that we plan to implement in future.

References

[1] Analysis & Design Platform Task Force. Unified Modeling Language (UML). OMG Available Specification Version 1.4, Object Management Group (OMG), September 2001.

[2] P.V. Biron and A. Mahotra. XML Schema Part 2: Datatypes. W3C Recommendation, World Wide Web Consortium (W3C), May 2001.

[3] S. Boag, D. Chamberlin, M.F. Fernandez, et al. XQuery 1.0: An XML Query Language. W3C Working Draft, World Wide Web Consortium (W3C), August 2002.

[4] B. Chang, E. Litani, J. Kesselman, and R. Rahman. Document Object Model (DOM) Level 3 Abstract Schemas Specification. W3C Note Version 1.0, World Wide Web Consortium (W3C), July 2002.

[5] B. Chidlovskii. Using Regular Tree Automata as XML Schemas. In *Proc. of the IEEE Advances in Digital Libraries 2000 (ADL 2000)*, Washington, D.C., May 2000.

[6] J. Clark. XSL Transformations (XSLT). W3C Recommendation, World Wide Web Consortium (W3C), November 1999.

[7] J. Clark and S. DeRose. XML Path Language (XPath). W3C Recommendation Version 1.0, World Wide Web Consortium (W3C), November 1999.

[8] R. Cowan and R. Tobin. XML Information Set. W3C Recommendation, World Wide Web Consortium (W3C), October 2001.

[9] A. Davidson, M. Fuchs, M. Hedin, et al. Schema for Object-Oriented XML. W3C Note Version 2.0, World Wide Web Consortium (W3C), July 1999.

[10] DCMI. Dublin Core Metadata Element Set. DCMI Recommendation Version 1.1, Dublin Core Metadata Initiative (DCMI), July 1999.

[11] A. Deutsch, M. Fernandez, and D. Suciu. Storing Semistructured Data with STORED. In *Proc. of the ACM SIGMOD International Conference on Management of Data (SIGMOD 1999)*, Philadelphia, Pennsylvania, June 1999.

[12] eXcelon Corp. Managing DXE. System Documentation Release 3.5, eXcelon Corp., December 2001.

[13] M. Fernandez, J. Marsh, and M. Nagy. XQuery 1.0 and XPath 2.0 Data Model. W3C Working Draft, World Wide Web Consortium (W3C), August 2002.

[14] D. Florescu and D. Kossmann. Storing and Querying XML Data Using an RDBMS. *IEEE Data Engineering Bulletin*, 22(3), 1999.

[15] C. Frankston and H.S. Thompson. XML-Data Reduced. Unpublished Draft of W3C Note Version 0.21, University of Edinburgh, July 1998.

[16] G. Gardarin, F. Sha, and T.D. Ngoc. XML-Based Components for Federating Multiple Heterogeneous Data Sources. In *Proc. of the 18th International Conference on Conceptual Modeling (Conceptual Modeling - ER '99)*, Paris, France, November 1999.

[17] R. Goldman, J. McHugh, and J. Widom. From Semistructured Data to XML: Migrating the Lore Data Model and Query Language. In *Proc. of the ACM SIGMOD Workshop on The Web and Databases (WebDB '99)*, Philadelphia, Pennsylvania, June 1999.

[18] S. Higgins, O. Alonso, S. Banerjee, et al. Oracle 9i Application Developer's Guide – XML. Product Documentation Release 1 (9.0.1), Oracle Corp., June 2001.

[19] G. Huck, I. Macherius, and P. Fankhauser. PDOM: Lightweight Persistency Support for the Document Object Model. In *Proc. of the Workshop "Java and Databases: Persistence Options" of the 14th Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA '99)*, Denver, Colorado, November 1999.

[20] IBM Corp. IBM DB2 Universal Database – XML Extender Administration and Programming. System Documentation Version 7, IBM Corp., 2000.

[21] IEEE P1484.12 Learning Object Metadata Working Group. Draft Standard for Learning Object Metadata. IEEE Draft Standard P1484.12/D6.1, Institute of Electrical and Electronics Engineers, Inc. (IEEE), April 2001.

[22] Infonyte GmbH. Infonyte-DB – User Manual and Programmers Guide. System Documentation Version 2.0.2, Infonyte GmbH, May 2002.

[23] ISO/IEC JTC 1/SC 29/WG 11. MPEG-7: Context, Objectives and Technical Roadmap, V.12. ISO/IEC Document N2861, International Organization for Standardization/International Electrotechnical Commission (ISO/IEC), July 1999.

[24] ISO/IEC JTC 1/SC 29/WG 11. Information Technology – Multimedia Content Description Interface – Part 1: Systems. ISO/IEC Final Draft International Standard 15938-1:2001, International Organization for Standardization/International Electrotechnical Commission (ISO/IEC), November 2001.

[25] ISO/IEC JTC 1/SC 29/WG 11. Information Technology – Multimedia Content Description Interface – Part 2: Description Definition Language. ISO/IEC Final Draft International Stan-

dard 15938-2:2001, International Organization for Standardization/International Electrotechnical Commission (ISO/IEC), September 2001.

[26] ISO/IEC JTC 1/SC 29/WG 11. Information Technology – Multimedia Content Description Interface – Part 3: Visual. ISO/IEC Final Draft International Standard 15938-3:2001, International Organization for Standardization/International Electrotechnical Commission (ISO/IEC), July 2001.

[27] ISO/IEC JTC 1/SC 29/WG 11. Information Technology – Multimedia Content Description Interface – Part 4: Audio. ISO/IEC Final Draft International Standard 15938-4:2001, International Organization for Standardization/International Electrotechnical Commission (ISO/IEC), June 2001.

[28] ISO/IEC JTC 1/SC 29/WG 11. Information Technology – Multimedia Content Description Interface – Part 5: Multimedia Description Schemes. ISO/IEC Final Draft International Standard 15938-5:2001, International Organization for Standardization/International Electrotechnical Commission (ISO/IEC), October 2001.

[29] IXIASOFT Inc. Creating Client Applications for TEXTML Server – Programmer's Guide. System Documentation Version 2.1, IXIASOFT Inc., December 2001.

[30] H.V. Jagadish, L.V.S. Lakshmanan, and D. Srivastava. Hierarchical or Relational? A Case for a Modern Hierarchical Data Model. In *Proc. of the IEEE Workshop on Knowledge and Data Engineering Exchange (KDEX'99)*, Chicago, Illinois, November 1999.

[31] C.C. Kanne and G. Moerkotte. Efficient Storage of XML Data. Technical Report 8/99, University of Mannheim, Germany, August 1999.

[32] A. Le Hors, P. Le Hégaret, L. Wood, et al. Document Object Model (DOM) Level 2 Core Specification. W3C Recommendation Version 1.0, World Wide Web Consortium (W3C), November 2000.

[33] A. Le Hors, P. Le Hégaret, L. Wood, et al. Document Object Model (DOM) Level 3 Core Specification. W3C Working Draft Version 1.0, World Wide Web Consortium (W3C), April 2002.

[34] Microsoft Corp. Microsoft SQL Server 2000 – SQLXML 2.0. System Documentation, Microsoft Corp., 2000.

[35] M. Murata. Forest Regular Languages and Tree Regular Languages. Unpublished Manuscript, Fuji Xerox Co., Ltd., May 1995.

[36] F. Nack and A.T. Lindsay. Everything You Wanted to Know About MPEG-7: Part 1. *IEEE MultiMedia*, 6(3), 1999.

[37] F. Nack and A.T. Lindsay. Everything You Wanted to Know About MPEG-7: Part 2. *IEEE MultiMedia*, 6(4), 1999.

[38] A. Salminen and F.W. Tompa. Requirements for XML Document Database Systems. In *Proc. of the ACM Symposium on Document Engineering 2001 (DocEng '01)*, Atlanta, Georgia, November 2001.

[39] A. Schmidt, M. Kersten, M. Windhouwer, et al. Efficient Relational Storage and Retrieval of XML Documents. In *Proc. of the Third International Workshop on the Web and Databases (WebDB 2000)*, Dallas, Texas, May 2000.

[40] J. Shanmugasundaram, K. Tufte, G. He, et al. Relational Databases for Querying XML Documents: Limitations and Opportunities. In *Proc. of the 25th International Conference on Very Large Data Bases (VLDB '99)*, Edinburgh, Scotland, September 1999.

[41] T. Shimura, M. Yoshikawa, and S. Uemura. Storage and Retrieval of XML Documents Using Object-Relational Databases. In *Proc. of the Database and Expert Systems Applications, 10th International Conference (DEXA '99)*, Florence, Italy, September 1999.

[42] Software AG. User Guide. System Documentation Version 3.1.1, Software AG, November 2001.

[43] K. Staken. dbXML Developers Guide 0.5. System Documentation Version 1.0, The dbXML Project, September 2001.

[44] K. Staken. Xindice Developers Guide 0.7. System Documentation Version 1.0, The Apache Software Foundation, March 2002.

[45] H.S. Thompson, D. Beech, M. Maloney, et al. XML Schema Part 1: Structures. W3C Recommendation, World Wide Web Consortium (W3C), May 2001.

[46] F. Tian, D.J. DeWitt, J. Chen, and C. Zhang. The Design and Performance Evaluation of Alternative XML Storage Strategies. *ACM SIGMOD Record*, 31(1), 2002.

[47] VRA Data Standards Committee. VRA Core Categories. VRA Standard Version 3.0, Visual Resources Assocation (VRA), February 2002.

[48] X-Hive Corp. X-Hive/DB 2.0 – Manual. System Documentation Release 2.0.2, X-Hive Corp., May 2002.

[49] xCBL.org. XML Common Business Library (xCBL). Structure Reference Version 3.5, Commerce One, Inc., November 2001.

[50] XML Global Technologies, Inc. GoXML DB Administrator Help. System Documentation Version 2.0.1, XML Global Technologies, Inc., December 2001.