UNIVERSITE DE GENEVE

CENTRE UNIVERSITAIRE
D'INFORMATIQUE
GROUPE VISION

Date: First version: October 6, 1999
This version: October 28, 1999
N° 99.04

TECHNICAL REPORT

VISION

# MRML: Towards an extensible standard for multimedia querying and benchmarking

## Draft Proposal

Wolfgang Müller Zoran Pečenović[1] Arjen P. de Vries[2*]
David McG. Squire Henning Müller Thierry Pun

Computer Vision Group
Computing Science Center, University of Geneva
24 rue du Général Dufour, CH - 1211 Geneva 4, Switzerland

[1]Laboratoire de Communications Audio-Visuelles and Groupe de l'Ergonomie des Systèmes Intelligents
Ecole Polytechnique Fédérale de Lausanne, Switzerland

[2]Department of Computer Science
University of Twente, The Netherlands

e-mail: Wolfgang.Mueller@cui.unige.ch,
Zoran.Pecenovic@lcav.de.epfl.ch

*Arjen was happy to test that thanks still worked.

**Abstract**

In recent years, the need for databases which query multimedia data by content has become apparent. Many commercial and non–commercial research groups are trying to fulfill these needs.

The development of research can be described as moving in two directions

- search for new, useful query and interaction paradigms
- deeper research to improve the performance of systems that have adopted a given query paradigm.

The search for new better performance given a query paradigm has led to "clusters" of systems which are similar in their interaction with the user, and which give a certain set of interaction capabilities to the user.

It is already visible, that research will move towards systems which enable the user to formulate multi–paradigm queries in order to further improve results.

As a consequence of the above, there is the need for

- A common mechanism for shipping multi–paradigm queries and their results , which assures that the right query processor processes the right query.
- For each paradigm a common language, which enables to formulate queries in this paradigm.

This mechanism, of course, has to be extensible in order not to constrain ongoing research.

We propose MRML (Multimedia Retrieval Markup Language), a new XML-based language that provides an extensible mechanism for shipping multi–paradigm queries. As a demonstration of its use, the current version of MRML already provides a detailed extensible language for shipping Query–By–Example (QBE) queries in a CBIRS (Content Based Image Retrieval) context.

In this article we highlight

- which advantages the use of MRML can have for use in the CBIRS and other communities. Especially we show how our groups (Lausanne, Geneva) use MRML to minimize programming work and to share user data, and how *MRML can be used for a common CBIRS benchmark.*
- that MRML is explicitly designed to minimize the programming work imposed by its use,
- how MRML can be extended towards other query paradigms,

In this paper, we also propose a development model, that will keep MRML an open standard that is usable while it grows.

**Keywords:** multimedia database,interoperability,query language

# 1 Introduction

The development of research can be described as moving in two directions

- search for new, useful query and interaction paradigms
- deeper research to improve the performance of systems that have adopted a given query paradigm.

The search for new better performance given a query paradigm has led to "clusters" of systems which are similar in their interaction with the user, and which give a certain set of interaction capabilities to the user.

It is already visible, that research will move towards systems which enable the user to formulate multi–paradigm queries in order to further improve results.

As a consequence of the above, there is the need for

- A common mechanism for shipping multi–paradigm queries and their results , which assures that the right query processor processes the right query.
- For each paradigm a common language, which enables to formulate queries in this paradigm.

Fulfilling this need would also enable the different communities

- to share user interfaces. At present almost every group has its own interface suiting its purposes. However, many interfaces are very much alike.
- to share implementations without sharing the code. Comparing the resulting systems of different groups of a research domain would be easier, if one could make them accessible to outside scripts without publishing the actual code.

The query shipping–mechanism should be designed in a way that it does not constrain ongoing and future research regarding both the search for optimal query formulation for each paradigm and the research for new query paradigms for MMDB queries. In short, the query–shipping mechanism should *separate the communication problem from the query formulation problem*, letting research groups evolve freely to find good domain specific query formulation schemes.

MRML, as proposed in this paper, provides such a shipping–mechanism. In addition to the requirements stated above it was designed for making the use of it as simple as possible, thus allowing groups with exotic development environments and little manpower to be able to use MRML.

MRML in the current version provides a complete solution for QBE in CBIRS, which was the initial use of MRML.

MRML is an XML based language for the communication between MMDB server and user interface. In this paper we demonstrate the utility of the framework provided by MRML by giving the example how we use MRML in our *Viper* system.

The paper is organized as follows:
First we describe the general framework with emphasis on extensibility. After this we give the first application of MRML which is the use in our CBIRS systems together with the CIRCUS interface written by one of us (Z.P.).

## 2 The design of the MRML query shipping framework

Common positions always limit the freedom of the individual. However, in this case the design is easily extensible. In this section we propose a development strategy which will preserve the freedom of the individual research groups, while keeping the standard.

The primary goals of our design are:

1. Extensibility is, as we said above, our primary goal. Main problem here is to provide a framework which permits independent growth of the products of different research groups (followed by periodical code merging).
2. We want to leave the developer the freedom of choice of the implementation language. A standard like this is unlikely to be adopted by the research community, if it works only with a given “mainstream” computing environment.

3. We want the use of the communication protocol to be as independent from third party libraries as possible. A group should be able to provide its own tools within finite time.

Our choice is to use an XML (eXtensible Markup Language) DTD (Document Type Definition – a grammar) for the specification of our communication protocol, together with specifications for the transmission of messages, and for extensions of the protocol.

When making this choice we saw mainly the alternative of using EJB (Enterprise Java Beans), CORBA, and other methods of remote procedure calls. However, we feared strong links with languages (Java/EJB) or large program packages (CORBA). Moreover, the use of an XML application implies a common user log file format for multimedia databases. Please note that the DTD is a way of expressing the capabilities of client and server, *i.e.* they can *negotiate* the kinds of query that are allowed.

The attractiveness of XML is further increased by the existence of free tools in numerous programming languages. XML has been designed explicitly for simplifying parser design. XML has to be parsable by deterministic parsers, thus it is simple to implement one's own XML/MRML parsers.

## 2.1 The structure of XML and "graceful degradation"

The structure of XML is similar to that of HTML, which stems from their common ancestry, *i.e.* SGML (Standard Generalized Markup Language): an XML document can be seen as a tree of "elements" which themselves contain other elements. The content of each node of the document tree is a list of attribute-value pairs, as well as a sequence of nodes (possibly interleaved with text). This structure is encoded using so called "tags" for the elements. The "opening tag" of an element with type `t` and attribute `anAttribute` being set to `x` would be <`t anAttribute="x"`>. The "closing tag" of an element `t` would be </`t`>.

This free structure is constrained by a Document Type Definition (DTD) which is a grammar for the tree structure. The details can be found in [**?**].

"Graceful degradation" is the key to MRML's extensibility. It means *"build extensions in a way, that ignoring them causes minimal harm"*. Examples will be given in §6 which contains the MRML–DTD as well as its description. We believe that they demonstrate the feasibility of this approach.

## 2.2 The main MRML tags

Each message sent can be one of the following:

**ihandshake** The first text sent when the user connects to the database will be an ihandshake message. ihandshake contains the name of the user. Via the DTD one can detect the abilities of the interface.

**shandshake** The server will respond to the ihandshake message by giving a shandshake message. This message contains a list of sessions the user has done using the database, as well as a property sheet specification.

**Sessions** give the user the possibility to perform across–session learning using the system.

**MRML property sheets** are a description of parameters variables and their dependencies which permit the interface program to build property sheet GUI elements,

which will permit the user to configure the database freely. How freely is entirely the decision of the server. Using this mechanism we sidestep the problem of defining a set of common database configuration parameters. We also give the possibility to send interfaces of different complexity to different kinds of users, etc. .

**iconfiguration** The user is now free to use the configuration options at will. If so, at each step the interface will send an iconfiguration message. We require that this iconfiguration message will have to consist of at least the configuration of the algorithm used. However, as it was said before, more can be configured if necessary.

If the user does not use the configuration facilities, a set of default values is used, until the user does use the configuration facilities.

**inewsession,irenamesession** For session management the user can choose to open a new session (the old one will be closed then), or to rename the current one the non-existence of an iclosesession tag is not an accident: in a WWW scenario, we cannot to rely on users properly closing their sessions.

**iquerystep** Fill this at your will. In this paper we propose an XML-based query language for QBE in CBIRS which will be described at §??, page ??.

**iresult** The result is a list of URLs together with calculated similarities and information on which panel the information is to be displayed.

**error** Server and interfaces are able to send error messages to each other.

Each of the described messages uses other "helper"–XML–elements. The relationship is further described in the DTD.

## 3 Connection protocol of MRML client and Server

The interface will connect to a socket of the server, and send a text in MRML. After that it will wait for a response in MRML. After receiving the response the connection will be shut.

...more and some figures to come here...

## 4 MRML for interfacing and benchmarking in CBIRS

In current CBIRS research there emerge three groups of query techniques which are used in the different systems, most of the time in combination:

- Query By Example (QBE) and browsing queries: the user gives an image, and retrieves similar images using the system. He or she can increase the quality of the result using relevance feedback (for example [8]). As a modification of this scheme: "browsing queries", which could be summarized as QBE without first example (e.g. used by [3, 5, ?]). In any of these cases the user feedback is limited to stating the relevance of display items.

- Query By Sketch/Query By Segment: these systems require more interaction from the user, who has to draw an example or who has to mark regions of interest in the example he or she has given [1, 4].

- Annotation and Query on Annotation, eventually linking annotation to low–level features: this, essentially, is extending standard database technology to image databases [2, 7, ?].

As one can see, each of these groups necessitates different activity from the user, as well as a different method of query formulation. Video and audio, again, need other methods of interaction.

The problem of query by example in content based image retrieval systems, is particularly simple in terms of the interaction required. In this paper, we use QBE in CBIRS as an example for the use and usefulness of MRML:

MRML, as it is now, will be able to help the CBIRS community

- Improve interface design: Given that groups could improve a common interface rather than starting their own complete system from scratch, the design, as well as the usability of interfaces of MM databases should improve. The area of MM databases could as well profit from new breakthroughs in HCI, which are to be expected from research areas like e.g. emotional computing.

- MRML will help meta-query research: MRML as it is now allows simple scripts to redirect one query (QBE) to several servers and collect their results. Once other query paradigms are implemented in MRML, research on multi-paradigm and multimodal queries also will profit from the development

- MRML will facilitate the development of real–world applications which use new research results (e.g. an CBIRS–plugin to the GNU Image Manipulation Program [?]). This could be very helpful for evaluating the immediate use of current research.

- MRML will help exchange user data: The log files of interaction between user and content based multimedia retrieval systems are rarely exploited for improving the system [3]. Logging the queries would provide a common format which could easily be exchanged, thus giving the research area the possibility to benefit from collected experience, as suggested in [6]

- It is our opinion the most important point for the CBIRS community is that MRML could be used as a common interface to a common benchmark for CBIRS, which thus could easily be distributed and employed, thus supporting work on a common image collection and benchmark.

  We would like to emphasize that a common benchmark for image database is likely to evolve strongly within short time, and thus needs a flexible framework as a basis.

An immediate advantage for the CBIRS community would be the possible use of the CIRCUS (Lausanne) interface developed by one of the authors (Z.P.).

### 4.1 Expressing Query By Example in MRML

A query will essentially be a *tree* of URLs together with user given relevance judgments together with information to which panel the results should be sent. Why not a list? A tree permits us to shorten the message: If we want to send a query to several algorithms with slight modifications, we can formulate the common parts at the root of the tree, and modify them in the inner nodes and in the leaves of the tree.

**URLs** in order to keep things open for true multimedia querying, queries do not send binary data, but an URL as a pointer to such data.

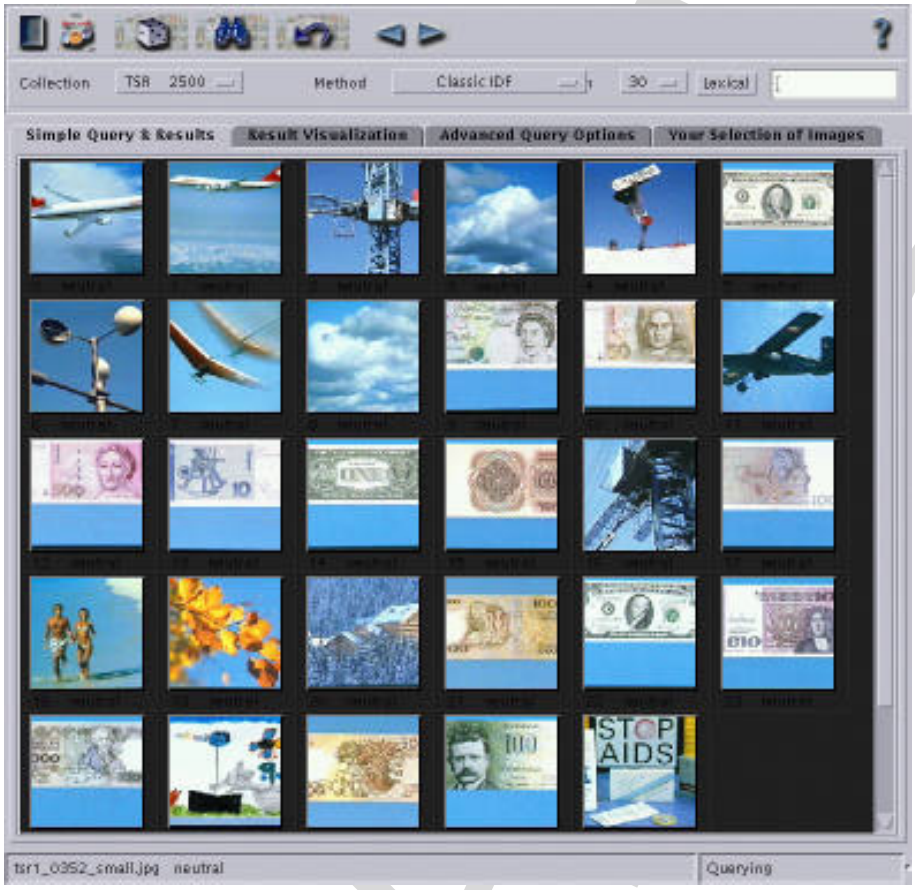


Figure 1: MRML-compliant CIRCUS interface after doing a query using a *Viper* server and database

**Panels** Panels give interface builders and users the possibility to ask query results distributed over several panels. [?], for example, is using this feature in their system.

# 5 Technicalities

## 5.1 MRML property sheets

MRML property sheets are a method to work around the fact that the a common set of configuration parameters for image databases is difficult to find and probably awkward to use. We suggest to achieve this by sending code which allows to build GUIs (*i.e.* the subset you would need for configuration of an algorithm), along with a specification of how to generate pieces of XML code from the GUI's state. This code is XML and it will not be executed, so, to our knowledge, there is no inherent security hole.

We would like to describe the property sheets by explaining needs of systems we know and how they are met by the property sheets.

### 5.1.1 A simple example

The "basic need" of a system would be to specify the collection, *i.e.* the database on which the retrieval is to be performed. For testing and comparison it would be interesting to have the choice between several algorithms (e.g. wavelet coefficient/color histogram based).

A choice out of a list of two elements:

```
<property id="p1"
          type="subset"
          caption="Collection"
          visibility="visible"

          sendtype="element"

          sendname="algorithm"

          minsubsetsize="1"
          maxsubsetsize="1">

          <property id="p2"
                    type="setelement"
                    caption="Dogs"
                    visibility="visible"

                    sendtype="attributetext"

                    sendvalue='id="algorithm1" collection="Dogs.db"'

                    defaultstate="selected"
          >
          </property>
          <property id="p3"
                    type="setelement"
                    caption="Cats"
                    visibility="visible"

                    sendtype="attributetext"

                    sendvalue='id="algorithm1" collection="Cats.db"' />

                    defaultstate="unselected"
          </property>
</property>
```

*What does this do exactly?*

- it defines a list of which the user is allowed to chose a `subset` of size between `1` and `1`, *i.e.* an exclusive choice.
- When asked for its state this list will generate an `element`, *i.e.* an opening tag and a closing tag. Content of this tag will be specified as content of the outer `property` tag (the one with id `p1`). The name of the generated element will be `algorithm`.
- The content of the the newly generated tag will be generated as follows: `property` Elements `p1` and `p2` are identical in structure. They denote the elements of our set

which can either be `selected` or `unselected`. *If selected* they send a text which will be placed like an attribute (`attributetext`). This text will be `'id="algorithm1" collection="Dogs.db"'` or `'id="algorithm1" collection="Cats.db"'`, depending on which of the two list items is chosen by the user.

As a result: the piece of MRML above will enable the interface to set up a property sheet which comprises a list of two items, of which one can be selected. Depending on the selection, the interface will send either

```
<algorithm id="algorithm1" collection="Dogs.db"></algorithm>
```

or

```
<algorithm id="algorithm1" collection="Cats.db"></algorithm>
```

to the server (with the appropriate surrounding tags, of course). *I.e.* there is only one algorithm, but the user is able to choose between two collections, and, as can be checked using the DTD, correct MRML will be generated.

### 5.1.2 More complex: a realistic case for many current systems

Consider the case most current research systems: one tends to be working on several algorithms, which are tested using different image collections of varying size and characteristics (*e.g.* the VisTex database [?] or parts of the Corel stock photo collection). Each of the algorithms investigated will have several parameters which one would like to be able to change during runtime, at least during testing and to get a feeling for the impact of each of the parameters on the query results. Usually the parameters involved will be not the same for each algorithm.

As a further complication, test data is not necessarily indexed for all query algorithms, necessitating a choice of algorithms which changes when choosing the collection or vice versa..

```
<property id="p1"
          type="subset"
          caption="Algorithm"
          visibility="popup"

          sendtype="element"

          sendname="algorithm"

          minsubsetsize="1"
          maxsubsetsize="1">

          <property id="p11"
                    type="setelement"
                    caption="18D continuous feature vector"
                    visibility="popup"

                    sendtype="attribute"
                    sendname="id"
```

```
                sendvalue="algorithm1"

                defaultstate="selected"
        >
                <property id="p111"
                        type="subset"
                        caption="Collection"
                        visibility="popup"

                        sendtype="attribute"
                        sendname="collection"

                        minsubsetsize="1"
                        maxsubsetsize="1"
                >
                        <property id="p1111"
                                type="setelement"
                                caption="Big Cats"
                                visibility="visible"

                                sendtype="value"
                                sendvalue="~viper/Big_Cats.db"

                                defaultstate="selected"
                        >
                        </property>
                        <property id="p1112"
                                type="setelement"
                                caption="Small Cats"
                                visibility="visible"

                                sendtype="value"
                                sendvalue="~viper/Small_Cats.db"

                                defaultstate="unselected"
                        >
                        </property>
                </property>
                <property id="p112"
                        type="subset"
                        caption="Metric"
                        visibility="popup"

                        sendtype="attribute"
                        sendname="metric"

                        minsubsetsize="1"
                        maxsubsetsize="1"
                >
```

```
                    <property id="p1121"
                              type="setelement"
                              caption="Euclidean"
                              visibility="visible"

                              sendtype="value"
                              sendvalue="1"

                              defaultstate="selected"
                    >
                    </property>
                    <property id="p1122"
                              type="setelement"
                              caption="Manhattan"
                              visibility="visible"

                              sendtype="value"
                              sendvalue="435"

                              defaultstate="unselected"
                    >
                    </property>
          </property>
</property>
<property id="p12"
          type="setelement"
          caption="Inverted file of discretized features"
          visibility="visible"

          sendtype="attribute"
          sendname="id"
          sendvalue="algorithm2"

          defaultstate="selected"
>
          <property id="p121"
                    type="subset"
                    caption="Collection"
                    visibility="popup"

                    sendtype="attribute"
                    sendname="collection"

                    minsubsetsize="1"
                    maxsubsetsize="1"
          >
                    <property id="p1211"
                              type="setelement"
                              caption="Dogs"
```

```
                    visibility="visible"

                    sendtype="value"
                    sendvalue="~viper/Dogs.db"

                    defaultstate="selected"
                >
                </property>
                <property id="p1212"
                    type="setelement"
                    caption="Small Cats"
                    visibility="visible"

                    sendtype="value"
                    sendvalue="~viper/Small_Cats.db"

                    defaultstate="unselected"
                >
                </property>
        </property>
        <property id="p121"
                type="subset"
                caption="Metric"
                visibility="popup"

                sendtype="attribute"
                sendname="metric"

                minsubsetsize="1"
                maxsubsetsize="1"
        >
                <property id="p1221"
                    type="setelement"
                    caption="Best fully weighted"
                    visibility="visible"

                    sendtype="value"
                    sendvalue="22"

                    defaultstate="selected"
                >
                </property>
                <property id="p1222"
                    type="setelement"
                    caption="Classic (tf.idf)"
                    visibility="visible"

                    sendtype="value"
                    sendvalue="5"
```

```
                                        defaultstate="unselected"
                                >
                                </property>
                        </property>
                </property>
</property>
```

The example above shows exactly the described scenario: The user has the choice between two algorithms, each of which can be configured by a simple choice, as in first example. However, we have to nest `property` elements deeper than in the last example.

Again the root tag of the property sheet, `p1`, designs a list, from which exactly one element has to be selected. When sending an XML message, it will send an `algorithm` tag. The first child of `p1`, `p11` is an element of the set described in `p1`. It designs the choice of an algorithm which involves 18-dimensional continuous feature vectors. In difference to the previous example `p11` itself has two children, `p111` and `p112`.

`p111` permits the choice of a collection, as described above, `p112` permits the choice of an additional parameter, the selected metric for determining the distance between the different images.

The same applies for `p12` and its descendants.

There are still two important topics:

1. *What happens in terms of dialog dynamics,* i.e. *which elements will be visible and selectable under which conditions?*

   Each element of the property sheet is only selectable when the all the ancestors of the element are active and its parent is selected. We call a selectable element of a property sheet *active*.

   Because of the `visibility="popup"` declaration in `p11` and `p12`, `p111` through `p112` will only pop up, if they are active.

2. *Which XML will be generated from this property sheet?* In general, XML is only generated by elements that are active. `p1` will open an XML element with the name `algorithm`. `p11`, if active, will add an `attribute` with the name `id` and value `algorithm1`. `p111`, will add the `attribute collection`, the `value` of which would be either `~viper/Big_Cats.db` or `~viper/Small_Cats.db`, depending on which of `p1111` or `1112` is selected. In a completely analog way `p112` will generate the `attribute metric`.

So in this example, texts of the form

```
<algorithm id="algorithm1" collection="~viper/Small_Cats.db"
metric="1">
</algorithm>
```

will be generated. At the same time sensible dialog dynamics will be assured.

### 5.1.3 A more formal description of MRML property sheets

As it has become clear from the examples, GUIs sent using MRML property sheets are in fact a tree of property sheets. Both the XML generated by the property sheet and the dialog dynamics are defined using simple rules.

**Dialog dynamics** A `property` element is *visible on the screen*, if

1. all its ancestors are visible
2. AND
   - its parent is non–selectable OR selected
   - OR its parent has the `visibility="visible"` attribute set.

*selectability* of `property` elements will be defined below.
A `property` element is `active`, if

1. all its ancestors are active
2. AND its parent is non–selectable OR selected

An active `property` element is defined as an element that can be *used for its purpose*, *i.e.* it will be enabled on the GUI screen.

**Generating XML** XML is generated during a depth-first-traversal of the `property` tree as follows:

- The XML string generated by a sequence of active elements is equal to the concatenation of the XML strings generated by each element. The sequence of concatenation is equal to the physical sequence of `property` elements in the MRML text.
- The XML string generated by an inactive element is empty.
- The XML string generated by an active element is given by the `sendtype` of the `property` element

  `sendtype="element"`: If there is any beginning of an opening tag in the XML generated by the ancestors of this `property` element, it will be ended by adding an `>` to the text generated so far.

  Afterwards this `property` element will generate the beginning of the opening tag of an XML element with a name that is specified by the attribute `sendname`, followed by a space and the content of the attribute `sendvalue`. As an example: if for a given element the `sendname` attribute has the value `xxx`, and the content of the `sendvalue` attribute is `'myattribute="5"'`, the generated output will be `<xxx␣myattribute="5"`.

  After that the children are evaluated in sequence and a closing tag of the element will be generated. Before that, the opening tag will be ended, if necessary.

  (`/</xxx>/`, in our example)

  `sendtype="attribute"`: If there is no beginning of an opening tag in the XML generated by the ancestors of this property sheet no text will be generated.

  If there is any beginning of an opening tag in the XML generated by the ancestors of this property sheet there are the following possibilities:

  `value` **is nonempty** Generate the text given by the values of the attributes `sendname` and `sendvalue` in the definition of this `property`. For example `sendname="myattribute" sendvalue="33"` will lead to the text `myattribute="33"` as output.

`value` **is empty** begin an attribute definition with a name given by the value of the attribute `sendname`. For example `sendname="myattribute" sendvalue=""` will lead to the text `myattribute=` as output. The actual definition of the value can be provided in two ways:

- If the current `property` has an inherent value (*i.e.* is `numeric`, `boolean` or `textual`), this value is taken, and thus the attribute definition will be ended.
- The value definition will be provided by a child.

`sendtype="value"`: If there is no attribute definition, which has been begun by any ancestor or sibling of this `property` element, no text is generated.

Otherwise either the inherent value or the value given by the attribute value `sendvalue` will be used, as described above.

`sendtype="attributetext"`: If there is no beginning of an opening tag in the XML generated by the ancestors or siblings of this property sheet no text will be generated.

Otherwise If there is any attribute definition which has been begun by an ancestor or sibling but which has not yet been closed, close this attribute definition by adding `""␣`.

Furthermore, add the text given by the content of the attribute `sendvalue` to the current begun XML tag.

`sendtype="children"`: This `property` element will only concatenate the XML code generated by its children.

`sendtype="none"`: This `property` element will not generate any code. It is only useful when using it with `type="reference"` which will be be described below.

**The building blocks of property sheets**

#### 5.1.4 `type="reference"`: Enabling nesting of algorithms

...to be continued here...

## 6 Examples for extended MRML

...to follow soon.

## 7 Future work in the specification of MRML

There are two main directions concerning further work on MRML and related goals.

1. Enhancing MRML: it is already clear at the time of writing, that MRML as is, very flexible, already very useful, but incomplete. We need to incorporate (at least):

   - region queries
   - text in queries

   Here we are hoping for cooperation with working groups who are using these query techniques in their systems. We would like to make MRML a "living standard", always keeping language specification and implementation date close together.

   An analysis of what could be done in the future can be found in [?].

2. Providing tools: In our opinion the best way of using the advantages created by MRML is to pool common tools which can be used and exchanged within the research community.

## 8 State and future of the implementation

At the time of writing we have a server which is already running under a "light" version of MRML (dubbed 0.8, without property sheets, but the query formulation much the same as presented in Appendix A), and the CIRCUS interface using this version of MRML. Until the middle of November both will be extended to the current version of MRML. Both the MRML treating code of the server as well as the interface will be published under (L)GPL then.

## References


[1] Serge Belongie, Chad Carson, Hayit Greenspan, and Jitendra Malik. Color- and texture-based image segmentation using EM and its application to content-based image retrieval. In *Proceedings of the International Conference on Computer Vision (ICCV'98)*, Bombay, India, January 1998.

[2] Ingemar J. Cox, Joumana Ghosn, Matt L. Miller, Thomas V. Papathomas, and Peter N. Yianilos. Hidden annotation in content based image retrieval. In *IEEE Workshop on Content-based Access of Image and Video Libraries (CBAIVL'97)*, pages 76–81, June 1997.

[3] Ingemar J. Cox, Matt L. Miller, Stephen M. Omohundro, and Peter N. Yianilos. Target testing and the `PicHunter` Bayesian multimedia retrieval system. In *Advances in Digital Libraries (ADL'96)*, pages 66–75, Library of Congress, Washington, D. C., May 13–15 1996.

[4] Myron Flickner, Harpreet Sawhney, Wayne Niblack, Jonathon Ashley, Qian Huang, Byron Dom, Monika Gorkani, Jim Hafner, Denis Lee, Dragutin Petkovic, David Steele, and Peter Yanker. Query by image and video content: The QBIC system. *IEEE Computer*, 28(9):23–32, September 1995.

[5] Wolfgang Müller, David McG. Squire, Henning Müller, and Thierry Pun. Hunting moving targets: an extension to Bayesian methods in multimedia databases. In Sethuraman Panchanathan, Shih-Fu Chang, and C.-C. Jay Kuo, editors, *Multimedia Storage and Archiving Systems IV (VV02)*, volume 3846 of *SPIE Proceedings*, Boston, Massachusetts, USA, September 20–22 1999. (SPIE Symposium on Voice, Video and Data Communications).

[6] Rosalind W. Picard. Toward a visual thesaurus. Technical Report 358, MIT Media Laboratory Perceptual Computing Section, 20 Ames St., Cambridge MA 02139, 1995.

[7] John R. Smith and Shih-Fu Chang. VisualSEEk: a fully automated content-based image query system. In *The Fourth ACM International Multimedia Conference and Exhibition*, Boston, MA, USA, November 1996.

[8] J. Vendrig, M. Worring, and A. W. M. Smeulders. Filter image browsing: Exploiting interaction in image retrieval. In Dionysius P. Huijsmans and Arnold W. M. Smeulders,

editors, *Third International Conference On Visual Information Systems (VISUAL'99)*, number 1614 in Lecture Notes in Computer Science, pages 147–154, Amsterdam, The Netherlands, June 2–4 1999. Springer-Verlag.

# A The DTD of MRML

Here is the (at present partly) documented DTD of MRML:

```
<?xml encoding="US-ASCII" ?>
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

       Basic structure:
       ----------------------------------

       Messages are sent as MRML texts.
       In  order to make it easy for the server to know who connects,
       each message is assigned the id of its session as an attribute.


       The following
     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->
<!ELEMENT mrml (ihandshake
               |shandshake

               |iconfiguration

               |inewsession
               |irenamesession

               |iquerystep
               |sresult

               |error
               )>
<!ATTLIST mrml sessionid ID "default">

<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

       The Viper property sheet specification
       --------------------------------------

       Basic idea: send a property sheet together with a specification
                   what should be the XML output coming back.
                   Useful for configuring your database.

       If this seems too complex for your case, please look at the
       cheatsheet we prepared for you. (i.e. probably you will be able to do
       what you want using a small subset of the features offered here)
```

```
     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->
<!ELEMENT property (property)*>
<!ATTLIST property id ID                                  #REQUIRED
                   type (multiset

                         |subset
                         |setelement
                         |boolean
                         |numeric
                         |textual

                         |clone
                         |reference)                     #REQUIRED

                   caption  CDATA                         #IMPLIED

                   visibility (popup|visible)             "visible"

                   sendtype  (element
                              | attribute
                              | value
                              | attributetext
                              | children
                              | none)              #REQUIRED
                   sendname  CDATA                        #IMPLIED
                   sendvalue CDATA                        #IMPLIED

                   minsubsetsize CDATA                    #IMPLIED
                   maxsubsetsize CDATA                    #IMPLIED

                   from CDATA                             #IMPLIED
                   step CDATA                             #IMPLIED
                   to   CDATA                             #IMPLIED

                   defaultstate  CDATA                    #IMPLIED
>
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

       END: The Viper property sheet specification

     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->



<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

     iconfiguration
     --------------
```

```
     Sending the configuration from the interface
     to the server: at present just the "algorithm"

     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++--->

<!ELEMENT iconfiguration (algorithm)>

<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

     END: iconfiguration

     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++--->

<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

     An algorithm will be either an empty element with
     attributes (add some at your will, it will talk with your
     server anyway, and this is the server which sent the property
     sheet), or a tree of algorithms.

     What is the use of this?
     Think of configuring meta queries. Together with properties
     you get a powerful tool.

     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->
<!ELEMENT algorithm (algorithm*)>
<!ATTLIST algorithm   id           ID    #REQUIRED
                      collectionid CDATA "default">



<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

     Handshake between server and interface
     --------------------------------------

     As a sign, it wants the connection, the interface sends
     a message first. The server responds, specifying its
     capabilities

     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->
  <!-- ++++++++++++++++++++++++++++++++++++++++

       Interface side: we send our name.
       which protocol is spoken by the interface is coded in the
       DTD.

       ++++++++++++++++++++++++++++++++++++++++--->
```

```
    <!ELEMENT ihandshake EMPTY>
    <!ATTLIST ihandshake username CDATA #REQUIRED>

  <!-- ++++++++++++++++++++++++++++++++++++++++

       Server side: the server sends us
       one list and a property sheet containing information about

       - the available sessions for the user
         (what are sessions? description follows)
       - the available algorithms and how they can be combined
       - which algorithm can be applied on which data collections

       plus, of course, information how to give back this information
       to the server.

       Note, that the property sheet formalism would be flexible enough
       to do all this with just one property sheet. However, we regarded
       it useful to add some structure.

       ++++++++++++++++++++++++++++++++++++++++ -->

    <!ELEMENT shandshake (ssessionlist,
                          salgorithmproperty)>

    <!ELEMENT ssessionlist   (ssession+)>
    <!ELEMENT ssession       EMPTY>
    <!ATTLIST ssession       id          CDATA "default"
                             displayname CDATA "Default session">

    <!ELEMENT salgorithmproperty (property)>

<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

     END: Handshake between server and interface

     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->

<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

     Beginning and renaming sessions
     -------------------------------

     We want to give the user the possibility to save the current
     state into "sessions". This might be useful in the case that
     a user has several classes of goals which s/he knows in advance.

     The user can request a new session.
     S/he can also request a name change for a session.
```

```
    Ending sessions is implicit:
    we cannot afford being dependent on the user ending
    his/her session in a "nice" way, so we do not
    tempt programmers to do so

    ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->
 <!-- Interface side -->
  <!-- send the desired feedback method together with
       a name for the session -->

    <!ELEMENT inewsession     EMPTY>
    <!ATTLIST inewsession     sessionname CDATA #IMPLIED>

    <!ELEMENT irenamesession     EMPTY>
    <!ATTLIST irenamesession     id          CDATA #REQUIRED
                                 sessionname CDATA #REQUIRED>


 <!-- -->

 <!-- Server side -->
  <!-- confirms session operation by sending name and id -->
    <!ELEMENT ssession     EMPTY>
    <!ATTLIST ssession     id CDATA   #REQUIRED
                           name CDATA #REQUIRED>

<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

    END: Beginning and renaming sessions

    ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->

<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

    Simple user commands
    --------------------

    (like e.g. back or forward
        in the command history)
    (at present the only commands)
   ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->

    <!ELEMENT iusercommand  EMPTY>
    <!ATTLIST iusercommand  command (backward|forward) "backward"
                            steps   CDATA              #IMPLIED>

<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
     END: Simple user commands

    ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->

<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

     A query step
     ------------

     A query is formulated using a tree structure. The semantics
     of this is:

     A parameter set in the root of a subtree will be passed on
     to the subtree. Attributes can be overridden.

     Each query has the possibility to say, which panel the result
     is supposed to belong to.

     We define the panel "P1" being the default panel,
     thus freeing small programs from the obligation to
     specify this.


     At present we are only providing definitions for the "query" for
     random images as well as for a quer by example(s).

     With this construction
     one is able to do quite complex
     things (fun stuff!)

     We require at each query step and each level af the "query tree"
     the algorithm and the collection to be defined.

     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->

    <!ELEMENT iquerystep   (iuserrelevancelist?,
                            iquerystep*)
                           >
    <!ATTLIST iquerystep   stepid        CDATA #REQUIRED
                           panel         CDATA "P1"

                           resultsize    CDATA #IMPLIED
                           resultcutoff  CDATA #IMPLIED

                           querytype     (query|atrandom) "query"

                           algorithmid   CDATA #IMPLIED
                           collectionid  CDATA #IMPLIED
                           >
```

```
<!-- List of URLs with user given relevances
     Our way of specifying a QBE for images.

     relevances vary from 0 to 1
     -->
    <!ELEMENT iuserrelevancelist    (iuserrelevanceelement+)>

    <!ELEMENT iuserrelevanceelement EMPTY>
    <!ATTLIST iuserrelevanceelement userrelevance CDATA #REQUIRED
                                    imagelocation CDATA #REQUIRED>

<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

     END: A query step.

     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->

<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

     Results for queries
     -------------------

     each result image can be accompanied by the user given relevance,
     as well as the similarity calculated by the program, based on the
     feature space.

     calculated similarities vary from 0 to 1

     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->

    <!ELEMENT sresult  (sresultelementlist?,sresult*)>
    <!ATTLIST sresult  panel CDATA "P1">

    <!ELEMENT sresultelementlist (sresultelement+)>
    <!ELEMENT sresultelement EMPTY>
    <!ATTLIST sresultelement userrelevance        CDATA #REQUIRED
                             calculatedsimilarity CDATA #REQUIRED
                             imagelocation        CDATA #REQUIRED>


<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

     END: Results for queries

     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->
```

```
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

     Error messages
     --------------

     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->
    <!ELEMENT error EMPTY>
    <!ATTLIST error message CDATA #REQUIRED>
```