A Framework for User Interface Design in Visual Information Retrieval

Horst Eidenberger and Christian Breiteneder
Vienna University of Technology, Institute of Software Technology and Interactive Systems,
Favoritenstrasse 9-11 – 188/2, A-1040 Vienna, Austria

Abstract

This paper describes the user interface framework of the VizIR project ([4]). VizIR is an open project to develop a Java-based, extendible and well-documented asset framework for visual information retrieval. The paper includes a description of the visual components and their class structure, the communication between panels and the communication between visual components and query engines. Visual components include media panels, controls and renderer classes. Panels communicate through media events. Communication of user interfaces and query engines is based on the Multimedia Retrieval Markup Language (MRML, [10]). MRML is an XML-based language that was developed by the University of Geneva. To be usable with our querying paradigm, we extend MRML with additional elements. The paper contains a short implementation section with details on the Java components used, Java 3D graphics libraries (GL4Java and Java3D) and Java XML parsing. Finally, an appendix contains the MRML extension.

1. Introduction

In the past, the design of user interfaces of Visual Information Retrieval systems (VIR) was – in comparison to most other visual systems – quite simple. For Content-based Image Retrieval (CBIR) the usual approach offered a two-dimensional panel with a matrix of images (that were shown as thumbnails) and a limited possibility to select features and to add metadata (like weights, relevance feedback, etc.). Content-based Video Retrieval Systems (CBVR) focused on the problem of representing the temporal dimension in the static context of a user-interface. State-of-the-art solutions are Micons and Hierarchical Video Browsers (see [6]).

In this paper we present the user interface part of the VIR project VizIR. Goal of the project is the development of an open and extendible framework for VIR research. The basic structure of VizIR is laid down in [4]. See Section 3 for a short project overview. VizIR is open source and project participants of any kind are welcome. Especially, we would like to invite interested researchers to take part in the design and implementation process of this truly open project.

Goal of VizIR is not the development of a monolithic user interface but of a system-independent class framework of user interface components (interaction panels, event model, etc.). An important issue of VIR user interfaces is the communication with query engines. This communication should be standardized (in order to combine arbitrary user interfaces and querying systems) and be based on modern communication paradigms (XML, etc.).

The rest of this paper is organized as follows. Section 2 gives an overview on relevant related work, Section 3 points out the major goals of the VizIR project, Section 4 is a description of the user interface framework in VizIR, Section 5 discusses implementation issues and finally, in Section 6 current and next issues in the context of the paper are listed.

2. Related work

Subsequently, we will have a look at the user interfaces of well-known VIR systems: first CBIR systems and then CBVR systems. The focus in CBIR will be on classic systems (including QBIC and Virage) and two promising more recent approaches (El Niño and ImageGrouper). The Section ends with a short description of an approach to standardize the communication of VIR user interfaces and query engines.

The user interfaces of classic CBIR systems are quite simple. Most systems(QBIC [5], Virage [1], Photobook [14], VisualSEEk [18]) use a single 2D panel of images for query definition and result set display. Querying is done by selecting one or more query examples, one (f. e. QBIC), a few (f. e. MARS [13]) or all features (Virage) and – in the latter two cases – weights for the importance of these features. Iterative Refinement by Relevance Feedback can usually be done by defining the importance of result set elements textually and re-running the query. This paradigm has several drawbacks: earlier result sets are thrown away, selecting features and weights overtaxes the casual user

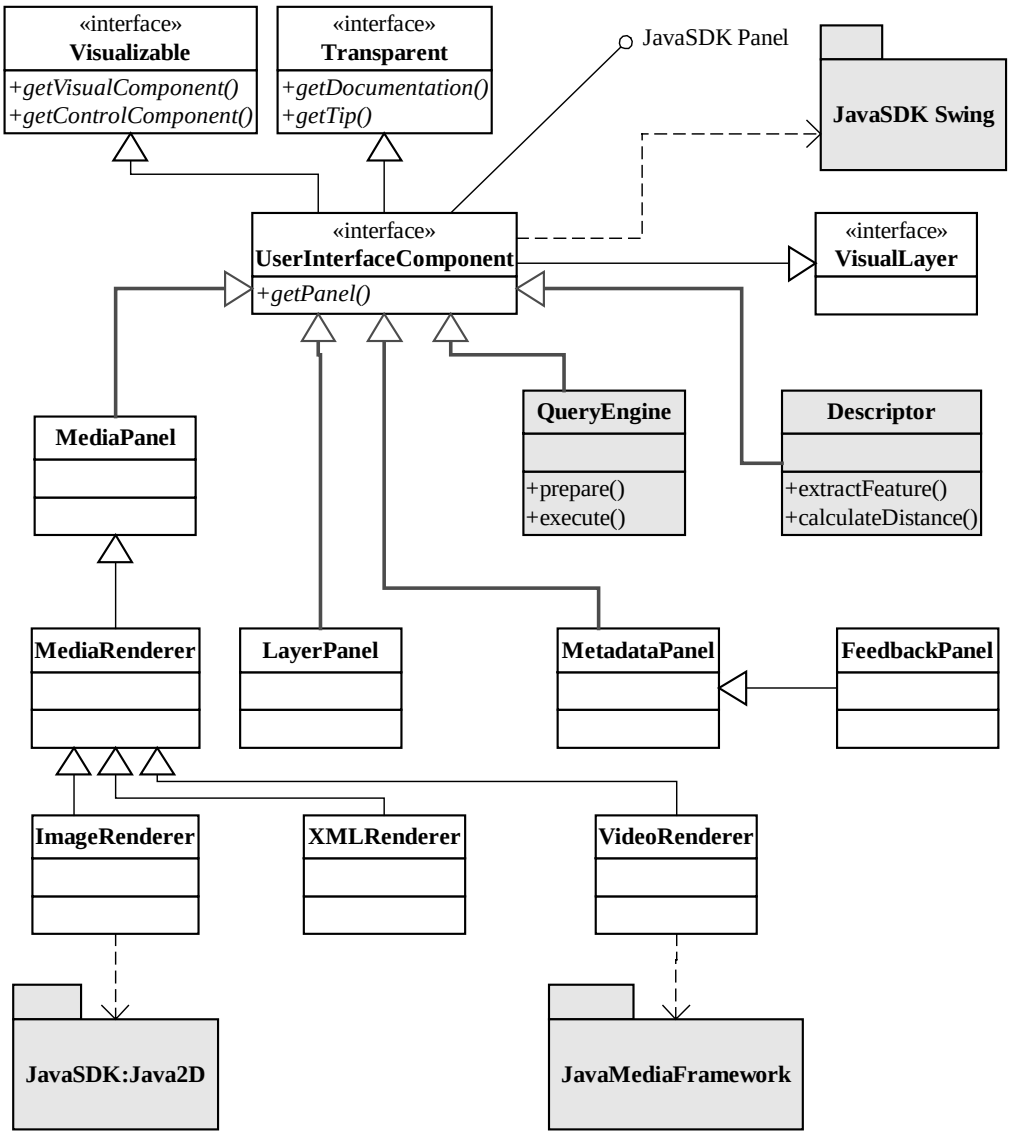

Figure 1. Class diagram for the user-interface framework in VizIR.

and after all, the static structure of this interface type is not user-friendly and old-fashioned.

That's why in the last years several groups have been working on new user-centric interface approaches. Two of the most interesting are El Niño ([16], [17]) and ImageGrouper ([11]). To the authors knowledge, El Niño is the first approach to define a query implicitly by the distance relations of objects in a 3D panel. This query definition process can be done intuitively and easily by drag-and-drop. The most interesting innovations in ImageGrouper are the usage of two panels for the active and the last query and a history over all refinement steps in a querying session. The central idea of ImageGrouper is the definition of queries through three groups: positive examples, negative examples and neutral examples. ImageGrouper's major drawback is that it has no standard interface to query engines and is bound to an engine with classic distance measurement and linear weighted merging.

Like El Niño, VizIR will contain 3D user interfaces for query formulation. Using 3D information visualization techniques instead of 2D methods has several advantages. Generally, each 3D view is just a 2D projection ([19]). 3D views take advantage of human spatial memory and allow displaying more information without incurring additional cognitive load because of pre-attentive processing of perspective views. In general, they lead to better retrieval results in user studies in sense of reaction time, number of incorrect retrievals and failed trials ([15]). Additionally,

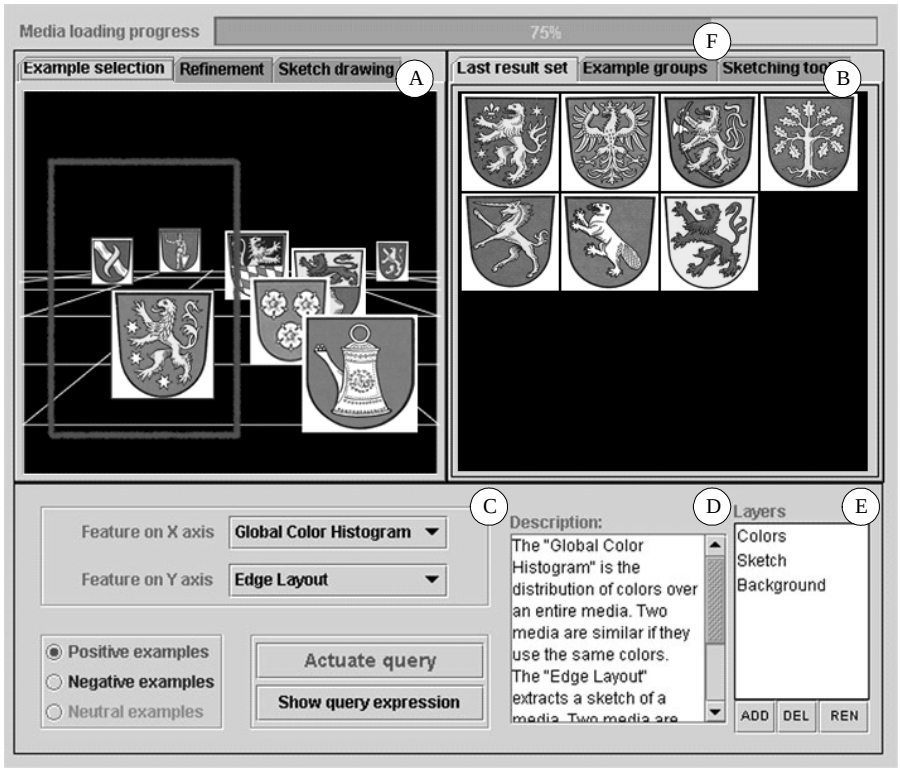

Figure 2. Screenshot of user-interface prototype.

they allow the rendering of more information items because of scaling possibilities and a better global view. Finally, there is experimental evidence that 3D displays enhance subjects' spatial performances ([19]). The major open problem of 3D systems in this context is the development of 3D user interaction techniques ([15], [9]).

Classic CBVR systems are OVID ([12]) and VQIS. The most interesting point concerning the user interfaces of CBVR systems is the handling of temporal media (video and animations) in a static user interface. Generally, there are three possible solutions: (1) integration of the full video with player controls into the environment (CPU power and network bandwidth consuming), (2) production and usage of animated icons (CPU power consuming, difficult) and (3) production of still images that represent the video content. The third solution is the most widely applied (in VIR). The simplest form of representation is an image matrix of all keyframes in a video clip. Another approach is the Micon, a pseudo-three-dimensional cube that shows the first frame of a video clip as well as the first line and the last column of all consecutive frames. Another type is the Hierarchical Video Browser, a tree-structured view of a video clip.

The interoperability of VIR user interfaces and querying systems is a matter that is gaining more and more attention. Interoperability should be achieved by standardized interfaces. The most promising effort in this direction is the Multimedia Retrieval Markup Language (MRML, developed at the University of Geneva by the group of Prof. Pun, see [10]). MRML is an XML-based standard. It is implemented in the user interface Charmer and the basis of the Benchathlon project (see [10] for details)

We tried to incorporate the best ideas of the systems above and MRML into our interface components. Before we continue with the user interface framework, the next section outlines the major goals of the VizIR project.

3. VizIR Project overview

The VizIR project aims at the following major goals: First, implementation of a modern, open *class framework* for content-based retrieval of visual information as basis for further research on successful methods for automated information extraction from images and video streams, definition of similarity measures that can be applied to approximate human similarity judgment and new, better concepts for the user interface aspect of visual information retrieval, particularly for human-machine-interaction for query definition and refinement and video handling.

Second, implementation of a working *prototype* system that is fully based on the visual part of the *MPEG-7*

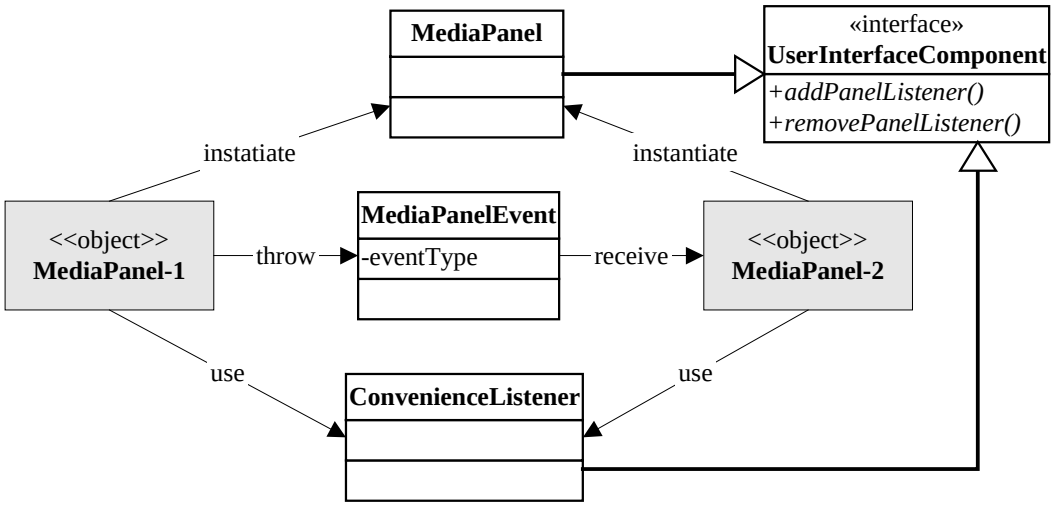

Figure 3. Event model for panel communication.

standard for multimedia content description. Obtaining this goal requires the careful design of the database structure and an extendible class framework as well as seeking for suitable extensions and supplementations of the MPEG-7 standard by additional descriptors and descriptor schemes, mathematical and logical fitting distance measures for all descriptors (distance measures are not defined in the standard) and defining an appropriate and flexible model for similarity definition. MPEG-7 is not information retrieval specific. One goal of this project is to apply the definitions of the standard to visual information retrieval problems.

Third, development of integrated, *general-purpose user interface components* for visual information retrieval. Such user interfaces have to include a great variety of different properties: methods for query definition from examples or sketches, similarity definition by positioning of visual examples in 3D space, appropriate result display and refinement techniques and cognitively easy handling of visual content, especially video. The user interface part of VizIR is central topic of this paper.

Finally, support of methods for *distributed querying, storage and replication* of visual information and features and methods for query acceleration. The importance of these issues becomes apparent from the large amount of data that has to be handled in such a system and the computational power that is necessary for querying by – often quite complex – distance functions. Methods for distributed querying, storage and replication include the replication of feature information, client-server architectures and remote method invocation in the querying and indexing modules as well as compression of video representations for the transport over low bandwidth networks. Methods for query acceleration include indexing schemes, mathematical methods for complexity reduction of distance functions and generation of querying heuristics ([3]).

To achieve these goals state-of-the-art software development is necessary. VizIR software development is based on reverse engineering and the Rational Unified Process. The output of VizIR will be available to the public. The overall goal of VizIR is providing the research community with a flexible tool for experiments. In the next section the user interface concept of VizIR will be explained in detail.

4. User interface design

In the first part of this section the class framework and the event model of the user interface components are explained (including a screenshot) and in the second part communication and querying issues are tackled.

4.1. User interface framework

Figure 1 shows the static structure of the VizIR user interface framework. The central element is the interface *UserInterfaceComponent* that is inherited by all classes that have a visual panel. These are *MediaPanel* (the mother class of all panels that deal with media objects), *QueryEngine* (the mother class of all querying engines, the panel contains all elements that are necessary for query formulation), *Descriptor* (mother class of all implemented features, the panel contains a toolbox for sketch drawing for this feature), *MetadataPanel* and *LayerPanel* (a layer manager for multi-layer image sketching like in Photoshop). VizIR is based on Java and the VizIR user interface components are based on Swing. *UserInterfaceComponent* inherits methods from the interfaces *Visualizable* (methods for receiving a visual

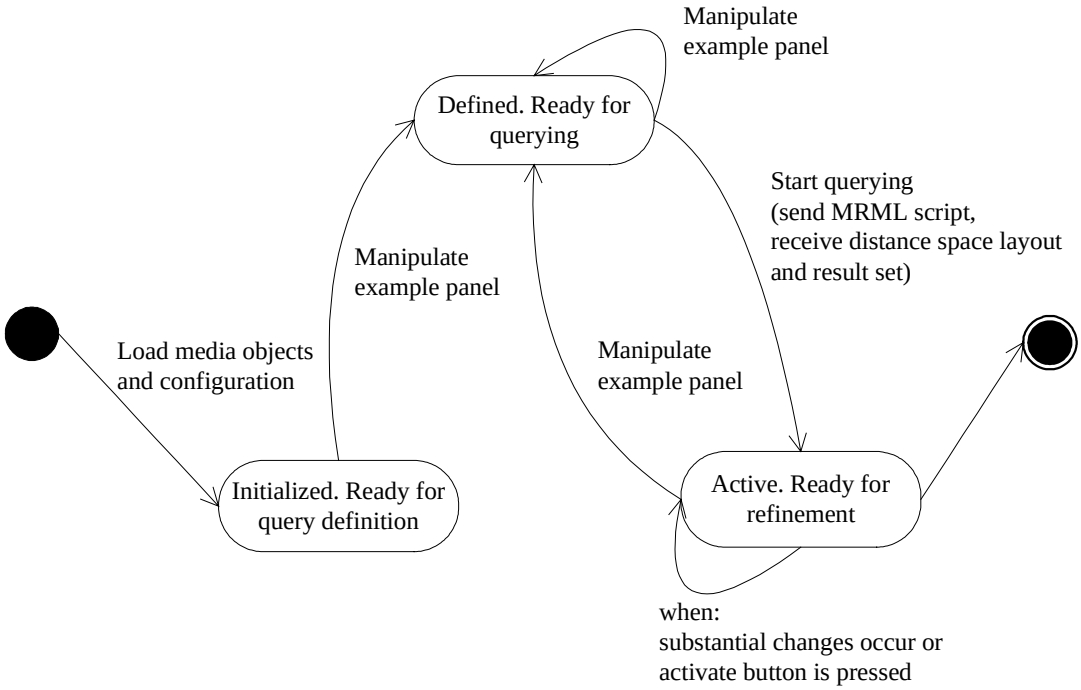

Figure 4. State-transition-diagram of querying process.

panel and a visual control component like in the Java Media Framework), *Transparent* (methods for receiving visual documentation and help in the user interface) and *VisualLayer* (defines the structure of a layer of the sketching panel, basically a Java Image type).

MediaRenderer is a special type of *MediaPanel* for visual rendering of media objects. So far, we have implemented three renderer for images (JPG, PNG, GIF, etc., based on Java2D), videos (generates Micons – see Section 2 – for arbitrary video formats: MPG, AVI, MOV, etc., based on the Java Media Framework [8]) and XML. *XMLRenderer* can render any XML-file that can be displayed in a web browser (see [7] for technical details). The most important media panel is our 2.5D media panel. For an example see element A of the screenshot in Figure 2. This panel can be used for example selection, browsing, query formulation and result display. The rendered equivalents of media objects are displayed as images, that are parallel to the image plane. It is possible to navigate in two dimensions (left-right, forward-back) and to zoom. Groups of objects can be selected, moved and associated with metadata (through communication with a *MetadataPanel*). The angle of the image plane and the X-Y-plane can be varied between 0° and 90°. The panel has a visual control component (element C of the screenshot). The upper part of this panel is initialized with all dimensions of the media space that should be displayed (in the VIR context: all implemented features). The view changes whenever new dimensions are chosen for the X- or Y-axis.

We are implementing two querying paradigms: query by example (QBE) and query by sketch (QBS) because these are the most intuitive ones. QBE follows our Click-and-Refine approach (C&R, see [2]): in the first step the user selects groups of media objects from the ‘example selection’ panel, initiates a query and refines the result in the ‘refinement’ panel (another 2.5D media panel) by the adoption of cluster borders (see [2] and Subsection 4.2 for details on the querying concept). Sketches for QBS can be drawn in the ‘sketch drawing’ panel. This panel contains layers of type *VisualLayer* that are managed by the *LayerManager* (element E of the screenshot) and allow drawing with the tools provided by the descriptor objects. These tools are collected in the ‘sketching tools’ panel (element B). The ‘last result set’ panel contains the media objects of the last result set (similarity values are associated as metadata). It is just a special 2.5D example panel with a image-plane X-Y-plane angle of 0°. The same is true for the ‘example groups’ panel that lists all query examples partitioned in three groups: positive, negative and neutral examples. The lower left part of the query engine control panel (element C) is a selection list for the current active grouping tool (relevant for the ‘example selection panel’) and the lower right part contains the querying button. The ‘description’ panel (element D) contains the information of the methods from the

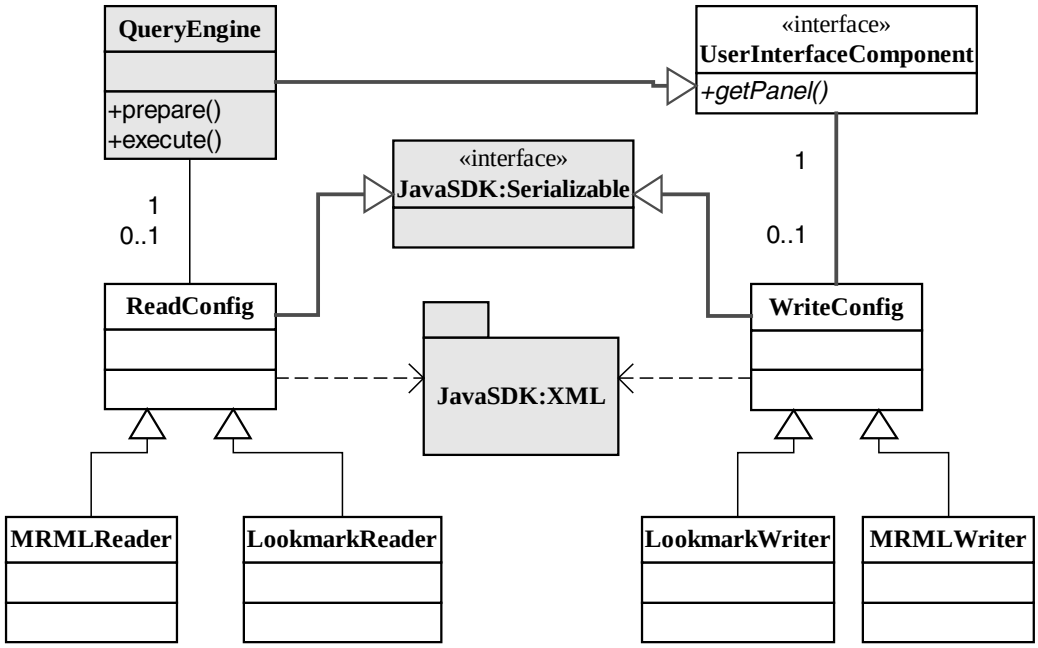

Figure 5. Class diagram for MRML communication in VizIR.

Transparent interface for the active user interface element.

The VizIR user interface class structure follows the paradigm that all components (methods, panels, etc.) are defined, where they are used. Thus, each query engine has a visual panel for query formulation and each descriptor has a panel with tools for sketching (f. e. line drawing tools for a edge layout descriptor). To guarantee the transparency of VizIR we defined in [4], each visual component has to implement the *Transparent* interface with documentation and tips. The panels of the framework can be integrated into any visual Java container and can be organized arbitrarily. The layout in the screenshot in Figure 2 is just an example.

Arbitrary combination is guaranteed by the communication mechanism of the framework. It follows the Delegation-Event-Model and is conceptually shown in Figure 3. Each object of class *MediaPanel* (MediaPanel-1 and MediaPanel-2) can communicate with any other *MediaPanel* through *MediaPanelEvent* objects. Thus, all media panels have to implement listener classes that are defined in *UserInterfaceComponent* and flag the media panel events they throw. For easier user interface building the framework contains convenience classes with listener functions for standard communication operations (f. e. communication of query control panel and 2.5D example panel when the example group selection is changed, etc.).

To conclude this sketch of the user interface classes, Figure 4 shows a State-Transition-Diagram of the underlying querying process. First the user interface components are initialized with media objects and query parameters (element F of Figure 2 shows a progress bar panel for media loading). Then the user can define a first query by selecting example media objects. This brings the user interface in the defined state. Executing the query brings the user interface in the active state where refinement can be started or a new query can be defined. In active state the query is re-executed whenever the user presses the 'activate' button or the query engine control component detects substantial changes in the query definition. The next Subsection is dedicated to the querying and communication process.

4.2. Querying & communication interface

Query engines in VizIR can be arbitrary. We are implementing a query engine that is based on our Query Model concept (QM, see [2], [3]). The communication of user interfaces and query engines is standardized and based on MRML (see Section 2).

Each framework component that uses MRML for communication, uses instances of the classes *MRMLReader* and *MRMLWriter* (see Figure 5). These classes are derived from *ReadConfig* (XML parser class) and *WriteConfig* (XML writer class). Communication classes for new XML languages can be implemented in the same way. To be able to perform QM queries with MRML we had to extend its Document-Type-Definition (see Appendix for DTD code). We have defined elements for context-free media and media group definition, descriptor

definition and query definition. The following example illustrates how these extensions can be used:

```
<logicalQuery>
  <clusterDefinition>
    <clusterRestriction>
      <clusterDimension
        lowerBound="0.0"
        upperBound="0.5">
        <mediaGroup id="qe1"
          type="positive">
          <mediaObject
            dataLocation="img1.gif"
            iconLocation="thumb1.gif"/>
        </mediaGroup>
        <descriptor name="ColorHist"/>
      </clusterDimension>
    </clusterRestriction>
  </clusterDefinition>
</logicalQuery>
```

This construct defines a query (on the *collection* defined elsewhere in the MRML script) with a single feature. A color histogram is used to find all media objects that have a distance to the query example *'img1.gif'* (represented by the icon *'thumb1.gif'*) that is smaller than *'0.5'*. If we would like to retrieve all objects that fulfil this condition *and* a second one, we would put the second *clusterRestriction* in the same *clusterDefinition*. If we would like to retrieve all that fulfil the first *or* the second one, we would put the second one in a new *clusterDefinition*. These constructs are very flexible and can be used in various ways. They should not only support our QM concept but – according to its published querying paradigm – the one used in MARS as well ([13]).

Because the VizIR framework is based on Java and the JavaSDK it is possible to integrate the user interface components into any container (frame, applet, etc.), to do distributed querying (with CORBA, RMI, etc.) and querying in the background (in a separate thread). The next Section points out relevant implementation issues.

5. Implementation

VizIR is based on Java and media handling is based on the Java Media Framework ([8]). The 2.5D panel is based on Gl4Java ([7]) instead of Java3D for the following reasons: (1) Gl4Java is based on OpenGL and much faster than Java3D, (2) event handling is easier and bug-free, (3) it is easier to install (e.g. less dependent on the graphics hardware than Java3D) and (4) has less bugs than Java3D.

XML reader and writer classes are based on the Java XML classes. We use the JDOM parser for XML writing (because it allows the construction of an object tree in memory and does the serialization automatically) and SAX for XML parsing (because it is more flexible and faster than JDOM).

A special problem of VIR user interfaces is the transportation of media object to the client computer. We do media loading in the background through an RTP stream. The Java Media Framework contains a convenient RTP-based streaming component. The user interface is usable as soon as at least a certain quantity of the media objects has arrived at the client side. This is optimized by sending a subset of representative media objects through the stream first.

6. Current and future work

Currently we are working on the first release of the user interface framework. This first release will also include a video renderer and a webpage renderer for thumbnail creation. Next we will work on other methods for video representation. We will follow two approaches. First, we will implement a renderer that produces animated icons of selected keyframes of a video. The keyframes will indicate scene changes. The second approach originates in 2D animation. Short sequences of keyframes will be overlaid with an alpha-channel and thus integrated into a video thumbnail. Another idea we will follow in the future is the implicit definition of features from the selection of media elements or media element regions and expert knowledge. In the past we have been working on a similar idea that resulted in the system presented in [2].

7. Conclusion

This paper describes the user interface framework of the Visual Information Retrieval project VizIR. The framework consists of a class hierarchy of user interface panels with event communication, communication and configuration methods based on XML and an extension of the Multimedia Retrieval Markup Language (MRML). VizIR and the VizIR user interface framework are open, extendible and free. A first release of the user interface part should be available in autumn 2002. Interested researchers and software developers are invited to join the VizIR project. Finally, we would like to thank Geert Fiedler, Markus Raab and Herwig Steininger for their contribution to the user interface framework.

8. References

[1] Bach, J., Fuller, C., Gupta, A., Hampapur, A., Horowitz, B., Humphrey, R., Jain, R., Shu, C.: The Virage image search engine: An open framework for image management. Proc. of SPIE Storage and Retrieval for Image and Video Databases, San Jose (1996) 76-87

[2] Breiteneder, C., Eidenberger, H.: Automatic Query Generation for Content-based Image Retrieval. Proc. of IEEE Multimedia Conference, New York (2000) 705-708

[3] Breiteneder, C., Eidenberger, H.: Performance-optimized

feature ordering for Content-based Image Retrieval. e-Proc. European Signal Processing Conference, Tampere (2000)
[4] Eidenberger, H., Breiteneder, C.: A Framework for Visual Information Retrieval. Proc. Visual Information Systems Conf., Springer LNCS 2314, HSinChu (2002) 105-116
[5] Flickner, M., Sawhney, H., Niblack, W., Ashley, J., Huang, Q., Dom, B., Gorkani, M., Hafner, J., Lee, D., Petkovic, D., Steele, D., Yanker, P.: Query by Image and Video Content: The QBIC System. IEEE Computer 28/9 (1995) 23-31
[6] Furht, B., Smoliar, S.W., Zhang, H.: Video and Image Processing in Multimedia Systems. 2nd edn. Kluwer, Boston (1996)
[7] GL4Java (Jausoft) Website. http://www.jausoft.com/products/gl4java/
[8] Java Media API Website. http://java.sun.com/products/java-media/
[9] Keim, D.A., Visual exploration of large data sets. Communications of the ACM 44/8 (2001) 38-44
[10] Multimedia Retrieval Markup Language Website. http://www.mrml.net
[11] Nakazato, M., Manola, L., Huang, T.S.: ImageGrouper: Search, Annotate and Organize Images by Groups. Proc. of Visual Information Systems Conference, Springer LNCS 2314, HSinChu (2002) 129-142
[12] Oomoto, E., Tanaka, K.: OVID: design and implementation of a video-object database system. IEEE Transactions on Knowledge and Data Engineering 5/4 (1993) 629-643
[13] Ortega, M., Yong, R., Chakrabarti, K., Porkaew, K., Mehrotra, S., Huang, T.S.: Supporting Ranked Boolean Similarity Queries in MARS, IEEE Transactions on Knowledge and Data Engineering, 10/6 (1998) 905-925
[14] Pentland, A., Picard, R.W., Sclaroff, S.: Photobook: Content-Based Manipulation of Image Databases. SPIE Storage and Retrieval Image and Video Databases II, San Jose (1994) 34-47
[15] Robertson, G., Czerwinski, M., Larson, K., Data Mountain: Using Spatial Memory for Document Management. Proc. ACM Symposium on User Interface Software and Technology, San Francisco (1997) 153-162
[16] Santini, S., Jain, R.: Beyond Query By Example. Proc. ACM Multimedia Conf., Bristol (1998) 345-350
[17] Santini, S., Jain, R.: Integrated browsing and querying for image databases. IEEE Multimedia 7/3 (2000) 26-39
[18] Smith, J. R., Chang, S.: VisualSEEk: a fully automated content-based image query system. Proc. ACM Multimedia Conf., Boston (1996) 87-98
[19] Tavanti M., Lind M., 2D vs. 3D, implications on spatial memory. Proc. IEEE Symposium on Information Visualization, San Diego (2001) 139-145

9. Appendix

This appendix contains the Document Type Definition (DTD) for the essential part of our MRML extension. The extension includes elements for context-free media and media group definition, descriptor definition and query definition according to our querying paradigm. It is based on the MRML definition presented in [10]. See Subsection 4.2 for details. The tags below can be easily integrated into MRML by adding *logicalQuery* and *mediaGroup* as sub-tags of the *mrml* tag.

9.1. Media and media group definiton

In MRML media objects can be context-sensitively defined as *user-relevance-elements* (for querying) or as *query-result-elements*. For initialization we add a tag for general media definition:

```
<!ELEMENT mediaObject (descriptor*)>
<!ATTLIST mediaObject
  dataLocation CDATA #REQUIRED
  iconLocation CDATA #REQUIRED>
```

As far as we understand, the *collection*-construct of MRML can not be used for the definition of media groups (for querying, etc.). We define the following element for this purpose:

```
<!ELEMENT mediaGroup (mediaObject+)>
<!ATTLIST mediaGroup
  id CDATA #REQUIRED
  type (positive|negative|neutral|
        init|other) 'positive'>
```

The first three types define querying groups and the fourth is for initialization.

9.2. Descriptor definition

MRML uses the *algorithm*-construct for the definition of features. For extended use we define arbitrary descriptors as follows:

```
<!ELEMENT descriptor EMPTY>
<!ATTLIST descriptor
  name CDATA #REQUIRED
  value CDATA
  distanceValue CDATA>
```

distanceValue is a special field that is only used when media objects are grouped to describe the layout in distance space (related to the query examples) instead of feature space.

9.3. Logical Retrieval query definition

According to our Query Model approach, a query can be defined with these elements (see 4.2 for an example):

```
<!ELEMENT logicalQuery
  (clusterDefinition+)>
<!ELEMENT clusterDefinition
  (clusterRestriction+)>
<!ELEMENT clusterRestriction
  (clusterDimension+)>
<!ELEMENT clusterDimension
  (mediaGroup,descriptor>
<!ATTLIST clusterDimension
  lowerBound CDATA #REQUIRED
  upperBound CDATA #REQUIRED>
```