

MRML Specification: Version 1.0

November 8, 2003

Contents

1	Aims	2
2	Requirements	2
3	Introduction	2
4	Connection request	3
4.1	Client connection request	3
5	Connection	3
6	Server response	4
6.1	Collections	4
6.2	Algorithms	4
7	Property sheets	5
7.1	A very simple example	5
7.2	More complex: generate XML subtrees	6
8	Query	8
9	Extensions	9
10	Examples	9

- **Status:** Final
- **Requirements**¹
- Hard copy: PS file², PDF file³
- Structure (DTD⁴, XML Schema⁵)

1 Aims

MRML is the Multimedia Retrieval Markup Language. The aim is to standardize access to Multimedia Retrieval software components. MRML is an XML⁶-based protocol. It corresponds to a well-defined DTD (*resp.* schema).

2 Requirements

- **Extensibility:** Our main concern was to provide a framework which permits independent growth of the products of different research groups (followed by periodical code merging).
- **No preferred implementation language** 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.
- **Independence of third-party libraries:** 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.

3 Introduction

MRML-based communications have the structure of a remote procedure call: the client connects to the server, sends a request, and stays connected to the server until the server breaks the connection. The server shuts down the connection after sending the MRML message which answers the request.

MRML, in its current specification (and implementation) state, supports the following features:

- request of a capability description from the server,
- selection of a data collection classified by query paradigm; it is possible to request collections which can be queried in a certain manner,

¹URL: Section: ??(Requirements)

²URL: [http://www.mrml.net/specification/v1\(underscore\)0/MRML\(underscore\)v10.ps](http://www.mrml.net/specification/v1(underscore)0/MRML(underscore)v10.ps)

³URL: [http://www.mrml.net/specification/v1\(underscore\)0/MRML\(underscore\)v10.pdf](http://www.mrml.net/specification/v1(underscore)0/MRML(underscore)v10.pdf)

⁴URL: [http://www.mrml.net/specification/v1\(underscore\)0/MRML\(underscore\)v10.dtd](http://www.mrml.net/specification/v1(underscore)0/MRML(underscore)v10.dtd)

⁵URL: [http://www.mrml.net/specification/v1\(underscore\)0/MRML\(underscore\)v10.xsd](http://www.mrml.net/specification/v1(underscore)0/MRML(underscore)v10.xsd)

⁶URL: <http://www.w3.org/XML/>

- selection and configuration of a query processor, also classified by query paradigm; MRML also permits the configuration of meta-queries during run time,
- formulation of QBE queries,
- transmission of user interaction data.

The final feature reflects our strong belief that affective computing will soon play a role in the field of content-based multimedia retrieval. MRML already supports this by allowing the logging of some user interaction data. In particular, this is the case for the history-forward and history-backward functionalities of the SnakeCharmer interface.

4 Connection request

4.1 Client connection request

An MRML server listens on a port for MRML messages on a given TCP socket. When connecting, the client requests the basic properties of the server, and waits for an answer. Skipping standard XML headers, the MRML code looks like this:

```
<mrml>
  <get-server-properties/>
</mrml>
```

The server then informs the client of its capabilities. This message is empty in the current version of MRML, but it allows for the extension of the protocol:

```
<mrml>
  <server-properties/>
</mrml>
```

Using similar simple messages, the client can request a list of the collections available on the server, together with descriptions of the ways in which they can be queried.

5 Connection

The client can then open a session on the server, and configure it according to the needs of its user (interactive client) or its own needs (eg meta-query agents). The client can also request the algorithms which can be used with a given collection:

```
<mrml>
  <get-algorithms collection-id="collection-1"/>
</mrml>
```

This request is answered by sending the corresponding list of algorithms. This handshaking mechanism allows both interactive clients and programs (such as meta-query agents or automatic benchmarkers) to obtain information describing the server.

In a similar simple manner, the client can open and close sessions for a user, and configure the algorithms chosen by the user. This enables multi-user servers and also on-the-fly learning by the query processor.

6 Server response

As a basic response the server details its capabilities to the client so that the client can adapt consequently. This response is mostly concerned with two aspects of the server's capabilities. Namely, the collections the server has access to and the algorithms that can be used to search these collections.

6.1 Collections

By default, the server returns basic collection information under the form of a collection-list:

```
<collection-list>
  <collection
    collection-id="c-tsr500-id"
    collection-name="TSR500"/>
</collection-list>
```

In an extended setup (eg, that of), one may this of adding more relevant information. For example attributes like: viper-number-of-image = "500"

6.2 Algorithms

Similarly, the server will return all the required information for accessing available algorithms that are available for interacting with (eg, querying) the collection. This is done via the definition of an algorithm-list of algorithm. For example:

```
<algorithm-list>
  <algorithm
    algorithm-id="a-idf-id"
    algorithm-name="Classical TF/IDF"
    algorithm-type="adefault"
    collection-id=" c-tsr500-id ">
    <property/>
  </algorithm>
</algorithm-list>
```

7 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.

7.1 A very simple example

is a system which uses inverted files for the indexation of images. Each image is translated in a variablelength sequence of features which describe the image. Each feature is assigned a weight determined dependent on the frequency of the feature within the image and within the collection. How exactly this is done, depends on the weighting functions. Both retrieval performance and processing speed of the system depend on the weighting function. gives the possibility to choose the weighting function at runtime, using an attribute cui-weighting-function of the algorithm element. The following property sheet gives the possibility to choose between two weighting functions. 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="Weighting function"
  visibility="visible"
  sendtype="attribute"
  sendname="cui-weighting-function"
  minsubsetsizesize="1"
  maxsubsetsizesize="1">
  <property
    id="p2"
    type="setelement"
    caption="Best fully weighted"
    visibility="visible"
    sendtype="value"
    sendvalue="best-fully"
    defaultstate="selected">
  </property>
  <property
    id="p3"
    type="setelement"
    caption="Classical IDF"
    visibility="visible"
    sendtype="value"
```

```

    sendvalue="classical-idf"
    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 attribute, i.e. the text given by send-name, plus cui-weighting-function. The value of the attribute will be determined 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 value (value). This text will be "best-fully" or "classical-idf", 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 cui-weighting-function="best-fully" or cui-weighting-function="classical-idf" to the server. The MRML client will use this property sheet when generating a configure-session message.

7.2 More complex: generate XML subtrees

The following example describes the generation of whole document subtrees. This feature is not yet immediately useful for or CIRCUS. However it provides an explanation on how the text generated in the previous is included into configure-session message. More important is the fact that it provides a general framework for describing (GUI) entities which can send XML. Consider the following example: Imagine an algorithm which runs the query image through a series of filters before running them through a simple query processor. Being a research system, we would like these filters to be run-time-configurable. Each filter needing some parameters, the and number of filters being variable, we simply need to define some new MRML tags which permit us to describe the sequence of filters. We would an output like the one given below:

```

<cui-filter-list>
  <cui-filter
    cui-filter-type="horizontal-gabor"
    cui-filter-gabor-sdev="50"
    cui-filter-gabor-wavelength="10"/>
  <cui-filter
    cui-filter-type="gauss"
    cui-filter-gauss-sdev="5"/>
</cui-filter-list>

```

The corresponding property sheet would look like:

```

<property
  id="p1"
  type="panel"

```

```

caption="Filter Sequence"
visibility="invisible"
send-type="element"
send-name="cui-filter-sequence">
<property
  id="p1"
  type="multi-set"
  caption="Filter"
  visibility="visible"
  send-type="element"
  send-name="cui-filter"
  minsubsets="0"
  maxsubsets="5">
<property
  id="p11"
  type="set-element"
  caption="Gaussian blur"
  visibility="pop-up"
  sendtype="attribute"
  sendname="cui-filter-type"
  sendvalue="gauss"
  defaultstate="selected">
<property
  id="p111"
  type="numeric"
  caption="Standard deviation"
  visibility="pop-up"
  sendtype="attribute"
  sendname="cui-filter-gauss-sdev"/>
</property>
<property
  id="p12"
  type="set-element"
  caption="Horizontal Gabor"
  visibility="pop-up"
  sendtype="attribute"
  sendname="cui-filter-type"
  sendvalue="horizontal-gabor"
  defaultstate="selected">
<property
  id="p121"
  type="numeric"
  caption="Tile size"
  visibility="pop-up"
  sendtype="attribute"
  sendname="cui-filter-gabor-sdev"
  numeric-from="5"
  numeric-to="100"
  numeric-step="5"/>
</property>

```

```

    id="p122"
    type="numeric"
    caption="Tile size"
    visibility="pop-up"
    sendtype="attribute"
    sendname="cui-filter-gabor-wavelength"
    numeric-from="2"
    numeric-to="20"
    numeric-step="1"/>
  </property>
</property>
</property>

```

The example above shows exactly the described scenario: The user has the choice to use sequences of 0 up to 5 filters. The filters can be either Gaussian blur or horizontal gabor filters (yes, this is a toy example). The Gaussian blur can be configured by giving a number between 1 and 100, which will be sent as an attribute (cui-filter-gauss-sdev). The gabor filter can be configured using the two parameters cui-filter-gabor-sdev and cui-filter-gabor-wavelength. Both the configuration panels will pop-up when the corresponding filter has been selected in the sequence. In the following section we describe how the text is actually generated, and how dialog dynamics is specified.

8 Query

The query step is dependent on the query paradigms offered by the interface and the search engine. MRML currently includes only QBE, but it has been designed to be extensible to other paradigms.

A basic QBE query consists of a list of images and the corresponding relevance levels assigned to them by the user. In the following example, the user has marked two images, the image 1.jpg positive (user-relevance="1") and the image 2.jpg negative (user-relevance="-1"). All query images are referred to by their URLs.

```

<mrml
  session-id="1"
  transaction-id="44">
  <query-step
    session-id="1"
    resultsize="30"
    algorithm-id="algorithm-default">
  <user-relevance-list>
    <user-relevance-element
      image-location="http://viper.unige.ch/1.jpg"
      user-relevance="1"/>
    <user-relevance-element
      image-location="http://viper.unige.ch/2.jpg"
      user-relevance="-1"/>
  </user-relevance-list>

```

```
</query-step>
</mrml>
```

The server will then return the retrieval result as a list of images, again represented by their URLs.

Queries can be grouped into transactions. This allows the formulation and logging of complex queries. This may be applied in systems which process a single query using a variety of algorithms, such as the split-screen version or the system described by Lee . It is important in these cases to preserve in the logs the knowledge that two queries are logically related one to another.

9 Extensions

In order to demonstrate how easily MRML can be extended to other query paradigms, we give as an example QBE for images with user annotation. We assume that the user is invited to associate textual comments with images he or she marks as relevant or irrelevant. Since a tag for this purpose does not yet exist in MRML, we add an attribute `cui-user-annotation="..."` to the element. The prefix `cui-` is added to avoid name clashes with extensions from other groups which use MRML.

```
<user-relevance-list>
  <user-relevance-element
    image-location="file:/images/1.jpg"
    user-relevance="1"
    cui-user-annotation="tropical fish"/>
</user-relevance-list>
```

It is important to note here that servers which do not recognize the `cui-user-annotation` attribute still can make use of the remaining information contained in the `user-relevance-element` element.

As an example of how *not* to extend MRML, we give an extension with the same semantics but which does not respect the principle of graceful degradation:

```
<user-relevance-list>
  <cui-user-relevance-element
    image-location="file:/images/1.jpg"
    user-relevance="1"
    user-annotation="tropical fish"/>
</user-relevance-list>
```

Instead of adding an *attribute* to an existing MRML element (`user-relevance-element`), a new element was defined that contained the same kind of extension, namely `cui-user-relevance-element`. Consequently, servers which do not recognize this element will not be able to exploit any relevance information.

10 Examples