An MPEG-7 Multimedia Data Cartridge

Mario Döller[a] and Harald Kosch[a]

[a]Institute of Information Technology
University Klagenfurt
9020 Klagenfurt, Austria

Abstract

Broadly used Database Management Systems (DBMS) are not able to tackle the requirements of multimedia in querying, indexing and content modeling. Therefore, extenders for multimedia data types have been proposed. These extensions, however, offer only limited semantic modeling and rely on basic index structures which do not meet the whole nature of multimedia, for instance for a Nearest-Neighbor Search. In this context, the paper presents a methodology for enhancing extensible ORDBMS for multimedia data. In particular, we introduce an MPEG-7 Multimedia Data Cartridge which includes a semantically rich metadata model for multimedia content relying on the MPEG-7 standard. Furthermore, to fulfill the needs for efficient multimedia query processing, we created in this Cartridge a new indexing and query framework for various types of retrieval operations.

Keywords: *Multimedia Databases, MPEG-7, Multimedia Index Structures.*

1. INTRODUCTION

Multimedia Database Systems (MMDBMS) organize and store multimedia data for content retrieval. These systems rely on multimedia data models representing high- and low-level abstraction of media objects for facilitating various operations (e.g., insertion, indexing, querying or retrieval). Many models have been proposed in the past which reflect the needs of database users and developers.[23] However, these models reveal important shortcomings: they are either limited by the use of one kind of multimedia data (e.g., only images are supported) or by the capacity of semantic modeling (e.g., only a keyword description of the content may be entered)*. This is as astonishing, as there has been recently published a new standard for describing the content of different types of multimedia data that offers richer semantics than existing systems. It is MPEG-7,[21] the first standard of the Moving Picture Experts Group not dealing with coding exclusively (see section 3). The reason for this re-orientation is quite straightforward, coding has reached a more or less satisfactory state (see MPEG-4), but yet we are not able to locate multimedia by standardized content. As mentioned above, the benefits of MPEG-7 have not yet reached MMDBMS. In this context, our paper presents a **Multimedia Data Cartridge** (MDC) that maps the MPEG-7 standard into a database model.

Besides the more effective modeling of multimedia content, access and retrieval has to be considered. We need an efficient support for common multimedia retrieval functions, like similarity search, semantically meaningful queries and browsing in an MMDBMS. An important tool to guarantee efficient query processing is the indexing mechanism for commonly used datatypes. Innately, most database systems provide only a limited number of integrated access

*see also our experience report in[1]

methods such as B-tree or hashing facilities. These techniques limit the use of database systems in multimedia. Available Object-Relational DBMSs (ORDBMS) provide some multimedia database extension packages such as DataBlades (IBM-Informix) or Extenders (IBM-DB2). They rarely handle indexing of d-dimensional data (d>2) and advanced similarity-search functionality, like a k-Nearest-Neighbor (k-NN) Search which is common for multimedia retrieval. Indexing feature vectors must be a core tool in a MMDBMS, as it is not unusual to index images with a feature vector dimensionality higher than 64 in.[18–20] In this context several access methods for high dimensional data have been proposed, e.g., SS-tree,[8] SR-tree,[9] M-tree,[10] X-tree[11] or TV-tree.[12] A good survey on multidimensional access methods is given in[13] and.[14] Unfortunately, these access methods are rarely available in common DBMS, nor are extensions proposed to integrate them. We are only aware of an extension package for Informix by M.Kornacker.[16] This work integrates an adapted GiST framework[2] into the Informix Dynamic Server with Universal Data Option (IDS/UDO). However, the framework does not supply a multimedia support (e.g., data model and index integration).

Being aware of the shortcomings of related approaches, we designed a methodology for enhancing extensible ORDBMS. In particular, we implemented an **MPEG-7 Multimedia Data Cartridge (MDC)**. Our system relies on the capacities of leading DBMSs (Oracle, IBM DB2 and IBM Informix, MSSQLServer) to provide services to extend their management system. Oracle's extension services were selected for the practical implementation, due to their availability (see `http://technet.oracle.com`). The multimedia extension created are, however, representatives for other DBMS extension services like DataBlades from Informix or Extenders from IBM. Our MDC enhances the Oracle type systems with an MPEG-7 compliant type system and integrates a new indexing system for multimedia retrieval and queries. This indexing system is supported by an external framework (**Multimedia Indexing Framework** (MIF)) developed and validated by us in this context.

The remainder of this paper is organized as follows. In section 2 we define general characteristics of extensible ORDBMS. This is followed by an overview of currently existing extensible ORDBMS. Section 3 introduces briefly to the MPEG-7 MDS parts used in the MDC. Then, the common characteristics of ORDBMS extension services are introduced. We discuss how our cartridge technology take advantage of them. Section 4 describes the core technologies of MDC. Section 5 presents our *Multimedia Indexing Framework* (MIF). Section 6 discusses experimental results on our indexing framework and finally Section 7 concludes this paper and points to future work.

2. MULTIMEDIA EXTENSION FOR EXTENSIBLE ORDBMS

Although many well-working multimedia retrieval systems are available, DMBS are mostly lacking support for multimedia applications, i.e., missing type models and indexing capabilities.
A practical solution to overcome these drawbacks is to use an extension service provided by many ORDBMS products (e.g., Oracle's Data Cartridges, Informix DataBlades or DB2 Extenders). Note that, there are many extensions available for GIS and simple image repositories, but there is not yet a proposal for a multimedia extension package available.

The following subsections define some main characteristics an extensible package should offer for the creation of a multimedia extension package and examines then available extension services.

2.1. General Characteristics of extensible ORDBMS

Extensible packages should enable database designers and programmers to extend at least the following parts of a DBMS system (see figure 1): type system, server execution, query processing, query optimizer and data indexing.

These parts should furnish the following characteristics. First of all they should be **server-based** which means that all components reside at the server. Furthermore they **extend** the server. By defining new types, one is able to create a solution-oriented image of a real world problem. The **integration** with the server ensures that the optimizer, indexer and other mechanisms recognize and respond to these extensions. Finally these parts should be **packetizable** for transferring them to another database.

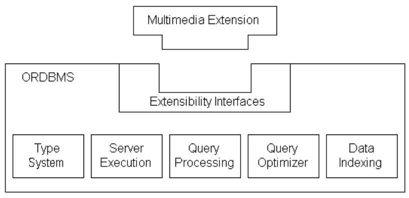


Figure 1. Multimedia Extension for extensible ORDBMS

- **Type System:** Besides the common native SQL data types, such as e.g., `INTEGER`, the type system should support new types including user-defined objects or internal large object types (e.g. `BLOB`). New user-defined object types (e.g., an Image Type) should specify the characteristics of the resource and the low- and high-level content.
- **Server Execution Environment:** This environment should allow the usage of popular programming languages such as PL/SQL, Java or C language routines for the realization of stored procedures, functions and *methods* of user-defined object types.
- **Query Optimizer:** The query optimizer should be able to consider additional statistic collections or cost functions of new access methods for choosing an optimal query plan.
- **Data Indexing:** An efficient usage of user-defined object types requires the capability of enhancing the database by new access methods. The extension package has to provide processes to maintain the index content during load and update operations and to search the index during query processing. The index itself should be stored internally as heap-organized or index-organized table or externally as an operating system file.

2.2. Oracle Data Cartridges

The extension package provided by Oracle is called *Data Cartridge*. Oracle databases are build as a modular architecture with the extensible services described above. The usual way for using Oracle's Extensibility Services is to implement a Data Cartridge that extends the extensibility interface (see[7]).
Oracle adds support for new types including user-defined objects, collections (e.g. `VARRAY`) or internal large object types (e.g., `BLOB` or `XMLType`).
Oracle's server execution environment provides two main advantages. First, the components of a data cartridge and other database procedures and functions can be implemented in any popular programming language such as PL/SQL, Java or external C language routines. Second, the type system decouples the implementation of an object's method from its specification. The specification just defines the head of the method with appropriate in and out parameters. It is not defined what kind of implementation is behind the object's method.
Further Oracle introduces the concept of an *indextype* for user-defined access methods. Each *indextype* has specific operators and uses an object that is responsible for the implementation. This object has to implement all necessary functions which are provided by ODCI (Oracle Data Cartridge Interface) for indexing e.g., ODCIIndexCreate or ODCIIndexInsert.

2.3. Informix DataBlades and DB2 Extenders

IBM offers currently two different extension packages for their database products, namely Informix DataBlades and DB2 Extenders. Besides SQL native types, Informix supports three different user defined data types. A *row type* encapsulates a grouping of multiple columns into the definition of a new data type. The *distinct type* is a customized version of an existing data type. *Opaque types* are defined by writing C, C++, or Java code to store, index and operate against that data type. DB2 Extenders generally specifies UDT‘s (user-defined types) and allows besides the CLOB type an additional XMLCLOB type for handling XML files.
Both systems enable a database developer to enhance the server execution environment by user-defined functions and stored procedures implemented in PL/SQL, C(++) or Java.

Both systems also have the possibility to enhance their database by user defined access methods. Informix DataBlades provides a *Virtual Index Interface (VII)* that allows the developer to create new indexes by implementing their corresponding methods (e.g., am_create, am_insert). DB2 Extenders uses a specific SQL command for index extensions together with some search methods. The consistency during insert, update and delete has to be managed with the help of triggers.

3. MPEG-7

MPEG-7[21, 22] is an ISO/IEC standard developed by MPEG (Moving Picture Experts Group) and formally known as ”Multimedia Content Description Interface”. The standard is organized in eight parts (from system, low- and high-level multimedia description schemes, to reference software and conformance) and provides a rich set of standardized tools to describe multimedia content. A detailed explanation of all parts is beyond the scope of this paper but can be found at.[21, 22] The multimedia data is represented with the help of descriptions. A description consists of *Description Schemes (DS)* and a set of *Descriptors (D)*. A Descriptor is a representation of a feature and defines its syntax and semantic, it may for instance be a distinctive characteristic of audio-visual information. The description scheme identifies relationships among other components (DS and D). Both, DS and D, can be defined and modified with the help of the *Description Definition Language (DDL)* which bases on XML Schema extended by new data types, like for feature vector representation. The example in figure 2 illustrates the use of MPEG-7 for describing the content of an Image.

```
<Mpeg7 xmlns="urn:mpeg:mpeg7:schema:2001"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:mpeg7="urn:mpeg:mpeg7:schema:2001"
  xsi:schemaLocation="urn:mpeg:mpeg7:schema:2001 Mpeg7-2001.xsd">
    <Description xsi:type="ContentEntityType">
      <MultimediaContent xsi:type="ImageType">
        <Image>
          <MediaLocator>
            <MediaUri> file://goal.jpg</MediaUri>
          </MediaLocator>
          <SpatialDecomposition gap="true" overlap="false">
            <StillRegion id="KloseGoal">
              <TextAnnotation>
                <FreeTextAnnotation>
                  Klose shoots a goal with right foot.
                </FreeTextAnnotation>
              </TextAnnotation>
              <VisualDescriptor xsi:type="ScalableColorType"
               numOfCoeff="64" numOfBitplanesDiscarded="0">
                 <Coeff> 2 4 6 8 10 ...
                 </Coeff>
              </VisualDescriptor>
            </StillRegion>
            <StillRegion id="Spectators">
              <TextAnnotation>
                <FreeTextAnnotation>
                  Spectators are crying
                </FreeTextAnnotation>
              </TextAnnotation>
              <VisualDescriptor xsi:type="ScalableColorType"
               numOfCoeff="64" numOfBitplanesDiscarded="0">
                 <Coeff>  3 5 7 9 11 ...
                 </Coeff>
              </VisualDescriptor>
            </StillRegion>
          </SpatialDecomposition>
        </Image>
      </MultimediaContent>
    </Description>
</Mpeg7>
```

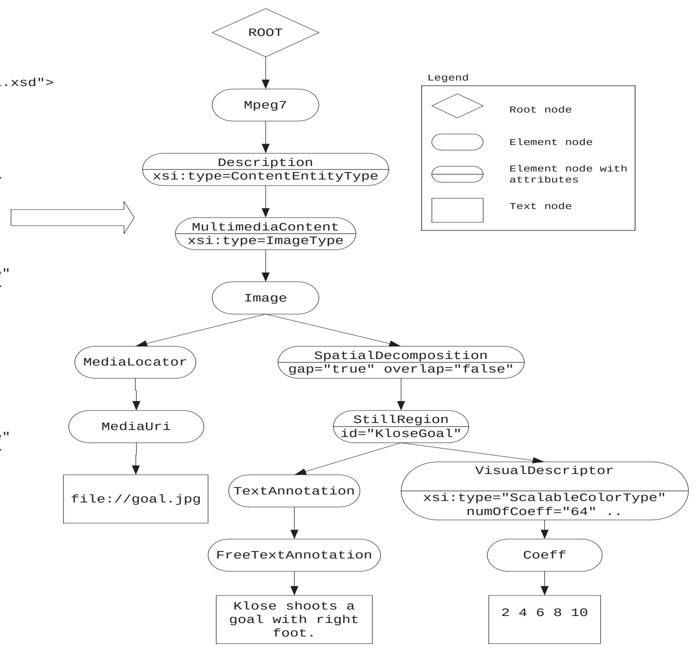


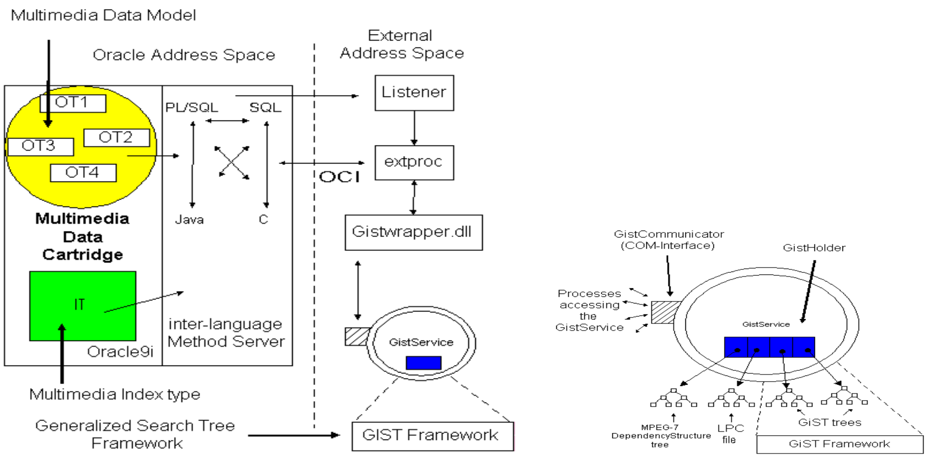
Figure 2. MPEG-7 Example Document and their corresponding DOM tree

Figure 2 uses the *ImageType and StillRegion DS* for this purpose. The ImageType DS describes the semantic, the representation and the context of images. We describe one image located at `goal.jpg`. This image is decomposed into two sub-images, each specified with a (*StillRegion*) DS. This DS offers a large number of different elements (e.g., spatial location information, text annotation, as well as means for further sub-regioning. In our example we specified an own id, e.g., KloseGoal), a *TextAnnotation* that contains text description on the image, and a *VisualDescription* which figures a feature vector of size 64 for each StillRegion.

4. MULTIMEDIA DATA CARTRIDGE

We have decided to use Oracle‘s Data Cartridge technology for the implementation of our multimedia extension called **Multimedia Data Cartridge** (MDC). The MDC currently consists of two parts (see figure 3 (a)). The fist part is the *Multimedia Data Model* which contains the metadata describing the multimedia content. It is realized with the help of the *extensible type system* of the cartridge environment. For this purpose, the MPEG-7 MDS schema is mapped to a database schema, i.e., to respective object types and tables. This mapping is discussed in subsection 4.1.

The second part is the *Multimedia Index Type* which provides an extensible indexing environment for multimedia retrieval. For instance, we may use an SS-tree or SR-tree from our Multimedia Indexing Framework (MIF) framework to index a high-dimensional color histogram. Another possibility is the use of an LPC-file[18] for an NN-search in high-dimensional spaces, or a V-tree[24] for spatial queries. These are only few possibilities which are useful and possible in our Multimedia Data Cartridge (MDC). The indexing part of MDC is described in subsection 4.2 and further sections.



(a) (b)

Figure 3. (a) Overview of the Multimedia Data Cartridge (MDC) Architecture and (b) Detailed view of GistService process

4.1. Multimedia Schema

The Multimedia Schema relies on the MPEG-7 standard to provide a complete metadata schema for structural and semantic content of multimedia data (high-level descriptions). We provide object types for low-level features, like color, shape, texture and link them to the high-level features. This enables us to retrieve multimedia data not only by low-level features, as this is done commonly in Content-Based Systems (CBR-systems),[27] but also on semantically meaningful content in combination with low-level characteristics. For instance, in a sports application we could think of a query like: “give me all goals of Miroslav Klose in the Football WorldCup which he shot by head (high-level) and where he wear the traditional white/black dresses (low-level)”. Obviously, indexing these events is of interest here. We provide therefore a flexible interface definition to the database in order to enable the plug-in of various multimedia annotation tools. In particular, we are working with Joanneum Research to integrate a Video Publisher. This tool enables us to annotate and publish video document like Word Document which render the cumbersomely semantic annotation process much easier. For more information on this tool, the reader is referred to `http://www.video-wizard.com/`.

Figure 4 shows a small extract of our multimedia database schema which is used in context with the multimedia indexing extension and their experimental results (see section 6) and which shall demonstrate the mapping strategy. The complete schema may be obtained from `http://www-itec.uni-klu.ac.at/~harald/codac/schema.pdf`. Please note that, this schema supports the MPEG-7 example, given in section 3.

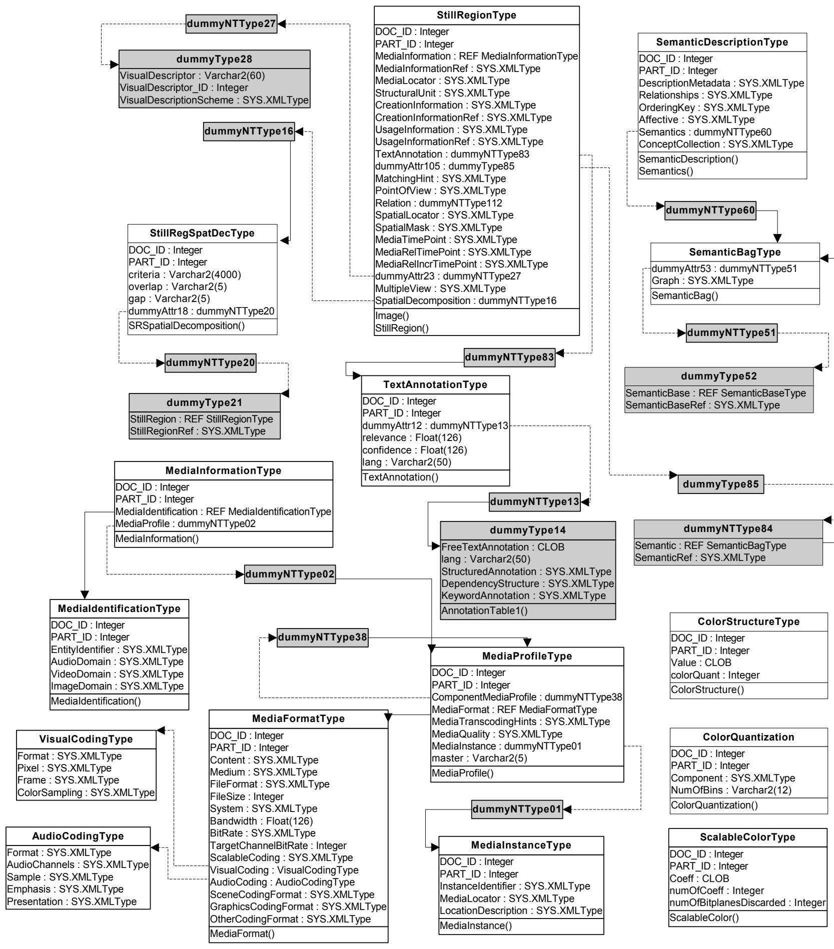

dummyNTType27
dummyType28
VisualDescriptor : Varchar2(60)
VisualDescriptor_ID : Integer
VisualDescriptionScheme : SYS.XMLType
dummyNTType16
StillRegSpatDecType
DOC_ID : Integer
PART_ID : Integer
criteria : Varchar2(4000)
overlap : Varchar2(5)
gap : Varchar2(5)
dummyAttr18 : dummyNTType20
SRSpatialDecomposition()
dummyNTType20
dummyType21
StillRegion : REF StillRegionType
StillRegionRef : SYS.XMLType
StillRegionType
DOC_ID : Integer
PART_ID : Integer
MediaInformation : REF MediaInformationType
MediaInformationRef : SYS.XMLType
MediaLocator : SYS.XMLType
StructuralUnit : SYS.XMLType
CreationInformation : SYS.XMLType
CreationInformationRef : SYS.XMLType
UsageInformation : SYS.XMLType
UsageInformationRef : SYS.XMLType
TextAnnotation : dummyNTType83
dummyAttr105 : dummyType85
MatchingHint : SYS.XMLType
PointOfView : SYS.XMLType
Relation : dummyNTType112
SpatialLocator : SYS.XMLType
SpatialMask : SYS.XMLType
MediaTimePoint : SYS.XMLType
MediaRelTimePoint : SYS.XMLType
MediaRelIncrTimePoint : SYS.XMLType
dummyAttr23 : dummyNTType27
MultipleView : SYS.XMLType
SpatialDecomposition : dummyNTType16
Image()
StillRegion()
SemanticDescriptionType
DOC_ID : Integer
PART_ID : Integer
DescriptionMetadata : SYS.XMLType
Relationships : SYS.XMLType
OrderingKey : SYS.XMLType
Affective : SYS.XMLType
Semantics : dummyNTType60
ConceptCollection : SYS.XMLType
SemanticDescription()
Semantics()
dummyNTType60
SemanticBagType
dummyAttr53 : dummyNTType51
Graph : SYS.XMLType
SemanticBag()
dummyNTType51
dummyType52
SemanticBase : REF SemanticBaseType
SemanticBaseRef : SYS.XMLType
dummyNTType83
TextAnnotationType
DOC_ID : Integer
PART_ID : Integer
dummyAttr12 : dummyNTType13
relevance : Float(126)
confidence : Float(126)
lang : Varchar2(50)
TextAnnotation()
dummyType85
dummyNTType13
dummyType14
FreeTextAnnotation : CLOB
lang : Varchar2(50)
StructuredAnnotation : SYS.XMLType
DependencyStructure : SYS.XMLType
KeywordAnnotation : SYS.XMLType
AnnotationTable1()
dummyNTType84
Semantic : REF SemanticBagType
SemanticRef : SYS.XMLType
MediaInformationType
DOC_ID : Integer
PART_ID : Integer
MediaIdentification : REF MediaIdentificationType
MediaProfile : dummyNTType02
MediaInformation()
dummyNTType02
MediaIdentificationType
DOC_ID : Integer
PART_ID : Integer
EntityIdentifier : SYS.XMLType
AudioDomain : SYS.XMLType
VideoDomain : SYS.XMLType
ImageDomain : SYS.XMLType
MediaIdentification()
dummyNTType38
MediaProfileType
DOC_ID : Integer
PART_ID : Integer
ComponentMediaProfile : dummyNTType38
MediaFormat : REF MediaFormatType
MediaTranscodingHints : SYS.XMLType
MediaQuality : SYS.XMLType
MediaInstance : dummyNTType01
master : Varchar2(5)
MediaProfile()
ColorStructureType
DOC_ID : Integer
PART_ID : Integer
Value : CLOB
colorQuant : Integer
ColorStructure()
VisualCodingType
Format : SYS.XMLType
Pixel : SYS.XMLType
Frame : SYS.XMLType
ColorSampling : SYS.XMLType
MediaFormatType
DOC_ID : Integer
PART_ID : Integer
Content : SYS.XMLType
Medium : SYS.XMLType
FileFormat : SYS.XMLType
FileSize : Integer
System : SYS.XMLType
Bandwidth : Float(126)
BitRate : SYS.XMLType
TargetChannelBitRate : Integer
ScalableCoding : SYS.XMLType
VisualCoding : VisualCodingType
AudioCoding : AudioCodingType
SceneCodingFormat : SYS.XMLType
GraphicsCodingFormat : SYS.XMLType
OtherCodingFormat : SYS.XMLType
MediaFormat()
ColorQuantization
DOC_ID : Integer
PART_ID : Integer
Component : SYS.XMLType
NumOfBins : Varchar2(12)
ColorQuantization()
dummyNTType01
AudioCodingType
Format : SYS.XMLType
AudioChannels : SYS.XMLType
Sample : SYS.XMLType
Emphasis : SYS.XMLType
Presentation : SYS.XMLType
MediaInstanceType
DOC_ID : Integer
PART_ID : Integer
InstanceIdentifier : SYS.XMLType
MediaLocator : SYS.XMLType
LocationDescription : SYS.XMLType
MediaInstance()
ScalableColorType
DOC_ID : Integer
PART_ID : Integer
Coeff : CLOB
numOfCoeff : Integer
numOfBitplanesDiscarded : Integer
ScalableColor()

Figure 4. Small Extract of the MPEG-7 Database Schema (Types and Tables)

The presented Multimedia Schema contains the mapping of an *MPEG-7 StillRegionType* which is a delegate for images in MPEG-7 (i.e., StillRegion denotes complete images and parts of them). In the database schema we created an object type of the same name (*StillRegionType*). Some of elements are declared as separate object types, some are defined by the specific *SYS.XMLType*. The decision which type to use was carried on importance for the querying process. For instance, the element of type *TextAnnotationType* was chosen to be detailed further, because it is of importance for free-text search in the database, while the *UsageInformation* was chosen to be declared as *SYS.XMLType*. The later type contain only few information for a concrete image, because meta-information on the usage may already be declared at the MPEG-7 root level, further it spans a DS which might contain many different descriptions with similar content. This would lead to many database object and tables, probably containing few content. Therefore, we decided to store the subtree of the MPEG-7 document containing *UsageInformation* directly in the database type/table. However, this information is not lost for querying. The reason is that the *XMLType* provides XPATH query functionality which enables one to reach elements and attributes of this document by XPATH. In other words, with a combined select and XPATH query any information may be reached. However, as pointed out earlier, important information can be reached directly through object navigation, as for instance from StillRegion to TextAnnotation.

Other object types are for *MediaInformation* (MediaInformationType, MediaProfile, MediaFormat etc.). They describe information on the coding, media attributes, locations and physical structure of the data. Semantic content of an image may be obtained through the Semantic reference to an *SemanticBagType* which is an abstract root object type for concrete semantic indexing classes, like events, places and time. Finally, the decomposition of the image (structural aspect) is specified by following the reference of the *SpatialDecomposition* element in the StillRegionType.

Further important object types are *ScalableColorType* and *ColorStrucutureType*. They are used to store the feature vectors of the color histograms extracted from the images described by a *StillRegion* and are indexed with the help of our *Multimedia Indexing Framework* (see subsection 4.2).

Finally, we had to introduce some dummy types. These stem from (m,n) relationships in the element associations of the MPEG-7 schema. They cannot be directly mapped to object associations and are represented as "dummy", i.e., not originally named in MPEG-7. In terms of tables, they are nested table in the respective parent table. An example is the *VisualDescriptor* which is an XML element collection in the MPEG-7 StillRegionType.

In order to store structured values, all relevant (queryable) types, have to be declared as tables. Table names for types are shown in the last line of each box defining a type. For instance for the *StillRegionType* we defined two tables *StillRegion()* and *Image()*. The later models the delegate functionality of the StillRegion DS in MPEG-7.

Example Mapping of an MPEG-7 Document This paragraph describes the most important steps that are used during insertion of our example MPEG-7 document (see figure 2 left side). The basic idea is to perform a post order traversal of the document‘s DOM tree (see figure 2 right side) which is created with the help of an corresponding XML parser. A leaf node is defined as either having no more child nodes (e.g.: FreeTextAnnotation, Note: the text element is not indicated as a node) or representing an XMLTYPE (e.g.: MediaLocator) which is inserted with all child elements. Besides the XMLTYPE and basic types, such as Integer, the insertion process has to identify the following types: *REF Type* represents a reference pointing to a row somewhere in the database, *Dummy Types* indicates nested tables and *Empty Types* meaning that the element has no corresponding column in the current table (e.g.: VisualDescriptor element of table StillRegion). The main part is to identify a node in the DOM tree that corresponds to a database table in our schema. After a successful identification the necessary insert statement has to be created. For this purpose we have to specify the table name the parameter list and their values. The attributes of the current node (identified as a database table) can be fetched with methods of the DOM API to retrieve all childs. The attribute name/value pairs are stored into the respective column and value list vectors. Finally, the complete insert statement can be build and executed. After a successful execution the algorithm proceeds with the next node in the post order traversal. The algorithm terminates as soon as the root element of the document is reached.
Looking at the example, that would mean that the tree is traversed along the leftmost subtree until the *MediaLocator* node has been reached. As its type is XMLTYPE, the traversal is continued at its sibling, *SpatialDecomposition*. Again, the tree is traversed until the next leaf node is reached, namely *FreeTextAnnotation*. The parent node is a possible insert candidate and therefore processed as described before. The other nodes considered immediately for being inserted are (in the order of traversal): *VisualDescriptor*, *StillRegion*, *SpatialDecomposition*, *Image*, *MultimediaContent*, *Description* and *Mpeg7*.

4.2. Multimedia Indexing Framework (MIF)

The Multimedia Indexing Framework (see figure 3 (a)) is divided into three modules. Each module may be used on its own and may be distributed over the network.

4.2.1. GistService

The GistService (see figure 3 (b)) is the part realized in the external address space and is implemented in C++. It runs as an own process (called service) in the Windows Operating System environment and manages all available access methods. The current version offers support for Generalized Search Trees (GiST, see section 5) and further access methods not relying on balanced trees (e.g., LPC-files[18] to support NN-search in high dimensional vector spaces.

The service is split into two main components: The *GistCommunicator* and the *GistHolder*. The *GistCommunicator* is a COM-object (Component Object Model) which offers services through an IDL interface (Interface Definition

Language). It is used for inter-process communication between the database (the GistWrapper shared library) and the implemented access methods. Thus, the *GistCommunicator* supplies the necessary functionality (e.g., creating, inserting, deleting) for accessing the index structures. The result of the operations are forwarded to the database.
It is the task of the *GistHolder* to manage all currently running index trees and the accesses to them. Each index tree is identified through a global and unique ID which is forwarded to the accessing process. For simplicity, the index trees are internally stored in an array, but this data structure can be replaced simply by any other more dynamic structure.

4.2.2. GistWrapper

The *GistWrapper* module is a in C++ implemented shared library that is used by the database to connect to the *GistService*. The library has two main tasks. First, it makes the GistService accessible for database procedures. An Oracle database has the possibility to call external C/C++ code via shared libraries. The GistWrapper acts as a wrapper for the GistService. The second task is the transformation of the input and output data to make it usable for both the GistService and the database. For instance, a simple C Char type has to be transformed into a BSTR string, or a VARIANT type into a String and so on. The GistWrapper module offers a similar interface as the GistService module.

4.2.3. Multimedia Index Type

The multimedia index type consists of several *indextypes*, their corresponding *operators* and the appropriate implementation (*objects*). We will illustrate the integration process with the example of an R-tree. For this purpose, we defined first a new Oracle *indextype*. Each *indextype* needs some *operators* that are offered by this type. Example of operators which we realized are the following:

– rt_equal_point(CLOB, CLOB, number, number);
– rt_nearest_point(CLOB, CLOB, number, number);

The first operator defines an equality search and the second one a nearest neighbor search for point data. The parameters are defined as follows: 1) element in the database table, 2) search item, 3) amount of results, and 4) the dimension.

In the second step we defined an object that delegates all necessary index methods (e.g., ODCIIndexInsert, ODCIIndexCreate, ...) to their corresponding implementations. In this example, most methods are forwarded to the GistService.

5. GIST FRAMEWORK

As mentioned above, our indexing framework, MIF, relies partially on the *GiST framework*. The theory and implementation of the GiST framework was developed by J. H. Hellerstein and his group at the University of California, Berkeley. The GiST framework enables a developer to plug-in new balanced tree implementations into the framework and to test their performance. This is done by implementing some specific methods for insertion, deletion and search. The current GiST version, 2.0, already includes source code[†] for the R-tree, R*-tree, SS-tree, SR-tree, SP-tree and B-tree.

Generally spoken, a GiST is a balanced tree with (key, RID) pairs in the leaves and (predicate, child page pointer) pairs as internal nodes. The framework and their corresponding trees have no restrictions on the key data stored within the tree or on their organization within and across nodes. Once a new access method is set up, its balance may be improved by an additional tool in the GiST framework environment, the amdb.[4] Amdb is a tool for designing, debugging, analyzing and performance measuring of GiST implemented access methods. The current GiST framework (version 2.0) has some shortcomings. First of all only one index tree may be used at time. Second, no support for additional non-balanced tree indexes is given.
The GiST framework is widely used in the CBR research,[15] but yet this framework has not been integrated into an extensible DBMS to support multimedia applications.

Finally, it is important to note that there has been extensive testing of the framework which is reported in,[25] also in combination with an CBR prototype called Blobworld.[5, 6] The efficiency of the GiST framework has been demonstrated in these previous works, yet its integration into a DMBS, together with the important multimedia extensions we made, has to be carefully evaluated for efficiency which is detailed in the next section 6).

[†]`http://gist.cs.berkeley.edu/`

6. EXPERIMENTAL RESULTS

This section describes a significant part of the series of experiments we performed in order to evaluate the effectiveness of our *Multimedia Indexing Framework* (MIF). The tests were carried out on two distinct datasets, one *synthetic (uniform dataset)* and one *real*. The experimental settings are as follows:

- The synthetic dataset contains 64 and 96 dimensional feature vectors that are represented as strings. The values were generated uniformly over the normalized [0..1] space.
- The real dataset was generated from an 1 hour and 46 minutes long movie, encoded with DIVX4. From the movie, we extracted 64 dimensional color histogram of 200000 frames of size 352x288 pixel, by retaining the two most significant bits in the RGB space. The generated feature vectors were inserted into the database. Further we separated color histograms of 100 frames for the query process.

In order to compare the built-in indexing mechanism and our MIF for efficiency we have to carry out exact match queries. The remaining supporting MIF retrieval functions, like reach search, NN-search and overlap search, are in addition to the build-in index functions.

The retrieval was carried out through server-sided JDBC, i.e., the java class resides in the database and are executed through Oracle's own JVM (Java Virtual Machine). The insertion was accomplished through a java class that resides outside of the database and was connected through thin JDBC. In both cases, we always used the same java classes for the measurements, thus the results are comparable.

6.1. Indexing Details

The feature vectors are stored in a column of type *Character Large Object (CLOB)*. We had to use the *CLOB* data type, because of the high dimensionality of the required data (e.g., multi-level feature vector of frames in MPEG movies). As a consequence, the build-in B-tree can not be used, as it does not handle CLOBs. In theory, the build-in B-tree should be able to index data up to 160 dimension (limited through the VARCHAR2 data type of Oracle), but in practice the dimension handled depends on the number of data points, for instance for a dimension of 2 (!) we are able to insert 2.8 millions, after this value the database crashed. Instead, we used the *Oracle Text Index* which is able to index CLOB string representations without severe limitations. Based on these index techniques we compared the response time between a normal (no index) solution, the Oracle Text Index and *MIF*. The response time was measured for insertion and query operations. The query operation was limited to exact match queries, because of the limited functionality of current database indexing mechanisms. As mentioned above this shortcoming can be compensated with MIF.

6.2. Detailed Results

The comparison of MIF using R-trees and Oracle 9i build-in Text Index[17] shows that the MIF based trees show less insertion efficiency (due to the external proc calls), but offer significantly higher query performance.

Indexing – Synthetic Dataset The following figures show the results for the insertion process: Left side of figure 5 for 64 dimensional point entries and right side of figure 5 for 96 dimensional point entries. These figures show that the MIF has a higher insertion time than the related solutions. The extra time for the MIF is caused by the overhead from switching and transferring the data between the Oracle address space and the external address space. This overhead does not penalize the MIF, because insertion are rare in our mostly read-only application context, and second the insertion does not explode even for a large data set.

It has to be noted that the memory consumption of the *Oracle Text Index* is enormous. The table space consumed is in the worst case over 3.8 GB for an insertion operation of 200.000 point elements with 96 dimensions. Compared to this, the memory consumption of the *MIF* is significantly smaller, e.g., for the same insertion operation as above, it requires only a table space of about 220 MB.

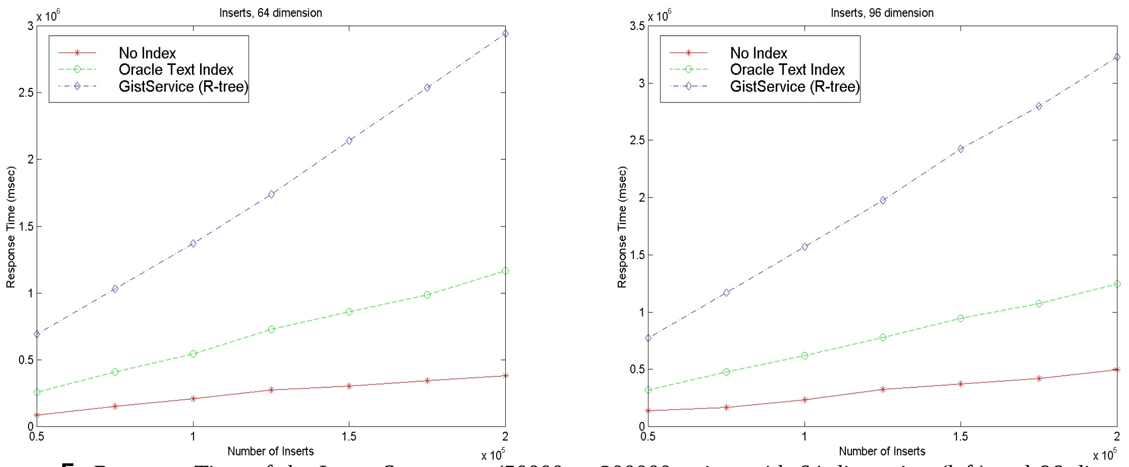


Figure 5. Response Time of the Insert Statements (50000 to 200000 points with 64 dimension (left) and 96 dimension (right))

Retrieval Query – Synthetic Dataset The results for the query evaluation show that our framework MIF outperforms clearly the related solutions. This is important, as in typical ad-hoc scenarios the query process is far more often used than the indexing process.

Left side of figure 6 shows that for exact match queries involving 64 dimensions, the *MIF* environment outperforms clearly both related solutions. The figures show the mean value of 5 measurements including each 100 select statements. A sequential search is, of course, significantly slower. Positively, the *MIF* environment is from 6 to 7 times faster than the Oracle Text Index for 175.000 data entries of 65 respectively 96 dimensions. Furthermore, MIF performs from 85 to 134 times better than the no index solution for the same amount of indexed elements.

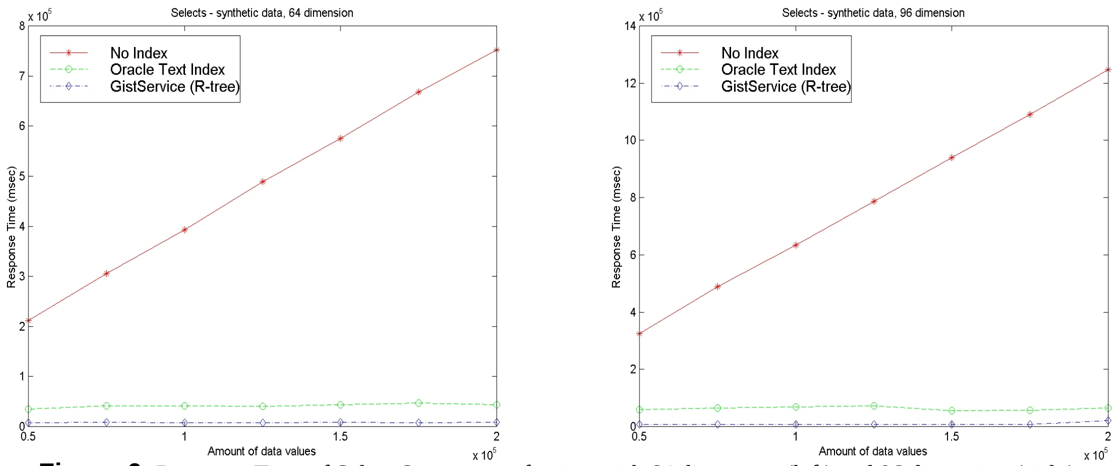


Figure 6. Response Time of Select Statements of points with 64 dimension (left) and 96 dimension (right)

Indexing and Retrieval – Real Dataset The response time for the insertion process with the real dataset was similar to the results obtained from the synthetic dataset test series and is presented in figure 7 (left side). The response time is again measured as the mean value of 5 measurements each including 100 select statements. The response time of the sequential scan was very similar to that measured for the synthetic data set (see figure 6). Moreover, in reason of the great discrepancy between the index and non-index solutions, the results of the non-index solution are not presented in this figure.

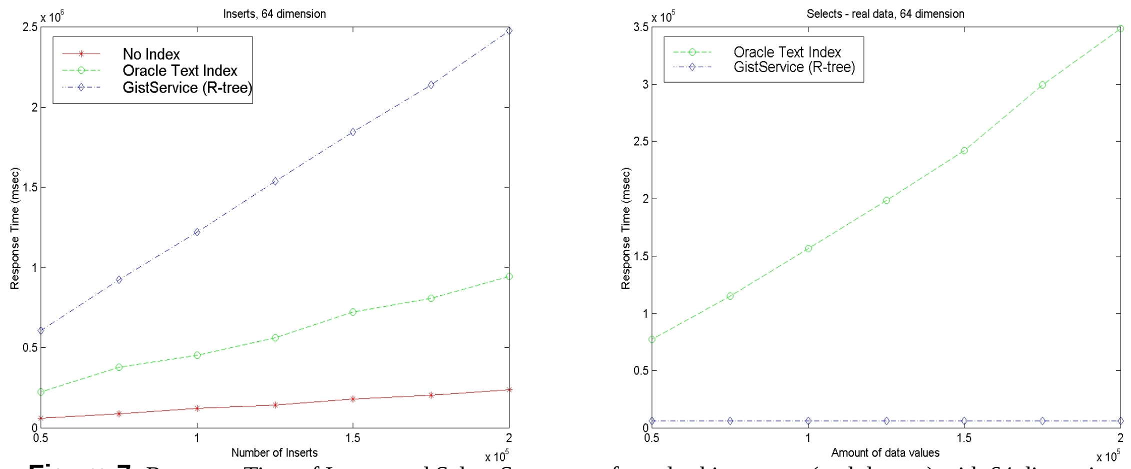


Figure 7. Response Time of Inserts and Select Statements for color histograms (real dataset) with 64 dimension

Figure 7 shows that the Oracle Text Index performs worse than in the previous test series. On average, we increased our efficiency by a factor of 6 compared to the synthetic data which leads to an overall improvement of 46 times faster than the Oracle Index (and this even with the call to an external address space) for 175000 indexed elements. Contrarily to the uniform dataset from above, the color histogram derived from this special movie contains far more zero values. This fact seems to shorten the use of the Oracle Text Index significantly.

7. CONCLUSION AND FUTURE WORK

This paper introduced the **MDC (Multimedia Data Cartridge)** as an MPEG-7 supported Database System Extension to an Oracle DBMS. It allows one to process (e.g., insert, query, retrieve) multimedia data more efficiently than related approaches (e.g., DMBS extender or CBR systems). Our approach introduced first a new database meta model for multimedia data based on the MPEG-7 MDS standard. Second, we proposed an extension of the indexing mechanism based on our new *MIF (Multimedia Indexing Framework)*. This framework allowed not only the execution of exact-match queries, but also supplied more multimedia specific operations, like range-search, NN-search, overlap etc., and meets thus more precisely the requirements of multimedia search and filter applications than the broadly used DBMS Extenders (e.g., DB2 Extenders). We furthermore showed that a new index type could be instantiated with only few steps using the Data Cartridge technology in combination with the GiST-framework. Finally, the experimental analysis showed that the build-in indexes of Oracle were outperformed for queries they may still handle (exact match). Moreover, the performance for an NN-search with MIF were comparable to those of the exact match. This showed together with our extension mechanisms (support for similarity search), the effectiveness of our approach.

The proposed methodology applies to other extensible object-relational DBMS as well, as long as they provide an extensible type system, query processing/optimization interface and access methods, as for instance IBM DB2 and IBM Informix do.

Future research will focus on integrating a cost-based query optimizer to our MDC by implementing the respective Cartridge interface. We will thereby rely on our preliminary work in.[28] Further work will also concentrate on integrating additional access methods into the MIF for supporting better the semantically meaningful queries (e.g., index the semantic relations). In this context we are also working on a query interface in order to admit a user friendly access to our MDC enhancement.

REFERENCES

1. Harald Kosch. MPEG-7 and Multimedia Database Systems. Sigmod Records, 31(2). June 2002.
2. Joseph M. Hellerstein, Jeffrey F. Naughton and Avi Pfeffer. Generalized Search Trees for Database Systems. In Proceedings of the 21st International Conference of Very Large Databases VLDB, pages 562-573, Zurich Switzerland 1995.
3. Marcel Kornacker. PHD Thesis: Access Methods for Next-Generation Database Systems. University of California at Berkley, 2000.
4. M. Shah, M. Kornacker, J. M. Hellerstein. Amdb: A Visual Access Method Development Tool. In Proc. User Interfaces To Data Intensive Systems, pages 130-140, Edinburgh, Scotland, 1999.
5. C. Carson, M. Thomas, S. Belongie, J. M. Hellerstein and J. Malik. Blobworld: A System for Region-Based Image Indexing and Retrieval. Third International Conference on Visual Information Systems, pages 509-517, Amsterdam, The Netherlands, June 1999. Springer Verlag ISBN:3-540-66079-8.
6. M. Thomas, C. Carson and J. M. Hellerstein. Creating a Customized Access Method for Blobworld, Proc. of the 16th Int. IEEE Conf. on Data Engineering, page 82, San Diego, CA, March 2000, IEEE Computer Society 2000.
7. Data Cartridge Developer‘s Guide. Oracle 9.2, Part No.: A96595-01, March 2002. `http://otn.oracle.com/docs/products/oracle9i/doc_library/release2/appdev.920/a96595.pdf`.
8. D. A. White and R. Jain. Similarity Indexing with the SS-tree. In Proc. of the 12th Int. IEEE Conf. on Data Engineering, pages 516-523, New Orleans, Louisiana 1996. IEEE Computer Society 1996, ISBN 0-8186-7240-4.
9. N. Katayama and S. Satoh. The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries. In Proc. of the 1997 ACM SIGMOD Int. Conf. on Management of Data, pages 369-380, 1997.
10. P. Ciaccia, M. Patella and P. Zezula. M-tree: An efficient Access Method for Similarity Search in Metric Spaces. In Proc. of the 23rd Int. Conf. on Very Large Data Bases, pages 426-435, Athens, Greece 1997. Morgan Kaufmann 1997, ISBN 1-55860-470-7.
11. S. Berchtold, D. A. Keim and H. P. Kriegel. The X-tree: An Index Structure for High-Dimensional Data. Proceedings of the 22nd VLDB, pages 28-39, Mumbai (Bombay), India August 1996. Morgan Kaufmann 1996, ISBN 1-55860-382-4.
12. K. I. Lin, H. V. Jagadish and C. Faloutsos. The TV-tree: An Index Structure for High-Dimensional Data. VLDB Journal 3(4), pages 517-542, 1994.
13. Volker Graede and Oliver Günther. Multidimensional Access Methods. ACM Computing Surveys, 30(2), pages 170-231, 1998.
14. C. Böhm, S. Berchtold and D. A. Keim. Searching in High-Dimensional Spaces - Index Structures for Improving the Performance of Multimedia Databases. ACM Computing Surveys 33(3), pages 322-372, 2001.
15. R. Bliujute, C. S. Jensen, S. Saltenis, G. Slivinskas. R-tree Based Indexing of Now-Relative Bitemporal Data. In Proc. of the 24th VLDB Conference, pages 345-356, New York, USA, 1998. Morgan Kaufmann 1998.
16. M. Kornacker. High-Performance Extensible Indexing. In Proc. of the 25th VLDB Conference, pages 699-708 Edinburgh, Scotland, 1999. Morgan Kaufmann 1999.
17. Oracle Text. An Oracle Technical White Paper, May 2001. `http://technet.oracle.com/products/text`.
18. G. Cha, X. Zhu, D. Petkovic and C. Chung. An Efficient Indexing Method for Nearest Neighbor Searches in High-Dimensional Image Databases. IEEE Transaction on Multimedia 4(1), pages 76-87, March 2002.
19. M. Flickner, H.Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele and P. Yanker. Query by image and video content: The QBIC system. IEEE Computer, 28(9):23-32, Sept. 1995.
20. M. A. Stricker and M. Orengo. Similarity of color images. In Storage and Retrieval for Image and Video Databases, SPIE, pages 381-392, San Jose, CA, 1995.
21. J.M. Martinez. Overview of the MPEG-7 standard, v8.0. ISO/MPEG N4980, MPEG Requirements Group, Klagenfurt 2002. Available at `http://mpeg.telecomitalialab.com/standards/mpeg-7/mpeg-7.htm`.
22. J. M. Martinez, R. Koenen and F. Pereira. MPEG-7. IEEE Multimedia, 9(2), pages 78-87, April-June 2002.
23. R. Tusch, H. Kosch and L. Boeszoermenyi. VIDEX: An Integrated Generic Video Indexing Approach, In Proc. of the ACM Multimedia Conference 2000, pages 448-451, Los Angeles, USA.
24. M. R. Mediano, M. A. Casanova and M. Dreux. V-Trees - A Storage Method for Long Vector Data. In Proc. 20th International Conference on Very Large Data Bases, pages 321-330, Santiago, September 1994.
25. Nathan G. Colossi and Mario A. Nascimento. Benchmarking Access Structures for High-Dimensional Multimedia Data. Technical Report 99-05. Department of Computing Science University of Alberta, Canada. 1999.
26. S.-C. Chen, R. L. Kashyap and A. Ghafoor. Semantic Models for Multimedia Database Searching and Browsing. Kluwer Press, 2000.
27. A. Yoshitaka and T. Ichikawa. A Survey on Content-Based Retrieval for Multimedia Databases. IEEE Transactions on Knowledge and Data Engineering, 11(1), pages 81-93, 1999.
28. S. Atnafu, L. Brunie and H. Kosch. Similarity-Based Operators and Query Optimization for MMDBMS. In International Database Engineering and Application Symposium (IDEAS) 2001, IEEE CS Press, pages 346-355, Grenoble, France, 2001.