A Data Management Layer for Visual Information Retrieval

Horst Eidenberger and Roman Divotkey
Vienna University of Technology
Institute of Software Technology and Interactive Systems
Favoritenstrasse 9-11, A-1040 Vienna, Austria
+43-1-58801-18853
{eidenberger, divotkey}@ims.tuwien.ac.at

ABSTRACT

This case study describes the data management layer of the VizIR visual information retrieval project. VizIR is an open source framework of software tools for visual retrieval research. In content-based multimedia retrieval media objects are described by high-dimensional feature vectors. These feature vectors have to be stored in an efficient way in order to accelerate the retrieval process. VizIR database management is based on object-oriented persistence management. The database interface has a three tier architecture: a pattern-based persistence system hides the underlying database, an object-relational mapping system maps classes to entities and a relational database provides state-of-the-art database features (transactions, integrity, recovery, etc.). The described database management prototype can be downloaded from the VizIR project website.

Categories and Subject Descriptors

H.2.4 [**Database Management**] Systems – *Multimedia databases, object-oriented databases, relational databases*.
H.2.8 [**Database Management**] Database Applications – *Data mining, image databases*.

General Terms

Management, Performance, Design, Reliability, Experimentation.

Keywords

Content-based Visual Information Retrieval, Video Retrieval, Image Retrieval, Object-oriented Database Design, Database Management, Persistence Management, High-dimensional Indexing, Multimedia Databases.

MDM/KDD'04, August 22, 2004, Seattle, WA, USA.

1. INTRODUCTION

Content-based visual information retrieval (VIR) is a field of multimedia research that aims at extracting meaningful (semantic) media information directly from the pixel level. Sophisticated algorithms (e.g. the MPEG-7 visual features [2, 7]) are used to locate relevant information (features, descriptors) in media objects. Usually, features are represented as high-dimensional data vectors. For example, if all visual MPEG-7 features are used to describe a media object, the data vector has more than 320 dimensions. Dis-similarity of media objects is measured as distance between feature vectors. See [3, 6, 8] for more information on content-based visual information retrieval.

The fundamental database problem of VIR is to establish the efficient storage of feature vectors in order to enable fast (but still flexible) content-based multimedia data mining. This case study describes the approach we implemented to solve this problem in the VizIR project [4]. VizIR aims at developing a software workbench of free tools for content-based image and video retrieval (see Section 2 for more information on VizIR). Below, we discuss general approaches for VIR database design, describe and argue for our design decision and give details on the concrete implementation in the VizIR framework (freely available from [10]).

The paper is organised as follows. Section 2 sketches the VizIR project. Section 3 points out principal data models for feature data. Section 4 describes the VizIR data management model. Finally, Section 5 describes selected implementation issues.

2. BACKGROUND: THE VIZIR PROJECT

Even though significant amounts of research on VIR have been conducted in recent years and a considerable number of research prototypes has been developed (see [8] for a quick overview), there is still no VIR software framework available that would satisfy the researchers' needs. Firstly, as similar methods are used for image and video retrieval, it would be desirable to support both media types in one environment. Furthermore, it would accelerate research work, if state-of-the-art VIR components (e.g. space to frequency transformations, kernel-based learning algorithms, user interfaces) would be readily available in an homogeneous environment.

With the VizIR project we are intending to satisfy these demands. VizIR is a framework of resources (mainly software components implemented in Java) that are needed to build VIR prototypes. The software components include classes for media access, transportation and visualisation in user interfaces, for feature extraction (including the content-based MPEG-7 descriptors), for querying and refinement based on a novel 3D retrieval and browsing panel, for user interface design, and for visualisation of media metadata, evaluation and benchmarking. As the framework itself and all elements have to be extendible, it is imperative that the underlying database system does not make any assumptions on the elements' structure in order to keep them persistent. This constraint drives the database design considerations presented in Section 4.

VizIR is an open project and all components are free under GNU General Public License. See [4] for a more detailed description on the VizIR project. All finished components (including the database layer presented in this paper) can be downloaded as source code from the project website [10].

3. RELATED WORK: DATABASE MANAGEMENT FOR FEATURE DATA

One scientific challenge of VIR is the high dimensionality of feature vectors. For example, if all content-based MPEG-7 descriptors are used to describe an image, the description has more than 300 dimensions. Solving the dimensionality problem adequately must be one of the first issues in designing a VIR system. Still, it is mandatory for the success of VIR in general and the VizIR project in particular that the database layer meets a number of software engineering requirements: Database access has to be simple, efficient, domain-independent and operating system-independent. Additionally, the database management system has to provide traditional features (integrity, recovery, etc.). Before we designed the VizIR database layer we surveyed approaches that were used in existing VIR systems or suggested for the future.

Classic RDBMS (e.g. DB2 in QBIC [8]) fulfil all software engineering requirements easily. If used, media objects are usually stored externally, feature vectors are stored as BLOBs (often in one table per feature) and indexed by context-free structures (e.g. B-trees). Therefore, the data can only be accessed sequentially (by ID). More sophisticated access methods (such as dis-similarity measurement by distance functions; for example, implemented as stored procedures) cannot be used. Fine-granular access would only be possible, if feature vector elements could be assigned to table attributes. This is usually impossible as many features have varying length.

In recent years, sophisticated indexing structures have been developed for multimedia RDBMS (see [1] for an overview). Various R-trees, SS-trees, etc. have been proposed to allow for efficient organisation and access to high-dimensional media data. Ideally, raw media data would be stored outside the database. Feature metadata should be stored in fine granulation in the database to enable context-specific indexing. If multimedia indexing structures do exist, feature data can be selected using distance functions. Unfortunately, a number of drawbacks are connected to this approach. Firstly, most indexing structures have the tendency to become inefficient for really high-dimensional data (in the MPEG-7 case: 320+ dimensions). Secondly, most indexing structures are unable to deal with multiple distance measures in one index (state-of-the-art in content-based retrieval). Thirdly, as for classic RDBMS it is mostly impossible to define a mapping from feature vector elements to entity attributes. Finally, multimedia indexing structures are hardly implemented in classic RDBMS and more specialised products are often not operating system-independent or do not provide traditional RDBMS features.

As it is very difficult to press polymorphic feature data in relational databases in fine-granular manner, we searched for alternative approaches of data representation. XML databases seem to provide ideal structures and properties for VIR data. Features can easily be mapped to XML documents (e.g. MPEG-7 defines an XML representation of its visual features). Media objects are per se separated from metadata and stored externally. All data points can easily be accessed by using document models and (simple) querying languages (e.g. W3C DOM and XPath).

One VIR-specific example for this group of systems is the PTDOM database [11]. PTDOM defines a document object model specific for the MPEG-7 features. All features (including those based on MPEG-7 types: vector, matrix) can be accessed on a fine-granular level and retrieved using XPath and database-internal user-defined functions (similar to stored procedures). Data elements can be indexed by B-trees. Of course, additionally, more sophisticated multimedia indexing structures could be implemented as well. The main drawback of PTDOM, in terms of practical application, is that the currently available implementation is strongly bound to commercial, operating system-dependent helper libraries.

The last VIR-specific approach that may become relevant in the future is the media mediator concept [9]. Media mediators are functions that are used to access media data live during a query. Conceptually, media mediators are defined on a semantic level and mapped to low-level features that extract information from the media samples. Theoretically, media mediators can be used to define arbitrary operations on media data but, as well, to implement distributed querying environments. The advantages of the media mediator concept are that everything is done on the fly and media objects are accessed in a fine-granular way. On the other hand, obviously, the comprehensive operations needed to implement mediators would be extremely resource-consuming. Additionally, it would be almost impossible to accelerate the querying process using indexing structures. These drawbacks make it unlikely that the media mediator concept can ever be implemented in its original form. Still, if particular operations could be identified as basic building blocks for media mediators, these operations could be computed prior to query execution. Hereby, the querying process could be dramatically accelerated while the flexibility of the concept would be largely preserved.

4. VIZIR DATA MANAGEMENT MODEL

Below, we describe the data management model we designed for the VizIR project from the described palette of approaches. Subsection 4.1 describes the design decision. Subsequent subsections describe all relevant aspects of the VizIR data management model.

4.1 Use case-driven design decision

Surveying principal VIR approaches showed that we could

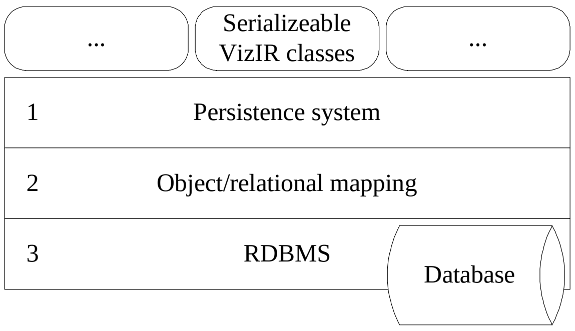

Figure 1. Layer structure of VizIR persistence system.

basically choose between a classic RDMBS (with self-implemented multimedia indexing structures) and an XML database. As VizIR is a software engineering project, we decided to follow a best practice and perform the database decision use case-driven.

VizIR is intended for general purpose VIR. For practical applicability it should provide reliable state-of-the-art persistence management. These requirements are best satisfied by classic RDBMS. An XML database would be a good choice, because the (implemented) visual MPEG-7 features are available as XML documents. Additionally, most feature structures can easily be represented in XML form. On the other hand, even professional XML databases have serious problems with handling large XML documents. Generally, implementing multimedia indexing structures would hardly make sense, since most features require variable distance measures. In this situation, an index would have to be defined for every distance measure used in the retrieval process. Obviously, following this approach would result in significant overload of indexing metadata. Furthermore, some distance measures used in VIR are not based on metrics and, in particular, do not meet the triangle inequality requirement. For these measures it would be even more difficult to define an index. Moreover, feature structures can be organised arbitrarily (e.g. as matrices). Additionally, in many retrieval situations, the query engine has to browse through the feature vectors sequentially anyway.

Therefore, we decided that VizIR should be grounded on a relational database and indexing structures should be implemented (if required) on the application level. Since VizIR is based on the query-by-example paradigm, low-level indexing in relation to a pre-defined origin (e.g. the zero vector of distance space) would not be feasible. An index would be required for every query example. However, variable indexing concepts on the application level (e.g. heuristics) may result in valuable query acceleration.

In order to guarantee application independence and framework extendibility we decided to employ object-oriented persistence management and to map serialised software objects to tables of a relational database. Figure 1 depicts the resulting three layer structure: The persistence system layer provides the methods needed to access the database (storage and retrieval), the mapping layer maps objects to entities and the database layer provides transactions, integrity and recovery. The advantages of this solution are that (1) any mapping tool and any database can

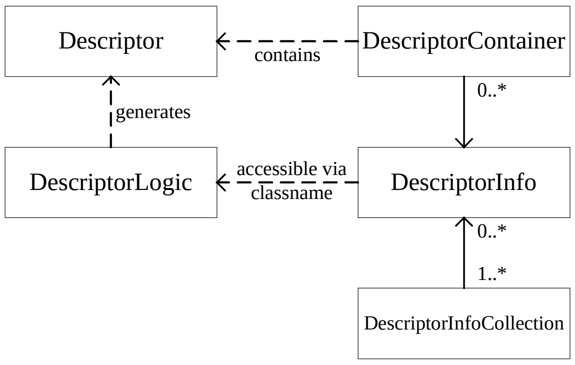

Figure 2. Descriptor-related entities (simplified).

be used behind the persistence system API, (2) any serialisable object can easily be made persistent and (3) database management is fully transparent to the rest of the VizIR framework.

4.2 VizIR entities

Generally, the VizIR persistence management system needs to store media-related and descriptor-related data. For media objects, just the visual data and some textual metadata are stored. The structure needed for descriptor-related data is shown in Figure 2 (in UML syntax). It is required both on the database level (as entities) and on the application level (as classes).

The main class is *DescriptorInfo*. This class holds the management methods for the other components. *DescriptorLogic* contains the extraction algorithms. *DescriptorLogic* may have an arbitrary structure: as it is stateless, it is not made persistent. The actual (XML) descriptor data are held in *Descriptor*. Since descriptors may have widely varying appearances, each *Descriptor* is encapsulated by a *DescriptorContainer*. As this class has a pre-defined, fixed structure, it can easily be made persistent (see Section 5). Additionally, every *Descriptor* may belong to a group (e.g. an MPEG-7 descriptor scheme). This relationship is implemented in *DescriptorInfo* and *DescriptorInfoCollection*.

Even though we did not have this generality in mind when we designed the VizIR persistence manager, the presented model is flexible enough to hold any type of feature data for any type of media. It could, for example, be employed to manage content-based features of audio streams or text features of arbitrary media objects.

4.3 Persistence management layer

The persistence management layer is responsible for offering all database-relevant methods to the VizIR framework while hiding the concrete implementation of the object-relational mapping and the database. Figure 3 illustrates the implemented model. The chosen design follows state-of-the-art software design patterns.

The main class *PersistenceSystem* is responsible for initialisation and the creation of all database-related entities (media objects and descriptors). Additionally, it contains a factory class for the creation of *PersistenceManager* classes (*PersistenceFactory*). *PersistenceManager* encapsulates all methods needed for database access and transaction management. This class is used

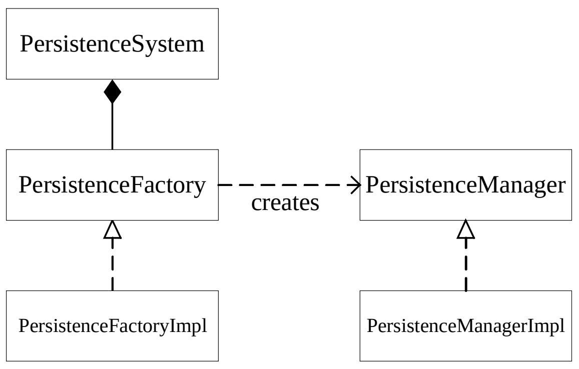

Figure 3. Persistence management classes.

to put VizIR objects under persistence control, reload objects from earlier instances and retrieve collections of objects by name. Currently, the persistence manager supports only direct queries by ID (e.g. descriptor class name). Joins can be used to retrieve, for example, all feature vectors for one media object or all media objects of a particular media collection. Generally, the level of sophistication of the querying components depends on the object-relational mapping tool.

In order to guarantee the exchangeability of the underlying mapping system, the persistence management classes implement the Bridge pattern: *PersistenceFactory* and *PersistenceManager* are just interfaces that define an API. The classes implementing these interfaces are dependent on the mapping layer. The factories *PersistenceSystem* and *PersistenceFactory* are responsible for instantiating the right implementing classes for a particular configuration of mapping layer and database.

5. IMPLEMENTATION

The Java implementation of the VizIR persistence management system makes use of the Hibernate system on the mapping layer [5]. Hibernate was selected, because it supports a wide range of commercial and open source database systems (including Oracle, DB2 and MySQL), provides powerful querying mechanisms and employs the Java Reflection API to analyse the structure of software classes. Furthermore, it is, like VizIR, an open source project that is published under GNU LGPL.

Classes that are made persistent using Hibernate have to meet a few requirements: A default constructor (without parameters, e.g. *newInstance()*) has to exist for each class and accessor methods (*get/set*) have to be available for every resource. These methods are used through the Reflection API. Optionally, every class should have an ID tag. Only two bits of information have to be provided externally: the mapping of resources to database data types and the primary/foreign key references in *1:n* and *n:m* relationships. This information is provided in simple XML documents. Even though it is possible to inform Hibernate about relationships of entities, the system leaves maintenance of referential integrity (at least of *n:m* relationships) to the user. Integrity can be achieved by implementing the *Lifecycle* interface and callback methods for data manipulation events (e.g. *onDelete()*).

We are making use of the properties of the Hibernate system to store arbitrarily shaped feature data in the database without the need to define mappings for every new *Descriptor* class: The mapping is defined for the resources of *DescriptorContainer*. Feature vectors (*Descriptor* objects) are properties of this class.

6. CONCLUSIONS AND FUTURE WORK

We tried to identify the most practicable database solution for a content-based visual information retrieval system that does neither make assumptions on features used nor on application domains. The VizIR framework is intended to be a modern, usable workbench for visual information retrieval research. Hence, grounding the system on a flexible and robust database layer was mandatory. It is interesting to notice that the best solution turned out to be a classic relational database in combination with an object-oriented persistence manager. Using the described design, VizIR can deal with arbitrary feature data and database systems. The programming effort for the VizIR user is reduced to a minimum. Actually, the VizIR persistence layer can be used to manage media objects and metadata (text or binary) of any kind. It is free software and can be downloaded from [10].

Future work will include performance tests with large MPEG-7 test datasets as well as architecture tests with mapping tools and database systems not considered so far.

7. ACKNOWLEDGMENTS

The authors would like to thank Christian Breiteneder for his valuable comments and suggestions for improvement. The VizIR project is supported by the Austrian Scientific Research Fund (FWF) under grant no. P16111-N05.

8. REFERENCES

[1] Böhm, C., Berchtold, S., and Keim D.A. Searching in High-Dimensional Spaces-Index Structures for Improving the Performance of Multimedia Databases. ACM Computing Surveys 33, 3 (2001), 322-373.

[2] Chang, S.F., Sikora, T., and Puri A. Overview of the MPEG-7 Standard. IEEE Transactions on Circuits and Systems for Video Technology 11, 6 (2001), 688-695.

[3] Del Bimbo, A. Visual information retrieval. Morgan Kaufmann, San Francisco CA, 1999.

[4] Eidenberger, H., and Breiteneder, C. VizIR – A Framework for Visual Information Retrieval. Journal of Visual Languages and Computing 14, 5 (2003), 443-469.

[5] Hibernate project website. http://www.hibernate.org/.

[6] Lew, M.S. (ed.) Principles of Visual Information Retrieval. Springer, Heidelberg, Germany, 2002.

[7] Manjunath, B.S., Salembier, P., Sikora T. Introduction to MPEG-7. Wiley, San Francisco CA, 2002.

[8] Marques, O., and Furht, B. Content-Based Image and Video Retrieval. Kluwer, Boston MA, 2002.

[9] Santini, S., and Gupta, A. Mediating Imaging Data in a Distributed System. in Proceedings of SPIE Electronic Imaging Symposium, Storage and Retrieval Methods and Applications for Multimedia (San Jose CA, January 2004), SPIE, 365-376.

[10] VizIR project website. http://vizir.ims.tuwien.ac.at/.

[11] Westermann, G.U., and Klas, W. A Typed DOM for the Management of MPEG-7 Media Descriptions. Multimedia Tools and Applications, to appear.