Research Friendly MPEG-7 Software Testbed

F.-C. Chang, H.-M. Hang, and H.-C. Huang[a]

[a]Department of Electronics Engineering, National Chiao Tung University (NCTU), 1001 Ta-Hsueh Road, Hsinchu, Taiwan 30010, R.O.C.

ABSTRACT

We design and implement a research friendly software platform, which aims at the flexibility and the abstraction of MPEG-7 application prototyping. We studied and analyzed the MPEG-7 standard, including a typical scenario of using MPEG-7. In order to fulfill to needs of researches, in addition to the normative parts of MPEG-7, additional requirements are included. By examining these requirements, we propose a research friendly software platform. The architecture consists of a framework, utility units, and the descriptors. Because this system is implemented using Java, it also incorporates the features of the Java environment, and thus it is flexible for developing new components and prototyping applications. We demonstrate the flexibility of this testbed by constructing an example program which allows users to manipulate image related descriptors.

Keywords: MPEG-7, testbed, platform, feature extraction, matching

1. INTRODUCTION

With the sweeping development and deployment of computers, the widespread use of the internet, and the advances of the audio-visual technology, it is easy to produce, distribute, and consume numbers of multimedia contents.[1,2] The enormous amount of digital contents hence make searching and retrieval a difficult task. The emergence of MPEG-7 aims at solving the problems for multimedia searching and retrieval.

MPEG-7,[3,4] also called *Multimedia Content Description Interface*, is an efficient tool for searching and retrieval of multimedia data. MPEG-7 specifies a standard way of describing various types of audio-visual information irrespective of its representation format and storage support.

The objective of MPEG-7 is to standardize content-based description for various types of audio-visual information. It enables fast and efficient content searching, filtering, and identification. Therefore, MPEG-7 provides a rich set of standardized tools to describe multimedia content[5] with a minimum set of tools, called the normative components, essential for interworking, including *Descriptor* (D), *Description Scheme* (DS), and *Description Definition Language* (DDL).[6] A Descriptor is a representation of a distinctive characteristic of the data. A DS specifies the structure and semantics of the relationships between its components, which may be Descriptors or Description Schemes. The DDL is a language that allows the creation of new DSs and Ds, and the extension and modification of existing DSs.

In addition to these standardized tools, the MPEG-7 eXperimentation Model (XM)[7] is a reference software used to evaluate the standard Ds and DSs. It is a collection of programs contributed by the members of MPEG standard committee. Originally, each piece in it implements a straightforward MPEG-7 application type, which generates a single feature and lists the matching results.

For real world applications, both the description generation, such as feature extraction, and consumption, such as search engine, are essential but they are non-normative parts of MPEG-7 (Fig. 1). We thus are motivated to design a platform that allows MPEG-7 researchers to develop various features and algorithms on this platform. Users can manipulate the MPEG-7 descriptors in the abstraction layer without concerning of the

Further author information: (Send correspondence to H.-M. Hang)
H.-M. Hang: E-mail: hmhang@cc.nctu.edu.tw, Telephone: +(886)3-5731861, FAX: +(886)3-5723283/5731791
F.-C. Chang: E-mail: u8811833@cc.nctu.edu.tw
H.-C. Huang: E-mail: huangh@cc.nctu.edu.tw

Image and Video Communications and Processing 2003, Bhaskaran Vasudev, T. Russell Hsing, Andrew G. Tescher, Touradj Ebrahimi, Editors, Proceedings of SPIE–IS&T Electronic Imaging, SPIE Vol. 5022 (2003) © 2003 SPIE–IS&T · 0277-786X/03/$15.00

low-level functions, such as system calls and database connections. They can dynamically assemble different algorithms together, execute them, and view the generated data in various formats by visual viewer tools. Also, they can compose a new searching scheme (search engine) and test its feasibility on this testbed.

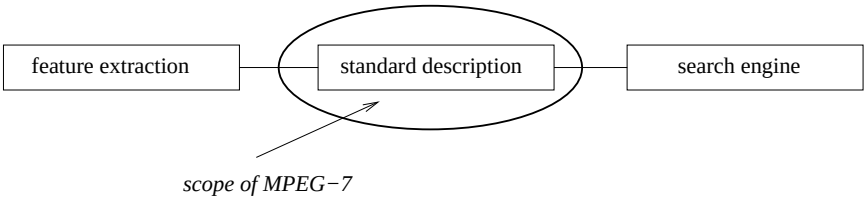


Figure 1. Scope of MPEG-7

This paper is organized as follows. Sect. 2 describes the motivation behind this project. Sect. 3 proposes the software architecture of this research friendly platform. The detailed system design and architecture implementation are described in Sect. 4. An application example is presented in Sect. 5. Sect. 6 concludes this paper.

2. MOTIVATIONS

Many different types of applications can be developed based on the MPEG-7 specifications. For example, a multimedia search engine typically has its own scheme in organizing and processing the meta-data. This scheme may comprise several standard Ds and DSs, but it is likely that we need to create application-specific descriptions and operations, and then to evaluate their performance. As discussed earlier, the MPEG-7 XM is a collective software set. Although it includes a few basic tools that can incorporate new Ds and DSs, it is mainly organized to demonstrate the extraction/matching results of using individual Ds (and/or DSs) and it has no database support. The database support is essential for a typical multimedia database application.[8] For research purposes, we need a platform that not only produces algorithm outputs, but also helps us analyzing the performance of an algorithm. We group the MPEG-7 related research topics into two categories. One category is developing a new description and its companion extraction and matching algorithms; the other is developing a new application, which is a composition of existing descriptions and algorithms.

For new description development, what we care is how useful the description is, and how well the extraction/matching algorithm performs. It is helpful if we have an environment that allows us to add and remove description data structures and algorithms easily. We evaluate the newly designed structures or algorithms by their experimental results. And the results need to be examined by various data visualization viewers.

For a new application development, we usually pick up several existing descriptions from a library, organize them into a scheme, and evaluate the new scheme to check its performance. A simple combination of the components, each designed originally for a single objective, may not perform well enough to achieve our goal. A carefully arranged processing sequence together with dedicated database schema design may greatly enhance the overall performance. After designing the application-specific descriptions and algorithms, we also need a versatile interface for handling the inputs and outputs of MPEG-7 streams.

3. ARCHITECTURE

Motivated by the discussions in Sect. 2, we start to design a research software platform. The design process to be elaborated below includes defining the requirements, analyzing the use-cases, and finally developing an architecture of the testbed. The unified modeling language (UML),[9,10] developed by the Object Management Group (OMG),[11] is a graphical language for visualizing and documenting a software intensive system. We will use UML to design our platform throughout this paper.

3.1. Requirements of the Testbed

The platform should provide an environment for both developing descriptors and feature-based applications; correspondingly, the users of the system are classified into two categories. We call the descriptor developers the "designers", and call the application developers the "developers". The common requirement for both groups of users is a flexible and extensible environment for conducting MPEG-7 related researches that involve algorithm simulation and data analysis.

The designers develop descriptors and their companion algorithms, such as extraction, matching, encoding, and decoding. We need a well-organized architecture that enables the designers to design the structure of descriptions, the algorithms to extract features from media contents, the algorithms to compute the similarity measure between two objects, and the encoding/decoding algorithms. In some cases, designers may want to compare different algorithm implementations for the same descriptor, *e.g.*, distance calculation based on the quantized values or the reconstructed values. Using this platform, designers can test their descriptors and algorithms by the quick prototyping mechanism.

In addition to evaluating the correctness of algorithms, the designers may want to evaluate the efficiency or the fitness of the algorithms. Sometimes we can rate the efficiency of a descriptor by examining the query results. For some descriptors, subjective examination of the extracted feature values may be helpful. To facilitate examining various types of descriptors, feature visualization components are necessary in our platform.

A designer concentrates on a single descriptor design, while a developer focuses on a larger scale of software organization. The typical scenario for a developer is picking up desired descriptors and algorithms from a library, preparing the input/output interface, organizing the operations, attaching the visualizers, and building the main program. The scenario suggests that the developers need an architecture that enables easy composition of the ready-to-use components developed by the designers.

In developing and prototyping MPEG-7 applications, the platform should be flexible for integrating various components into a complete system. This requires not only a structure that holds all the components, but also a management mechanism that gets/puts components in an organized fashion. Because a large percentage of applications are search-engine-like, the condition of multi-user accesses should be considered. This implies that the platform should be able to evaluate the performance when a new scheme is involved in concurrent processes. The database support, or the persistence mechanism, is critical in developing an application. It may not be a critical element for a designer, because a designer usually deals with a small set of test data. For a real application, a developer should concern the organization of the database; otherwise, the execution performance may degrade drastically with a large data set.

3.2. Use-Cases

A use-case diagram specifies the behaviors of a system and also it captures the expected interactions that occur at run-time. It is important in that it serves to validate the software architecture (which will be discussed in Sect. 3.3) during project development.[9] The requirements of the testbed suggest many use-cases between the system (the testbed) and the actors (the users). According to the classification of users discussed in Sect. 3.1, we identify three types of actors in the use-case diagram: the user, the designer, and the developer. A user, no matter he/she is a designer or a developer, may want to decode the input media data, extract some descriptors, and calculate the distances among descriptors. Since our system views the data structures and algorithms as components, the user needs to execute the component fetching operation to get the desired components before he/she can really perform the other operations.

In the following paragraphs, we will discuss the designer/developer specific use-cases. Fig. 2 shows the use-cases for designers. A designer performs two types of specific operations: one is to register the developed components, and the other is to view a descriptor by a visualizer. Examining the requirements, we know that the designer produces three types of components. The first is the descriptor. It is a pure data structure that holds the extracted multimedia features. The second type of component is the (data) associated algorithms, such as extraction, matching, encoding, and decoding algorithms. The last one is the descriptor visualizer (we call it "viewer" in the following text for abbreviation). When the designer registers a component to the system,

the system also records the associations related to a component. For example, we register a vector viewer with associations connected to the histogram descriptor. Then, the system records that the vector viewer is an available viewer of the histogram. Without the association records, the system is nothing but simply a component holder.

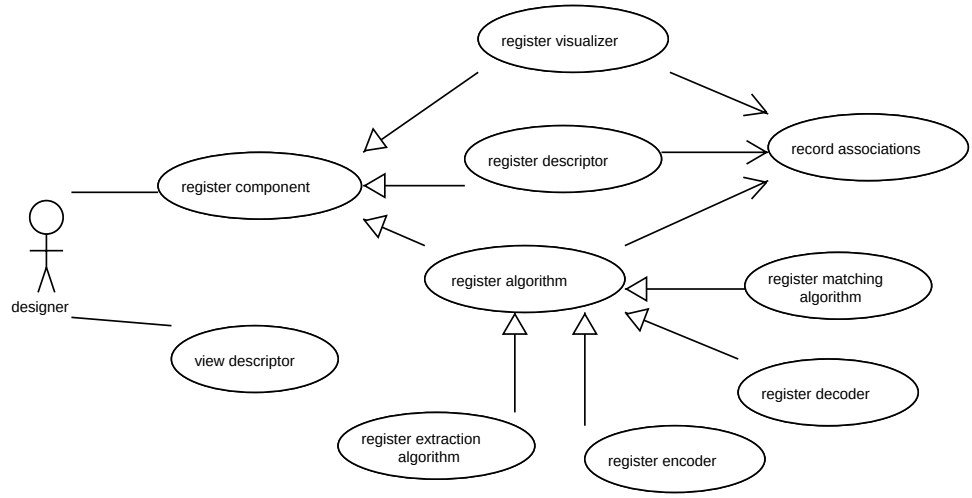


Figure 2. Use-Case diagram for designers

Fig. 3 illustrates the use-cases for developers. A developer performs three specific types of operations. The developer may want to extract application-specific features, calculate the similarity between two features, and export the feature values according to an external format. For example, a developer extracts the color histogram of a query image, transforms the internal feature representation to MPEG-7 format, dispatches the MPEG-7 feature to a search engine, and searches the best matches by comparing features in a local database. It is worth noting that the extraction and matching operations are different from those of general users in two aspects. First, the "features" that a developer requests is usually a composition of descriptors residing on the platform. Second, the extraction/matching algorithms are application-specific, because the MPEG-7 standard does not specify any method of combining multiple-descriptor algorithms.

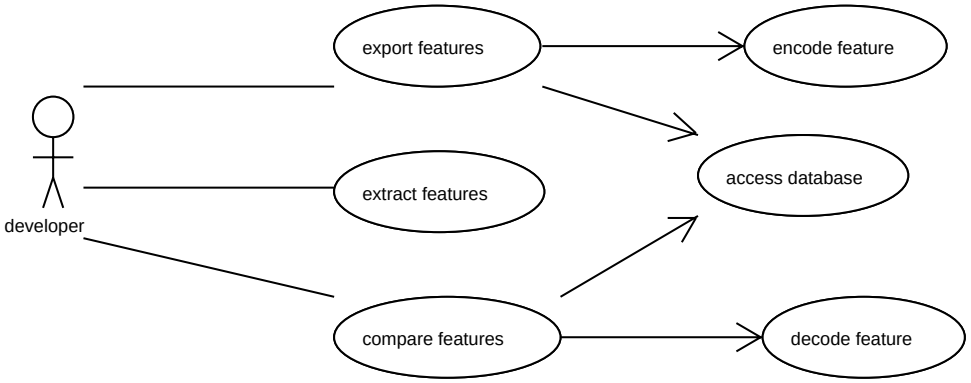


Figure 3. Use-Case diagram for developers

3.3. System Architecture

Before we design and implement the detailed structure of the system, we need an architectural view of the platform. The requirement and use-case analysis is focused on the behavioral view of the system. They explicitly describe how the system works, and implicitly hint what the system should have. We develop the main structure from those analysis results, especially the nouns appearing on the descriptions and the diagrams.

The system contains a number of concrete components, including descriptors, algorithms, and viewers, as shown in Fig. 4. To manage those components, the platform has a component management unit that deals with the registration and association of the components. The database support is handled by the persistence management unit. These two units are not directly related to the descriptor design, but are necessary to the application and prototype development. Hence, they are utility units of the testbed. The most important part, though not explicitly mentioned in the requirements, is the framework. The framework is the programming styles and rules when we use the platform to develop new applications. For instance, the common programming interface of using the components and the concurrent execution algorithms are defined in this part. It glues the scattered components and utilities by programming interfaces and class libraries.

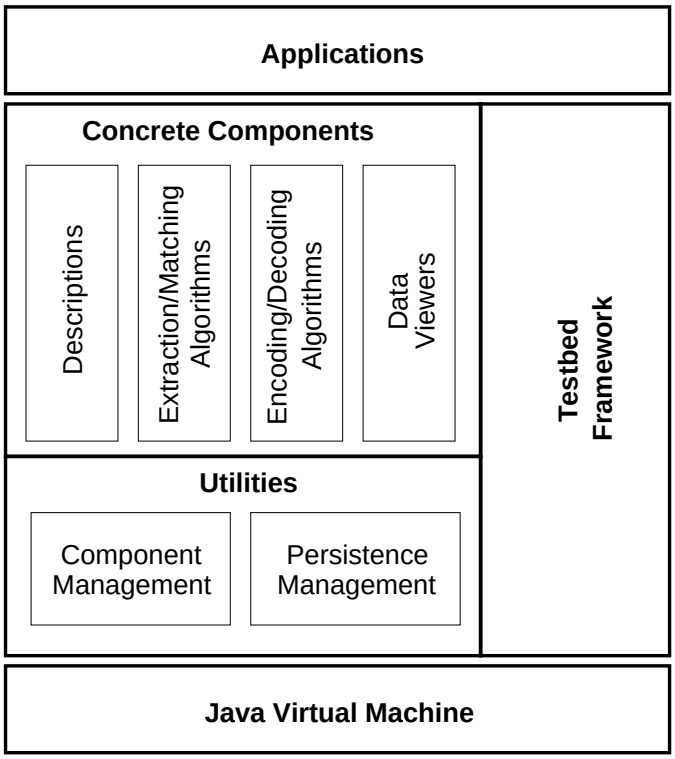


Figure 4. Architecture of the software platform

In general, database access and data input/output are the less portable part of an application, because they differ from system to system. We choose the Java platform as the base of our testbed, because it hides most platform dependent issues under the Java virtual machine. The Java platform tries to unify the database accesses through the JDBC drivers.[12,13] In our experiences, almost all JDBC driver implementations conform to the SQL standard, and handle text-based data very well. As to the binary data access, it is not always well-implemented in the drivers. Since binary object access is inevitable in a normal MPEG-7 application, this issue is handled by the persistence management unit rather than JDBC.

The interfaces exposed to the designer and the developer are the framework and the implemented components. Since these two parts are designed to be system independent, this framework enables users to create

or use the system components without knowing the underlying system configuration. Another feature of this architecture is component re-use. As projects (applications) are gradually developed on this platform, more and more components are registered on the system. It is beneficial to be able to re-use the previously developed modules.

4. IMPLEMENTATION

We will describe the implementation of our proposed testbed in this section. Based on the proposed architecture in Sect. 3, the core of the platform is the framework and the utilities units. The concrete components are the extensions to the core. Our design will proceed from the use-case and the architecture level to the class level.

4.1. Framework Implementation

The framework is the most important part of our platform. For users, it is the interface that connects their applications to the testbed. For the system itself, it is the internal structure that holds the relationships among components. We identify the functionalities of the system by use-case diagrams in the last section, and we now identify the basic classes in our system. These classes are called "domain classes." Fig. 5 shows the classes we identified.

First of all, the Viewer/Data/DataAlg classes are the root of all subsequent components. Data, as the name implies, is a collection of bytes that represent the multimedia content. Specifically, it can be a picture, a video clip, a duration of sound, or any meaningful media data. The DataAlg is the algorithms associated with the data. For example, JPEG encoder and decoder are two DataAlgs bounded to JPEG data. Viewer classes are the renderers for Data classes, such as an image viewer, a sound player, and etc.

There are two special branches of DataAlg: one is for media data and the other is for meta-data (descriptions). The reason we separate these two types of algorithms is that the operations for media data and meta-data are significantly different. The operations applied to media data are encoding and decoding, while the meta-data are usually connected to feature extraction and feature matching. It is often reasonable to group encoder/decoder and extraction/matching algorithms in pairs. Note that we treat an MPEG-7 descriptor as a kind of media data, and hence the MPEG-7 bit-stream is the encoded result of a plain data structure. The associations among viewers and the data are not hard-coded, because a viewer can display more than one kind of data. For example, a vector viewer can display the histogram or the zig-zag scanned DCT coefficients of an image block. The association between data and algorithm is not one-to-one, because several algorithms may be associated with the same data. All the association management tasks are delegated to the ComponentMgr which will be discussed in Sec 4.2.

The domain classes are the conceptual building structures of the platform. In constructing the testbed, we need to define more detailed behaviors of the classes. The Data interface is a wrapper of the real data, and it is the most generic type in the testbed. It acts as a tag interface indicating if the wrapped object is acceptable by this system. Programmers can retrieve MIME-type information through this interface if the system can recognize the object when it is constructed.

The DataAlg is also a tag interface, which represents the algorithms associated with the Data object. It is the root of all algorithm classes. It intends to serve as a common interface of the system to manage algorithms. For the developers (sometimes the designers), it is not directly used as often as its inherited classes. For example, the encoding/decoding algorithms consume MediaData. We use MediaCodec to group them for management, and we use the inherited MediaEncodeAlg class to implement the encoder.

Here we briefly describe how a designer can use our platform to prototype a simple evaluation application. He or she first implements the feature extraction and matching algorithms. Then, he/she specifies a designed ExtractionAlg to consume the MediaData and produce a Description; the MatchingAlg computes the similarity of two Descriptions and gives the distance as the measure of the similarity. To generate a list of matching scores under this framework, the designer may conduct one-by-one comparison. Obviously, this may not be the best solution to a database search, but we view this an application specific issue and we let the developer handle it.

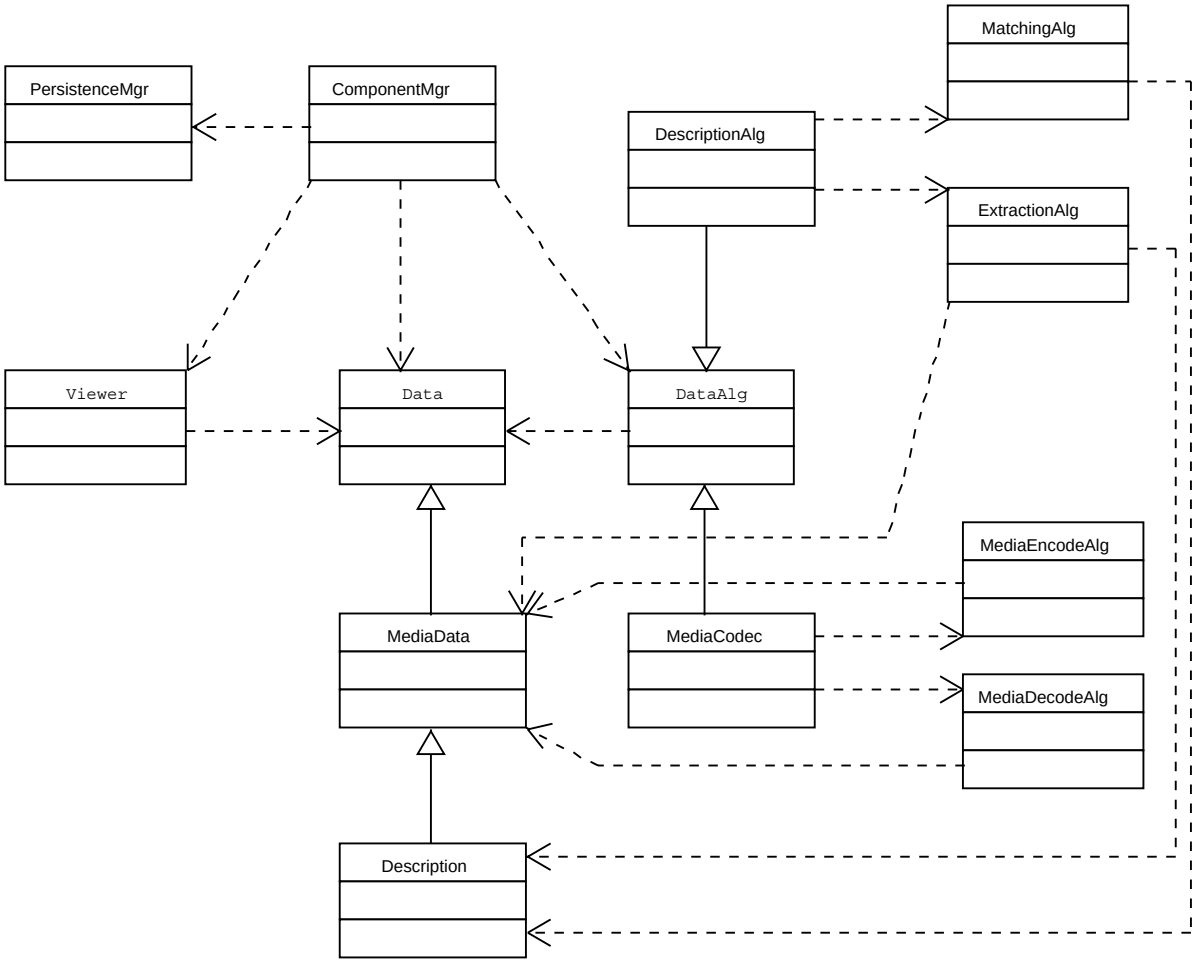


Figure 5. MPEG-7 testbed domain classes

A Viewer is used to view the data object. Following the notations of the model-view-controller (MVC) pattern,[14] the Viewer interface merely indicates that it would co-operate with the model object (the Data) and it provides the general application programming interface (API) for feeding the data into the viewer. Note that we do not define the controller interface in our system, because different applications demand different interactions between a view object and a model object.

The multi-user concurrent execution requirement can be solved by the threading mechanism in Java. Algorithms are a collection of executable codes and related objects. Their properties are similar to those of a thread. Hence, we can implement the algorithms by using the Java Runnable interface, which extends the usability of the class in both single-thread and multi-thread design.

4.2. Utilities

The ComponentMgr manages the associations among viewers, data, and algorithms. One of the requirements of our testbed is to integrate a vast variety of components with their implementations. As more and more applications added into the system, the number of associations grows at a very fast pace. This unit copes with the complexity of the relationships among concrete classes.

The associations of the components are described by the Data-Viewer-DataAlg relationships. They are complicated because they are multi-to-multi associations. To make them simpler, we choose Data as the main

entity and break the triple-relation into two double-relations. As illustrated in Fig. 6, one of the association is the relationship between the Viewer and the Data, and the other is the relationship between the Data and the DataAlg. The ComponentMgr holds all the valid associations. The manager provides the interface that retrieves/adds/removes the corresponding Viewers and DataAlgs for a given Data.

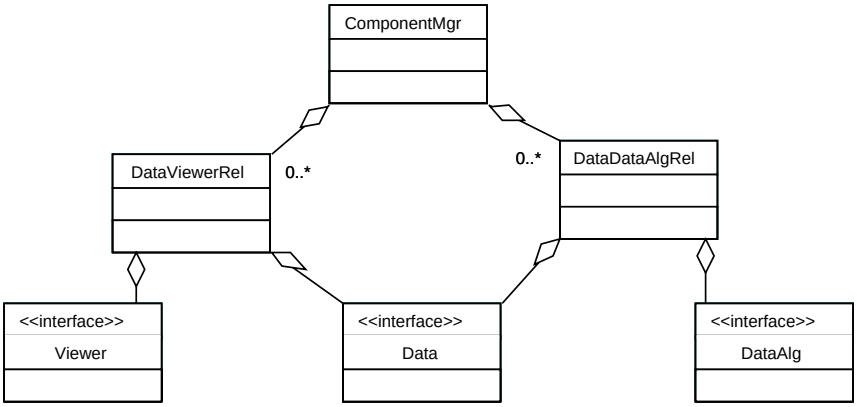


Figure 6. Relationships managed by the component management

The relationships should be stored for future use; however, it is not a good solution to record them as raw objects. The object references should be in an "indirect" form, such as string representation of the class names. We recover a real object from its indirect form. The Java platform provides the class-class, which loads the class by its name and instantiates an object from its class.

The system architecture allows that the users to store objects without knowing detailed database access commands. To satisfy this requirement, we design the persistence management unit, PersistenceMgr, which is a singleton[14] class, the run-time object it points to can be configured in the global configuration.

The PersistenceMgr also configures the JDBC driver automatically, and allows users to obtain Connection objects easily to handle the SQL commands. The binary data persistence is handled by other methods. Currently, our platform supports the binary object persistence of the PostgreSQL database. The implementation uses the LargeObject API provided by PostgreSQL to manipulate bit-stream type data.

5. AN APPLICATION EXAMPLE: A GENERIC IMAGE FEATURE MANIPULATION PROGRAM

In this section, we will demonstrate the flexibility of our testbed by constructing an example program. This application provides a user interface that manipulates image-related descriptors. The testbed framework makes it possible to operate different descriptors through a unified interface. Newly created descriptors can be easily included into the program, because the component manager has the ability to incorporate different kinds of descriptors.

Similar to the testbed design, we start developing this application program from its requirements. We will not describe programming details, such as the descriptor implementations, but we will show the framework and component re-use by an "evolutionary" design. The first requirement is that it shows an image on a window, and it extracts image features. The second requirement is to extract and store image features in a batch. The rest of the functionalities are needed for descriptor matching. We implement three matching schemes. They range from the simple single-descriptor matching to the more sophisticated multi-step matching.

5.1. Extraction-related Operations

Feature extraction is often the first step in a typical MPEG-7 scenario. In addition to the extracting operations, a research friendly program should provide the following extra functionalities:

- A user can select a desired image through a file chooser or an URL input dialog.
- A user can display an image using the specified image viewer.
- A user can extract a specified descriptor from an image.
- A user can display a descriptor by a specified viewer.
- A user can save an image and its associated descriptor for future use.

According to the requirements, we design the classes needed in our program to perform the extraction related operations. Many of these classes can be found in the testbed framework and in the Java class library; therefore, we only need to implement the graphical user interface (GUI) related and the I/O-related classes.

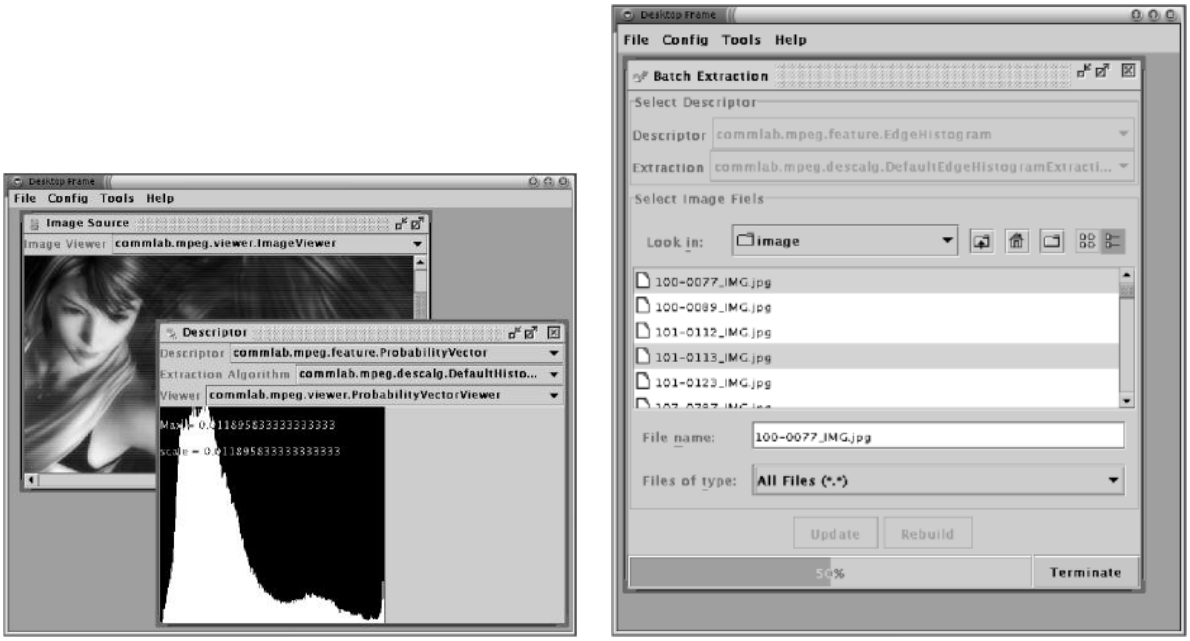
Sometimes we want to process a number of images by a single command, the so-called batch processing. It is convenient to have such a functionality to help users extract features of images. These extracted features can be saved for future processing. In this example program, we assume all images reside in the local file-system. Hence, additional requirements are listed below:

- Allow user to select image files for processing.
- Allow user to select descriptors for extraction.
- Allow user to (re-)build or update (according to the timestamp) the existing descriptors.

To implement the above operations, we need a few directory and file manipulation functions. Fortunately, Java provides them in the java.io.File class. We may create, check existence, and check timestamp by using these functions. Fig. 7 shows the result of our implementation. In Fig. 7(a), an image is loaded and its histogram is extracted. A two-file batch extraction process is shown in Fig. 7(b).

5.2. A Single-descriptor Match

The single-descriptor matching is a very simple matching method. We build a user interface to demonstrate this kind of search. The use of this functionality is similar to that in a typical MPEG-7 scenario. Since search techniques are not closely related to the framework flexibility, we implement a brute-force search algorithm. Reusing the CmdBase and CmdStatusListener introduced in Sect. 5.1, we design and implement the GUI and the command objects. The command extracts the feature from the query image, searches for similar images in the storage, and returns a list of matched results. The image files are displayed as 64×64 icons on the user interface. Fig. 8(a) is the implementation of the single-descriptor match function. The screenshot shows that a user selects the color layout descriptor as the feature, and chooses the default extraction and matching algorithms to perform the operations. The maximum number of returned (matched) images is set to 20. The sample image database contains 797 images. The returned image icons are displayed in the window from left to right and from top to bottom with the most similar one on the top-left corner. When an icon is clicked, its original image is displayed in a temporary image viewer. To speedup icon rendering, we implement a simple icon cache. The icon is generated and stored in the cache when the image icon is first-time listed on the query result. The same icon is loaded directly from the cache, if it is called again.



(a) Loading and extraction

(b) Batch extraction

Figure 7. User interface for extraction-related operations

5.3. Weighted Match

A second match, called weighted matching, is implemented and discussed in this section. The single-descriptor matching often does not produce satisfactory results. The returned images are not subjectively similar to the query item from time to time. A descriptor represents one specific aspect of an image. The single-descriptor matching is not sufficient to imitate the human semantic matching. One improvement is considering a weighted distance sum of several descriptors in matching.

The basic requirements of this functionality are similar to those of the single-descriptor matching. The descriptor selections and matching commands are now extended to multiple-descriptor selections. We also need to include the weighting factors that are associated with the descriptors. We use a table to select descriptors and set their properties, such as algorithms and the weighting factors. Fig. 8(b) shows the user interface of the weighted match parameter input table.

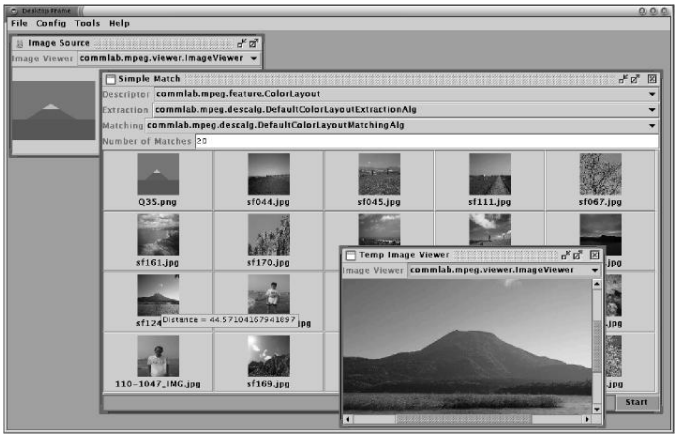
5.4. Multi-step Match

The matching methods described in Sect. 5.2 and 5.3 are one-step matching; that is, the similarity metrics are calculated in one-shot. We obtain the searching results right after we scan through the entire image database. Although the weighted matching method often performs better than the single-descriptor matching method, one-shot scheme has a rather limited performance. This can be further improved by the multi-step schemes.

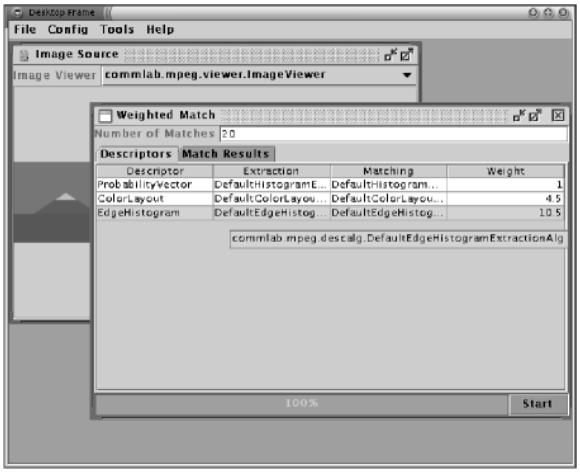
We can set the priorities of multiple descriptors using sequential steps. Each step produces a subset from the previous searching results. For example, we may first use the edge histogram to find, say, 100 similar images from the database. Then, we search by color layout in this list to produce the final 20 matches. This method can be considered as a chain of weighted matching processes.

One of the implementation issue is that we do not restrict the re-use of the same query descriptor in different steps. We have to modify the WeightedMatchCmd to prevent re-extract an extracted descriptor. This can be

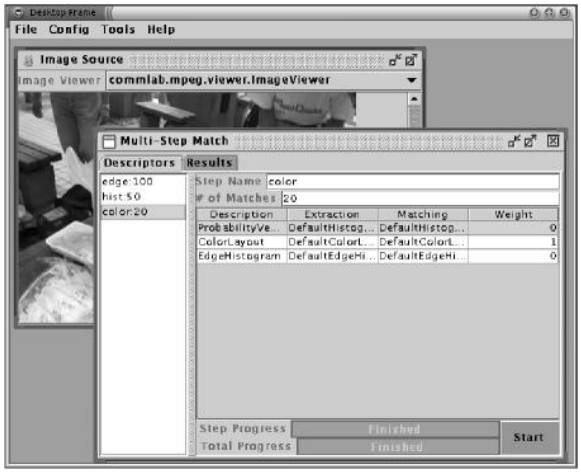
easily solved by adding a collection to hold all extracted descriptors. Before we extract the query descriptor, we fetch it in the collection. If there is an extracted version in the cache, we skip the extraction and continue the remaining process. If the descriptor has not been extracted, we perform the extraction and put the descriptor into the cache. Fig. 8(c) is a screenshot of the multi-step search. A user can input the step parameters in the left and the upper-right panel. For each step, the central part of the right panel is the same as the weighted matching parameter panel. The bottom part of the right panel displays the progress of both the global and the current processing.



(a) Single-descriptor matching



(b) Weighted matching



(c) Multi-step matching

Figure 8. Screenshots of matching related operations

6. CONCLUSIONS

In this paper, we designed a research friendly software testbed based on the concepts of MPEG-7. The system requirements come from an MPEG-7 typical scenario, plus the functionalities a researcher may need. The testbed has a framework that provides user programming interfaces and an abstraction of all MPEG-7 defined

operations. We believe the viewer-data-algorithm relationship covers most of the research requirements. Though the testbed is designed for researchers, it can also be used as the base for developing a real MPEG-7 application. The application designer may benefit from the additional functionalities offered by the framework in this testbed. For example, an image search engine can use the viewers as the renderers for browsing, and a distributed search engine may take the advantage of the Java platform to eliminate the cross-platform issue.

Treating the descriptors as a type of media data makes it possible to re-use the codec interfaces without messing up the architecture with redundant classes. The persistence and component management units hide the tedious database and configuration jobs, thus the users can focus on the descriptor design and application prototyping. With a few implemented concrete components, we show the flexibility of this testbed by an example, which contains three types of matching schemes. This example not only demonstrates the feasibility of developing applications on our platform, but also shows the extensibility and re-usability when the functionalities are incrementally added.

There are a few issues for further improving this testbed. Our system is built on the Java platform and it uses JDBC as the database connection method. Our experiences indicate that the binary object accesses are relatively slow comparing to the text-based accesses. The performance may be bounded by the JDBC driver. If we want to speed up the search, a search algorithm targeting at a particular application should be carefully designed. Java provides many network related functionalities such as socket, URL, and RMI.[15] It is an interesting topic to extend our testbed across the network. Java has several media extensions, such as JMF[16] and JAI.[17] They are useful extensions for handling media streams and images. We may enrich our testbed implementation by incorporating these extensions. Solutions to these issues are expected to be added into the next version.

REFERENCES

1. J. D. Gibson, *Multimedia Communications — Directions and Innovations*, Academic Press, San Diego, CA, 2001.
2. A. Puri and T. Chen, *Multimedia Systems, Standards, and Networks*, Marcel Dekker, New York, NY, 2000.
3. *Overview of the MPEG-7 Standard (version 5.0)*, *ISO/IEC JTC1/SC29/WG11 N4031*, MPEG7 Committee, Mar. 2001.
4. F.-C. Chang and H.-M. Hang, "An introduction to mpeg-7," in *Computer Vision, Graphics, and Image Processing Conference*, pp. 50–57, (Taipei), Aug. 2000.
5. http://www.cselt.it/mpeg/, "Mpeg homepage."
6. *Multimedia Content Description Interface - Part 2: Description Definition Language*, *ISO/IEC JTC1/SC29/WG11, FDIS N4288*, MPEG Committee, Jul. 2001.
7. *Multimedia Content Description Interface - Part 6: Reference Software*, *ISO/IEC JTC1/SC29/WG11, FCD N4206*, MPEG Committee, Jul. 2001.
8. Q. Huang, A. Puri, and Z. Liu, "Multimedia search and retrieval: New concepts, system implementation, and application," *IEEE Trans. Circuits Syst. Video Technol.* **10**, pp. 679–692, Aug. 2000.
9. G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, Addison Wesley Longman, Inc., 1999.
10. http://www.uml.org/, "Uml home page."
11. http://www.omg.org/, "Omg home."
12. http://java.sun.com/products/jdbc/, "Jdbc."
13. G. O'Sullivan, *Java 2 Complete*, SYBEX Inc., 1999.
14. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns*, Addison Wesley Longman, Inc., 1995.
15. http://java.sun.com/products/jdk/rmi/, "Remote method invocation."
16. http://java.sun.com/products/java media/jmf/, "Java media framework."
17. http://java.sun.com/products/java media/jai/, "Java advanced imaging."