University of Geneva
Center for
Computer Science

Computer Vision and
Multimedia Laboratory

Date: March 14, 2003

Number: 03.02

Technical Report
Vision

# MRML: Steps towards version 2

Stéphane Marchand-Maillet*

marchand@cui.unige.ch

http://viper.unige.ch/

*This work was supported by the NCCR IM2 and EU-IST M4 and Webkit projects.

# Steps towards version 2

| | | |
|---|---|---|
| **Author** | Stéphane Marchand-Maillet<br>marchand@cui.unige.ch | *Viper* group<br>http://viper.unige.ch |
| **Document Identification** | Technical Report 03.02 | Computer Vision and Multimedia Laboratory – University of Geneva |
| **Version** | Revised | March 14th, 2003 |

**Abstract**
This document is a first step toward a specification of a new version the Multimedia Retrieval Markup Language (MRML). Previous specifications are available in the home site http://www.mrml.net/. MRML mostly aims at three operations: creating a consistent connection between a search client and a query processing engine, acting as server, shipping actual queries and results and supporting administration tasks at the server level.
This document steps back from current specifications and implementations and presents an analysis of what MRML should be and also should not be. It then builds on a formal analysis and derives properties that should be fulfilled by MRML. While doing so, it identifies places where MRML may reuse already specified components, including from W3C.
This document concludes with a presentation of a formal XML syntax for MRML 2.0 that remains consistent with earlier versions. The document also proposes a scheme for an implementation that guarantees backward compatibility with earlier versions of MRML.

## Contents

## Introduction

This document is a first step toward a specification of a new version the Multimedia Retrieval Markup Language (MRML). Previous specifications are available in the home site http://www.mrml.net/. MRML mostly aims at three operations: creating a consistent connection between a search client and a query processing engine, acting as server, shipping actual queries and results and supporting administration tasks at the server level.
This document steps back from current specifications and implementations and presents an analysis of what MRML should be and also should not be. It then builds on a formal analysis and derives properties that should be fulfilled by MRML. While doing so, it identifies places where MRML may reuse already specified components, including from W3C.
This document concludes with a presentation of a formal XML syntax for MRML 2.0 that remains consistent with earlier versions. The document also proposes a scheme for an implementation that guarantees backward compatibility with earlier versions of MRML.

## 1. Scope of MRML:

MRML is intended to be as simple as possible, while providing powerful capabilities, for example:

- Permitting a flexible client-server architecture: This has been the driving aim for the development of MRML. The idea is that search technology components can be inter-related via a common protocol. Such components include search engines and query interfaces.
- Permitting post-search analysis: We see the information transiting between a search engine and a query interface as the core information that represents the interaction from the user and the search performance of the query engine. By recording this in a non-ambiguous fashion, we hope to create new analysis techniques.

MRML scope is mostly threefold:

- MRML should first allow a client and a server to synchronize in view of subsequent interaction. This synchronization boils down to informing the client of the server search capabilities. We define these capabilities as the description of a list of search algorithms and a list of collection searchable by selected algorithms. The reaction of this from the client should be a self configuration so as to be able to express queries, corresponding to the search algorithms described and receive and manage results.
- The latter introduces the second aspect of MRML, which is the wrapping of search queries and results. The idea here is to abstract search queries and results so as to wrap those within MRML and ship them on the MRML stream.
- Finally, MRML is completed by the ability of managing the server configuration and administration.

These two aspects are described in detail in the sequel. Before that, we consider as important to specify also what MRML is not (read also the "Why MRML and not...?" section):

- **MRML is not** a query language: MRML is not a query language in the sense defined by SQL for example. MRML can be extended to either embed queries (SQL queries for example). MRML may be extended to define a custom query language that can later be embedded in a core MRML sequence. It is not the aim of MRML to define on such query language. MRML define only a basic query protocol abstraction and provides specific definition accordingly.
- **MRML is not** only a communication protocol: MRML aims at being richer than the SOAP envelope. It is not just an empty shell that can contain information retrieval messages. While one aim is to carry messages from a client to a server, another important goal (perhaps as important) is to define an unambiguous structure to store these communications and enable their post-processing by automated analyzers.

A careful analysis of what MRML aims at doing shows that it can be summarized in a quite simple task (or state) list (see annexe 4 for details). In terms of global use cases, we can identify 4 major tasks:

- Configuring a server at startup
- Establishing a consistent connection between a server and a client

- Sending queries and receiving results
- Administrating the server

Similarly, in the process we define, a MRML-compliant server can essentially be to a limited number of states (see annex 4.2):

- Creating a connection with a client
- Setting up a session for querying
- Querying
- Disconnecting from a client
- Sending error messages

The structure of this document follows this analysis. In Section 2, we introduce some of the important terms that will be used throughout and require a formal definition. Section 3 addresses the problem of writing server configuration files in MRML. This is useful since this code can later be used to inform the client about the server's configuration. In Section 4, we see how a proper context for querying can be set up (connection and session management). Section 5 addresses the core problem of querying and managing results. In Section 6, MRML is used for server administration. Here again, several use cases are identified and catered for. Section 7 completes the analysis with a thorough error management framework.
While the analysis is done in a general manner (ie independent of the syntax chosen), it is important to finalize it with a full proposal for a XML syntax for MRML. This is presented in Section 8. Section 9 studies extensions mechanisms that are put into practice in typical examples in Section 10.
This document contains a number of Annexes that contain technical specifications and details that can be used as a reference to construct a practical MRML compliant-system.

## 2. Definitions

### *2.1 Glossary of terms*

Before proceeding further in the definition of MRML, we give some important definitions of concepts that will be extensively used later.

- Algorithm : A search algorithm is the implementation of a search technique within the server. An algorithm is the core processor of a query engine. In order to perform a search, this algorithm asks for the values of a number of parameters. These parameters should be described in a property sheet.
- Client : A search client is a piece of software that will send queries to the server in order to get response. The most obvious client is a graphical interface through which the user will input queries and receive result. However, MRML defines a more flexible concept for a client. It can be a part of a wider software (eg GIMP plugin) or a tool whose aim is not the search in itself (eg an automated benchmark).
- User : A user is a person or process that uses the client to formulate queries to the server and is proposed responses. In most cases, the user is the entity from which the analysis of search results is expected.
- Collection : A collection is a set of multimedia items that are gathered as a list of URLs. This collection is indexed by the server (by one or more of its algorithms) and therefore accessible for queries via different search techniques.
- Multimedia item : In the context of MRML, a multimedia item is anything that can be referred to by a URI. It is typically a piece of information. This information may be part of a wider multimedia item, as long as it is unambiguously referred to by the URI.
- Property sheet. A property sheet corresponds to a search algorithm. The aim is to transmit the algorithm parameters from the server to the client so that the latter can adapt accordingly. In this version of MRML, property sheets are deprecated and replaced by XForms.
- Query : A query is a request for information. Its formulation follows different techniques that correspond to the search targeted. Techniques include explicit formulation (SQL-like) or query-by-example (QBE).
- Server : An MRML-compliant server (resp client) is a piece of software that listens to a port (not fixed), digests MRML, parses it and responds to the query (resp displays the results). In other words, an MRML-compliant server is a piece of software that is able to answer a query wrapped within a MRML message and an MRML-compliant client is a piece of software that is able to process the results and send user

interaction (eg query) via MRML. In a more layered architecture, a server will make use of one of several query engine(s) to respond to the query.

- Connection: A connection to a MRML server is the set-up of an exchange procedure between the client and the server (identified by a IP address and a port).
- Session : A multimedia retrieval session is a consistent set of queries and responses that are made by a client to a given server. The main aim of defining a session is to be able to define a set of interactions as a consistent group and derive knowledge from it.
- Query engine: A piece of software that is able to understand a certain type of query and respond to it. A query engine will make use of an algorithm and most likely a data accessor that knows the specific data structure associated with the algorithm.

Now that the vocabulary and the context are fixed, we define the need for MRML. This is done in an abstract way in the three next sections. Typically, we wish to define and discuss what functionalities something like MRML should incorporate and where to stop. In this respect, the XML notation mostly is absent from the discussion since we consider that the syntax has little relevancy to the discussion. Rather, we wish to list the minimum information that is needed for a retrieval system to function. We do this in a generic context, while trying not to impose any constraint in the different aspect of the problem (eg type of media searched for, type of query technique, type of interaction).
Then, from this global view, we can propose an XML structure that will materialise MRML.
Four types of MRML excerpts can be identified:

- MRML messages: these are the most obvious ones, transiting from the server to the client and reverse.
- MRML configuration: this part stores a server information and typically includes what is sent to the client as a server description,
- MRML administration commands: here, MRML is used to manage the server. Typically, we wish here to create something like the functionalities offered by Unix signals;
- MRML error messages: It is important to have a formal way of handling all the types of errors that can be generated at both ends of the architecture. This part ensures the coverage of this type of problems.

### *2.2 Common grounds*

There is a number of common concepts that we will define across most definition. While it may be confusing to read in a first pass, it is useful to have a transversal definition for these items. Maintaining such a consistency makes it easier to grasp the specification since parts of the definitions become “expected”.

- ID This consists of a unique identifier of the associated item. It is a string that is normally constructed from the ID of the more general (or parent or container) concept added with a unique local ID. By nature, this owes to be included as an XML attribute attached to the element describing the concept;
- base-url: while we do not want to exchange all information related to an item or concept nor we want to constraint the syntax or length of the description, we define the base-url as the URL that will consist of the home of this item. It is different from the (possibly defined) URL of the item itself. Following the idea of the namespace URLs, it is not compulsory that the base-url attribute points to an existing location. However, if this location exists and the base-url does not explicitly point to a file, the place where it points should contain a file called description.xml (see below).
- description.xml: this is the default name of the file that will contain (in a rather self-documented fashion) some information related to the item whose base URL points to this location. This information is either meant to be read by the implementer of a related piece of software or acquired and automatically parsed for parameters. Note that this opens the way to a distributed description since the desciption.xml file may be a list of pointers to relevant pieces of information. Note that this file explicitly exploits the auto-descriptive nature of XML.
  As minimal contents and in order to be consistent with the rest of MRML we recommend that the description.xml file repeats a number of information attached to the item such as its ID, name and home URL.

## 3. The server configuration file

This file instructs the server of which of the media collections and retrieval algorithms are available to it. This file is not meant to manage the loading of plugins that materialize the algorithms. It is the responsibility of the actual server to make sure that the corresponding algorithm are implemented.
For most of it, this file contains the information that will be sent to the client for defining the server configuration. It will essentially contain three types of information: the server own details, the collection details and the algorithm details.

### *3.1 Server details*

What needs to be known from a multimedia retrieval server is essentially its IP address, port and name. Clearly, the IP address and port should be available to the client for creating the connection. Therefore, we will create the server's ID under the form of the string
IPaddress:port:name:version
where name and version refer to the name and version of the server software (eg GIFT:0.0.8). This will later be useful in the MRML session for identifying which server (type and location) was used for generating the MRML logs.
Beyond this basic information, we recommend that the server sends a base-url that will contain an information file description.xml (see section 2.2) written in XML. The aim of this file is to give (or point to) details that will help the client designer to better have access to the server. The presence of this file is optional (similar to the definition of an URL associated to a namespace). Typically, if present, the content of this description file should give:

- The ID of the server
- Its name
- Its description (eg author, software used)
- Its associated organization
- Possibly the URL of a stylesheet the client would use to create a query interface (eg comprising the server's logo).

As stated above, we wish to generalize the use of this base-url principle where every entity referred to (including server, client, algorithm, collection) in MRML possesses a base-url location that contains a description file itself explicitly listing or pointing to information that helps better understanding of the item in question. Via this extensible mechanism and thanks to the auto-documentation property of XML, detailing an item is not constrained nor limited.

### *3.2 Algorithm details*

The server can make use (implement) a number of retrieval algorithms whose parameters should be defined in an abstract way so as to enable their use from any client. In the context of MRML, an algorithm is a piece of software that accepts queries under a given form, using a number of parameters that should be described to the client for it to set them properly. In more details, the description of an algorithm comprises:

- ID
- Description
- Base-url
- Type
- Parameters
- The interaction technique associated to the algorithm

The parameters of an algorithm are a standard list of values of several types. We propose and recommend that such a list should be described with the XForms recommendation from W3C. In this recommendation, three concepts are defined

- The data model: it lists the parameters expected and their related types, as defined by the W3C type recommendation (and possibly extended).
- The data form: this is the way data should be inputted. This is useful for guiding the design of the client. Note that this does not constraint the GUI design but complements on the type of data that is expected.
- The data instance: this is the place where default values are set and incrementally replaced are user-given values.

We believe that XForms offer a suitable solution for the problem of transferring arguments of an algorithm, as defined in the MRML context. As mentioned in the recommendation:

> "The primary difference when comparing XForms with HTML Forms, apart from XForms being in XML, is the separation of the data being collected from the markup of the controls collecting the individual values. By doing this, it not only makes XForms more tractable by making it clear what is being submitted where, it also eases reuse of forms, since the underlying essential part of a Form is no longer irretrievably bound to the page it is used in.
> A second major difference is that XForms, while designed to be integrated into XHTML, is no longer restricted only to be a part of that language, but may be integrated into any suitable markup language."

In version 1.0 of MRML, property sheets were defined to transfer parameters from the client to the server. Here, we suggest that not only parameters need to be communicated but also interaction technique (see the XML (8) and Example (10) section for more details). Typically, in the QBE paradigm using relevance feedback, one should not only pass the parameters of the search engine but also what is expected as a response from the user (ie a relevance judgment for each returned item).

### *3.3 Collection details*

Through these algorithms, the server has access to a number of media collections. It is the role of this part to describe each of these collections so as to be able to formulate queries for these items. In MRML, a collection is a set of URLs, thus enabling the notion of distributed and/or heterogeneous collections. In more details, the description of a collection includes:

- The collection ID
- The collection name
- A base-url
- The type(s) of media involved (referring to MIME types)
- The size (eg number of items)
- The list of algorithms that allows access to this collection.

Note that although connection and sessions are separated concepts, they can be merged in their actual implementation.

## 4. Creating the right context for querying

Setting up the possibility for a client to query a server is done in a two-stage process. First the client and the server exchange properties and configure each other. This is what is called the connection process. Then, a user will start an MRML session. As stated earlier, the main goal of defining such a session is to label a set of interaction as being a consistent group for extended analysis.

### *4.1 Defining an MRML connection*

In MRML, we assume that the client has the knowledge of the IP address and port of the server. MRML does not try to impose the use of any specific port for querying. The use of specific ports such as 80 (HTTP port) can be controversial (see "Why MRML and not HTTP?", below).
The client first asks the server for a connection, with the knowledge of its IP address and port. At this stage, the client may send some relevant information so that the server can filter what type of information is needed by the client and which will be irrelevant. This may also be useful whenever the server wishes to make statistics on access or even restrict its access. Typically, what can be sent by the client is:

- Its ID
- Its name
- Its type (eg underlying language – Java, PHP, CGI,…)
- A base-url pointing to any other relevant information.

To this request, the server responds by sending its details that will help the client to configure itself in order to start a session. At this stage, the list of collections and available algorithms is sent.

*4.2 Defining an MRML session*

The connection is now open and the client and the server know each other's personal details. Via the client, a user will now be able to open a specific session on the server. This is done by requesting a new session or by instructing the sever that the user wishes to reuse a given session. If the proposed session can be reused, it is returned as current session otherwise a new session is created. If this new session cannot be created, an error message is return. In any case, a session has a unique identifier that we recommend to be a function of

- The server's ID
- The client's ID
- Its creation time

Once defined, this session will form the consistent framework within which the client will exchange MRML messages with the server, under the guidance of the user. The aim is to create the notion of session history. This can be useful for example for on-line learning via relevance feedback or off-line learning via posterior re-creation and analysis of sessions.
At this stage, it is worth mentioning the link that can be made between sessions and user profile management. User profiles are specific properties that can be attached to the user. These properties can either be given explicitly by the user or deduced from the sessions by any form of learning. In any case, the aim is to adapt the response to the user. In MRML though, it is not planned to manage user profiles as a part of MRML itself. However, via the concept of sessions (and mostly reusable sessions), it is possible to have user-personalised search contexts. In that respect, the session ID implicitly contains a user ID.
Finally, we can define a transaction as a consecutive pair of MRML messages exchanged between the client and the server within a MRML session.

## 5. Sending queries and receiving results

In the context of an MRML session, a user will then be able to formulate queries via the client (now properly configured) and receive results. Here again, we wish to define a minimal set of conceptual entities that need to be specified, in order to achieve this. It is mostly within this part that specific extensions will take place (see further below for the MRML extension mechanism).

*5.1 Sending queries*

In the context of MRML, a query is the expression for a request of information. We identify several ways of formulating such a query. The simplest example is that inspired by a SQL-like query language where the query is made by merely describing what type of information is expected. Another type of query is the Query-by-Example (QBE), where (positive and negative) examples are provided and the underlying query is "Find something that looks like this but not like this".
It is clearly not possible (and would be limitative) to list all possible types of queries and express them into MRML. Further, we insist that MRML is not a query language and should rather enable the shipping of queries of any types. However, since the examples cited above are fairly common, we feel it is important to simplify their formulation within MRML.
To this end, we define a set of classical query formulations that we hope will enable the expression of most of the queries. As specifications go by, we aim at integrating into MRML the most recognized and used types of queries (while still not making MRML a query language).

- Explicit query : This is the most basic case of query where a excerpt of an external query language is encapsulated by MRML. It is then the responsibility of the server to parse and process this specific query. The type of query (eq SQL) may be specified as a parameter
- Query-by-example : In the case of a QBE, the query is formulated by giving examples and associated relevance.
- Misc: Any basic query cannot be expressed using the above types is called a "misc" query. This type is typically the same as the explicit query type without any given type. The query excerpt is just wrapped within MRML.
- Complex : this term comes in opposition to the basic query above. A complex query is the combination of the above types.

Each of these query formulations first lead to the use of a given algorithm, itself leading to the choice of a particular interaction mode. For example, the use or not of relevance feedback may be specified. However, since this is a generic characteristic of the algorithm used, we leave it to the algorithm to define this relevant information. Nevertheless, MRML will still allow the expression of specific interaction modes, like relevance feedback.

### *5.2 Receiving results:*

Whereas the aim of the query is simply to express an information need, receiving the results will lead to essentially two actions. Firstly, the results need to be presented to the user as a response to its information need. Then a form of feedback may be expected. Here again, we leave it to the algorithm definition to define what type of interaction is expected from the user on the basis of given results.
Similarly, the results may be slightly more complex than a simple list of multimedia documents abstracted by URIs. It may well be the case that multimedia documents returned as a response to the information query are part of a complex organised structure and that structure is expected to be given to the client for the interaction step. An example of such a structure is as follows.

> The system has indexed a number a video sequences by characterising and indexing their scenes via keyframes, themselves associated with a textual annotation (eg the transcript of the audio contained in the scene they represent).
> The query may be done via keywords using the text retrieval algorithm. The client should then display the keyframes associated with the relevant text excerpts found as relevant to the text query. A link to the original scene and the original video should also be given for easy inspection.

There are different possibilities to handle such a scenario. The simplest and most valid one is to consider that, even if the expected interaction is quite complex, the system is still essentially made up of two mono-media sub-systems, a text retrieval engine and a image QBE engine. As such, each of these cases should be considered rather separately but chained to each other at the client level. In this case, the client will be an interface that will allow to make the link between video, scenes, keyframes and text via a database, for example. This is an example where the description file of the server, algorithms and collections may be used to detail the necessary information to setup such a client.
Another scenario that looks simpler but is actually not is the following:

> Suppose now that one wishes to make a query such as: “find the sequences where this person (photo) says that (text)”. We assume the existence of a multimedia query engine (algorithm) that will process simultaneously the text and the picture(s) to respond by a number of video sequences and selected text transcripts.

In this scenario, what is needed is the ability to send mixed queries consisting of different interaction modes applying on different medias. This is normally given by the complex query type and should pose no problem. In this case, the results received are not a chain of media linking to each other but rather a combination of media items that should be kept in groups.

## 6. Managing the server

Up to know we have specified the use of MRML in classical usage scenarios. Since the server will be by definition MRML-compliant, we think that it can be beneficial to include the ability to talk to the server in MRML for various administrative reasons.
Clearly this implies a number of security issues to be addressed and will impact on the MRML client-server architecture. Here, we mostly concentrate on defining appropriate MRML messages and assume that a security mechanism has been put into place. The default such mechanism is the use of a secured socket on a specific port via which administrative messages are inputted to the server.
We essentially group the use cases of administration within three categories.

### *6.1 Server administration*

Here, the messages are dedicated to manage the server as essentially any server. We would provide facilities such as

- Restart
- Shutdown
- Read the configuration files again
- Send information about
  - The configuration (log files, paths, …)

    - Users logged in (sessions open)
    - …
- Spawn a new process
- ….

*6.2 Data administration*

Here the idea is to handle the server as a tool to manage the data. This is the case when one wants to index a new data collection. This should be triggered by a MRML message giving details about the collection to index. Similarly, it should be possible to disable the access to a collection or algorithm via MRML.
*Note on multimedia data indexing.* In the same line, an interesting extension of MRML would propose a solution for the following indexing specific problem. As of now, an MRML-compliant server is designed to accommodate one or several search algorithms where each of these algorithms materialize a search technique based on a (possibly specific) data structure. When evolving the search technique, and since the indexing (calculation and filling of the data structure) is offline, there is no way to guarantee that the search is performed on a data structure that corresponds to the same version of the search technique. When imposing that the indexing is done by the server and triggered by MRML, this encourages the design where the writing and reading of the data structure are next to each other (eg implemented in the same object). Therefore, this encourages a design where whenever modifying the writing methods, one is forced to update the reading methods.
A consequence to this is that, providing that the indexing is part of the server, it could be organized into a feature extraction library where components could be reused for data characterization within several search techniques. A further extension of MRML could therefore experiment with the design of features by specifying feature definitions along with the indexing administration command. No such extension is defined in this document.

*6.3 User administration*

Here, we wish to enable user profiling and manage user rights. A number of user management facilities should be added and accessible via MRML such as:
- Add/remove a user
- Give/revoke rights to a user to index/remove a collection
- …

Note that user rights and their management are readily inspired from SQL-DB user right management model.

## 7. MRML error messages

MRML should be designed so as to feed back the errors from an MRML component to others. This in turn implies that MRML compliant software are resilient to these errors and can manage them efficiently.
We identify different types of errors:
- MRML well-formedness error: this is a type of error triggered by syntax errors in the XML code (XML not well formed). This type of errors should be captured by the XML parser and then passed onto the MRML processor
- MRML validity error: This is an error triggered by the non-compliance of the MRML excerpt to the definition given by the appropriate schema.
- MRML should also support error triggered by the non-match between an MRML request and the component capabilities.

## 8. Turning all this into XML

The above specifications could be mapped in several forms or concepts. In this respect, one may feel that all this exists under a form or an other in another language or protocol. We have listed a number of such possibilities in the “Why MRML and not … ?” section. Here, we choose to make MRML an XML-based protocol, in order to inherit from most XML good properties:
- Human-readability (even if heavy notation)

- Easy to binarise (see BSDL – Binary Stream Definition Language – for example)
- Facilitated parsing
- Auto-documentation: This will prove useful in the extension case.
- Available associated software: perhaps one of the main reasons
- XML is the best we know for doing what we want to do.

### *8.1 MRML namespace*

We feel it is essential for MRML to propose a proper namespace. We suggest simply to use the namespace mrml : xmlns:mrml="http://www.mrml.net/ns" so that elements and attribute names can be prefixed by this namespace.

### *8.2 MRML envelope and version enumeration*

As required in the XML Protocol Requirements (to which we think that MRML mostly complies), MRML defines an envelope, an outermost tag that surround MRML content. We suggest that this surrounding tag should be <mrml/>.

Any MRML message should show its version. The version should come as an attribute of the envelope of the message. Since this was not the case in earlier versions, in the case of no explicit version statement, version 1.0 is taken as default.

It is important to note that MRML will be specified within an XML Schema Definition (XSD). Implicitly, the association between the XML document (MRML message) via the XML Schema Instance mechanism will associate a MRML version to the MRML message. In this case, it is this version that takes priority on the version attribute of the MRML envelope. This decision comes from the pragmatic point of view that the xsi mechanism will trigger validation in standard XML software and therefore should not be shortcut or duplicated.

Hence, a typical MRML message should be contained a tag set that resembles:

```
<?xml version=”1.0”?>
<mrml xmlns:mrml="http://www.mrml.net/ns/" version="2.0">
  <!--Any MRML portion -->
</mrml>
```

Note that in the above excerpt, mrml stands for the envelope and for the namespace. These concepts are not to be mixed.

### *8.3 Connection and session opening*

In MRML communications, the connection is generally defined when opening sessions (see 8.8 for specific cases). Thus, by default, there is no need for explicitly defining the connection opening. However, it may be required to do so, so that we address this issue in more detail at the end of this section.

With the knowledge of the server’s IP address and port, the client requests the opening (or re-opening) of a session. This is done using the mrml:open-session tag. The client declares itself by the mrml:client tag. As specified earlier, it provides a base-url attribute where a complete information file description.xml may be found. Within the MRML message, only an ID, a suggested name is given. A type attribute is used to indicate useful information about categorizing interfaces. We suggest that the type attribute should follow the idea of the “language” attribute of the HTML script tag (following recommendation RFC 3066).

```
<open-session username="..." session-id="..." >
 <client id="http://viper.unige.ch/demo/php/index.php"
        name="Viper PHP demo interface"
        base-url="http://viper.unige.ch/demo/php"
        type="php"/>
```

```
</open-session>
```

Unless an error is encountered (see section 8.7, next), the server then respond by the mrml:session tag, indicating the ID of the session. This is either the one requested by the client if available at the server side or a new one. This allows for some basic form of profiling. The construction of session IDs is left to the designers but we suggest that it should follow some strict principles to ensure uniqueness of these IDs. A suggestion is simply to use the ID of the server appended with the date of the session opening.
Similarly to the session request process, the server identifies itself using the mrml:server tag. Within the server description, a number of its attributes should be present. These are essentially algorithms and collections accessible from the server and are declared as given in the server configuration file (see section 8.5, next).

```
<session username="..." session-id="...">
 <server id="viper.unige.ch:12790:gift:0.0.8"
        name="GNU Image Finding Tool V0.0.8"
        base-url="http://www.gnu.org/software/gift">
   <!-- (section 8.5)-->
 </server>
</session>
```

*8.4 Query exchange*

At this stage, the server and client have exchanged sufficient information to start a query process. A query will therefore be initiated via an MRML message containing something like:

```
<query id="..."
      type="explicit"
      result-size=".."
      result-cutfoff="...">
 <algorithm id="...">
  <xforms>
   ...
  </xforms >
 </algorithm>
 <collection id=".."/>
   ...
</query>
```

We suggest that each query should have an ID for further reference. For example, the ID should be constructed from the extension mechanism using the session ID completed with the query number. The result-size and result-cutoff attributes are there to limit the size of the result set. Both are optional and the precedence rule for these attributes is to be defined by the algorithm. The xforms excerpt is meant to contain the values of the algorithm parameters via the xforms:instance mechanism.
In the case of a complex query, a parent query tag with its type attribute set to “complex” will wrap specific query elements.

In our context, whatever the query type is, the result of a query is a list of elements identified by a reachable URL and possibly a value that indicates its closeness to the query. We will therefore return the results in a syntax close to the following:

```
 <result query-id="...">
 <result-element location="..." alt="..." similarity="..."/>
  …
</result>
```

Variations will come from the fact that result elements may be better described using a structure that is more complex than the location attribute. In this case, attributes may be added to this tag. The similarity attribute is also optional and may be replaced. In any case, the structure should then be described in the algorithm description file.

Alternatively, the result-element may be open and new descriptive tags may be inserted within it. Here again, the local structure of the tag should be described in the algorithm description file.

*Explicit queries*

This type of queries is identified by their type attribute set to "explicit". Typically, the MRML query tag then becomes a container for any type of data description such as regular expressions (Perl-like) or SQL-like query language (SQL, XMLQL, XQL, Quilt, …).
The idea is that the content of the tag is shipped directly to the algorithm and it is left to the implementation of the algorithm to understand it. Here, the use of the description file in the base-url of the algorithm (provided with examples of queries) is crucial.
Examples: SQL, Perl
Note: connection to an SQL server???

*QBE queries*

This type of queries is identified by the type attribute set to "qbe". This type is defined here since QBE is one of the common type of queries for multimedia retrieval. In the MRML context, a Query-by-Example is abstracted by a list of example items, associated with a relevance judgement.
Within the query tag, we will therefore list a number of query-elements. In the most simple setup, query-elements will be identified by reachable URLs indicated in their location attribute. Similarly, they will be given a relevance value in a relevance attribute. A number of specific attributes may be added to this syntax. Their structure should then be described in the algorithm description file.
Now, it may be the case that the QBE context is a bit more generic. In that case, an alternative mechanism is defined where either or both location and relevance attributes are replaced by location and relevance elements. Similarly, any other element can be added to this syntax.

This first offers the possibility of factoring QBE query. If only the relevance attribute is present in the query-element tag, it will apply to all subsequent location child elements.
Further, in a complex setup, the fact of describing the query item at the element level gives the opportunity to attach a number of other items to this item. For example, if the query item is a video sequence, one may like to attach to the query a set of sequence key frames.

*Misc queries*

Since it is not possible to help the specification of generic queries, we provide a container for any type of queries. This simply goes down to setting the type attribute of the query element to "misc" and insert any query excerpt as a child of this element. The structure of the query should then be specified in the algorithm description file. Clearly, the above specific types of queries are compatible with this syntax.

## *8.5 Server configuration*

Following the above analysis and similarly to the previous specifications of MRML, essential features are processing algorithms and multimedia collections. They are both given into lists where each is identified by  a unique ID. Again this ID should reflect the origin or type of the algorithm or collection. Then, a correspondence is to be created with the collections. A collection should be accessible via an algorithm if and only if it has been indexed in relation to this search algorithm.

The MRML server configuration file should be a consistent MRML document, therefore wrapped into the MRML envelope (see sections 8.1 and 8.2). It is aimed at describing capabilities of a server. The server itself will be declared using the mrml:server tag, as already shown in section 8.3.

```
<server id="viper.unige.ch:12790:gift:0.0.8"
```

```
        name="GNU Image Finding Tool V0.0.8"
        base-url="http://www.gnu.org/software/gift">
 <algorithm-list>
   ...
  </algorithm-list>
  <collection-list>
   ...
  </collection-list>
 </server>
```

We start by the description of algorithms. They are gathered under the umbrella of an algorithm-list that can be used for reference of their usage. Typically, within this list, an algorithm is defined once and the XML reference mechanism should be used to refer to this algorithm into another –algorithm-list (ie using the syntax <algorithm ref=”c-cbir”/>).

```
  <algorithm-list id="cbir-algos">
   <algorithm id="c-idf"
            name="Classical IDF"
            base="http://viper.unige.ch/tf-idf"
            type="cbir/qbe/rf"
            xmlns:viper="http://viper.unige.ch">
  <xforms:model>
   <xforms:instance>
   <viper:qbe>
    <viper:colorhist xsi:type="boolean">1</viper:colorhist>
    <viper:percentfeature></viper:percentfeature>
   <viper:qbe>
  <xforms:instance>
 </xforms:model>
 <xforms:bind nodeset="/viper:qbe/viper:percentfeature" type="int"/>
 <mrml:ui>
   <select1 ref="colorhist">
    <label>Use color histogram</label>
    <item>
     <label>Yes</label>
     <value>1</value>
    </item>
    <item>
     <label>No</label>
     <value>0</value>
    </item>
   </select1>
   <select1 ref="percentfeature">
    <label>Percentage feature</label>
    ...
   </select1>
  </mrml:ui>
  <mrml:interaction>
    <!-- (see section 8.8)-->
  </mrml:interaction>
   </algorithm>
  </algorithm-list>
```

As discussed earlier, we replace former property sheets by the usage of Xforms. Not only this allows to characterise algorithm parameters, as did the property sheets but this also brings a number of advantages:

- This is a complete done recommendation. There is already associated implementation. This does not only provides libraries but also validates concepts. Further, maintenance and evolution is then left to the (active) XForms community
- Values can be associated with types in a standard manner using the XForms data model separation. The complete XML typing mechanism is inherited. Similarly, default values can be suggested (imposed) via the xforms:instance mechanism.
- Values can be associated together and constrained (see the XForms recommendation for details)
- An typical user interface (augmented with XEvent handling) may be suggested using the classical XForms element specification.

We wish to add in this description of the algorithm the definition of user interaction that is associated with the algorithm. We propose this as an extension in section 8.8.

The collections are then described in terms of their properties and association with algorithms. Collection properties are its ID, name, size, type of items.

```
<collection-list id="image-collections">
  <collection id="corel-image"
            name="Corel Image Collection"
            base-url="http://viper.unige.ch/images/corel/"
            size="5000"
            type="image/jpeg"
            algorithm ref="cbir-algos"/>
 …
</collection-list>
```

The list of attributes given above is the minimum set of attributes that can be associated with a collection.

- The type attribute should be as representative as possible and in the worst case be "complex" for the case of mixed collections of items (see also section 8.8)
- We leave the algorithms that can access the collection as an attribute although that implies a unique definition. We wish by this mechanism to encourage the grouping of algorithms in algorithm-list with no duplication using the ref attribute

### *8.6 Server administration*

In this new specification of MRML, we propose that server administration should be possible via the exchange of MRML messages.

The basis of server administration will go through the command tag that will contain predefined commands

```
<command id="…"/>
```

### *8.7 Errors*

Similarly to server administration, error messages will be exchanged via a unique error tag, referring to predefined parameters.

```
<error type="…"
        level="…">
   <message lang="en">Error message in English</message>
</error>
```

## 9. MRML extensions

The use of the mrml namespace allows for defining an extension mechanism. We foresee extensions at different places within this specification.

## 10. Examples

In order to make this specification more complete and also to evaluate its generality, we study some examples that are drawn from experience.

### *10.1 A session with the Viper plugin of the GIFT server*

This is the most study example using the previous MRML version. This example is also the occasion  to express the extent of changes between this proposed version and the current version.

# Annexes

## Annex 1: How about MRML version 1.0 (changes involved)?

We call MRML version 1.0 the version that is currently implemented (*eg* in the GIFT package and in kmrml). Version 0 correspond to the early draft that were published first when developing MRML (this version has no – and should not  have – any implementation support). As mentioned several times in this document, version 1.0 is implicit since no version attribute was planned.
The version handling mechanism proposed here to evolve from version 2.0 onwards allows for ensuring backward compatibility. This is necessary, in order to preserve implementation efforts. However, this should not lead to the unnecessary maintenance of legacy code at a later stage.
We propose here at least two mechanism to realise version handling in practice. Before going into this part, we summarise what changes are essentially involved.

### *A1.1 The MRML 2.0 revolution*

The proposals made within this document could be felt as cosmetic and that they bring little changes to the current MRML version. We believe that this document brings original features to MRML that are necessary for a possible wider development. In this section, we summarise what is effectively cosmetic and what deeper modifications are involved.

- Inclusion of MRML namespace: This is necessary for a proper definition of MRML extensions and the use (re-use of existing standard such as XForms)
- Inclusion of a version handling mechanism: this is a crucial step for a clean and well-defined evolution of MRML. This will enable the use and re-use of MRML software without creating the need for the maintainance of legacy code, as proposed in this Annex.
- Use of XForms in place of the Property Sheets: while Property Sheets (as defined in MRML 1.0) are a nice attempt to abstract algorithm parameters, we think it is an unnecessary specification and that it is better to include XForms in for the same usage. XForms are already a specification from W3C, they have a wider scope that prove useful in the MRML context and already have associated software in a number of languages. By using XForms one readily inherits from all their properties (see section 8.5)
- Better abstraction of the multimedia document: MRML 1.0 is clearly biased toward content-based image retrieval using QBE and supporting relevance feedback (as is implemented in the *Viper* GIFT plugin). There is a strong necessity for emancipating from the image media type while keeping the simplicity of MRML. This does not only boil down to renaming few MRML tags, it also involves a reshaping of the document handling to better exploit XML properties and not be limited.
- Better abstraction of the search method involved: this complements the above. What is done here is essentially to move QBE as a common extension of MRML, thus allowing for the easy definition of other search paradigms.
- Better extension mechanism: this is supported by the use of the mrml namespace and will be extensively tested while using MRML in several contexts. Throughout this specification, we have tried to have a consistent mechanism for extending several aspects of MRML.
- Better fit with other XML protocols: using generic specification practices is essential for ensuring an easy reading and understanding of the MRML specification to these

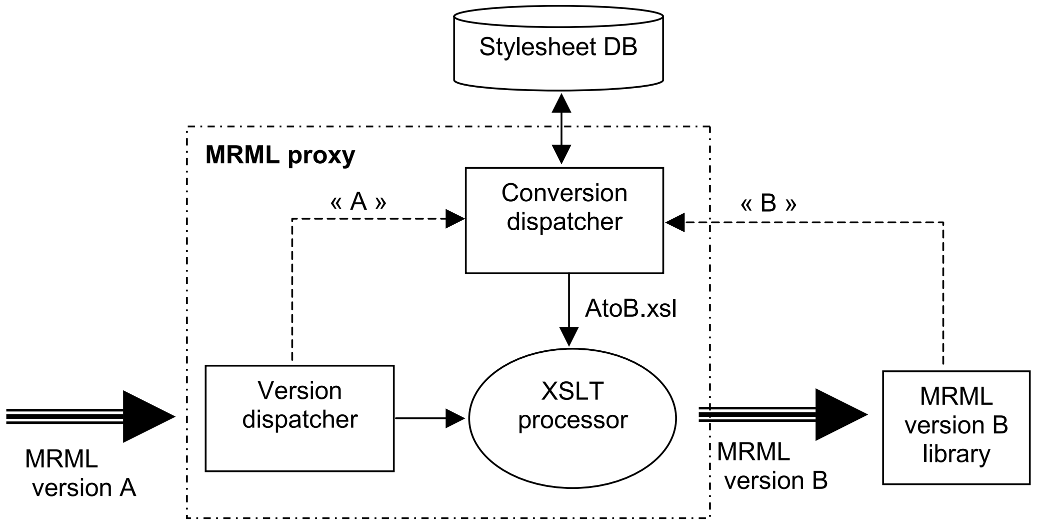
who are used to read specifications. This further highlights inconsistencies within the specification and also facilitate the inclusion of existing standards.

*A1.2 Version conversion*

(Essentially proposed by Wolfgang Müller)
Here, we propose to use a proxy that simply converts MRML messages to the appropriate implementation. Since any MRML message is an XML document, this may be easily done via an XSLT stylesheet. We could then have a number of such stylesheets loaded and used online by a proxy process, in order to ensure that the software receives the appropriate version.
This may be the simplest solution but it presents few shortcomings:

- The conversion between MRML version may not be straightforward or even possible since features are added, reshaped or simply removed.
- This will not encourage the shift towards higher versions.
- It may not well separate version evolution and therefore lead to confusion.
- A number of inter-version conversion stylesheets need to be designed anytime a version evolves.

The figure below shows the structure of an MRML conversion proxy that would assume the responsibility of feeding the MRML-compliant component with messages compliant with a proper MRML version.



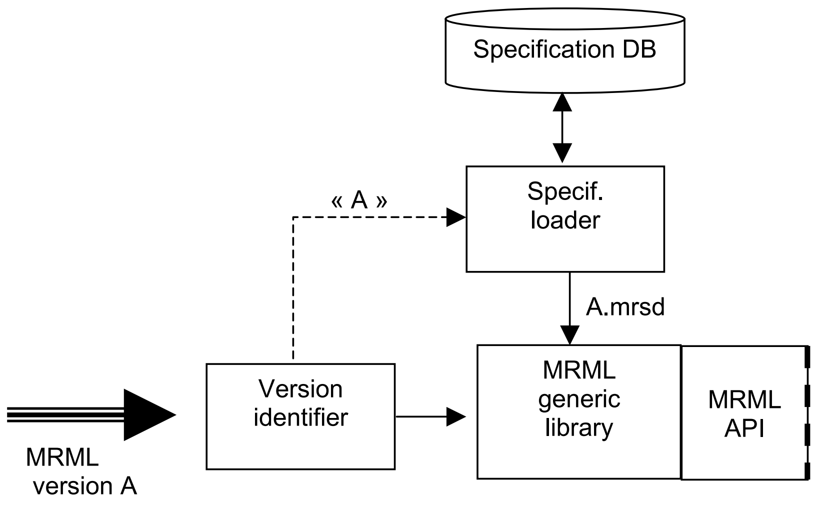
*A1.3 Version control*

We propose here an alternative to the above version conversion system. The idea is to detect the version prior to taking any action. Once the version of the incoming MRML message is defined, the associated MRML specification is loaded under the form of a augmented MRML Schema (X.mrsd). This will constraint both the syntax (well-formedness), the structure (validity) and define the association between MRML elements and actions to be triggered.
This version control system would show a number of advantages over the previous conversion system.

- It allows flexible evolution and testing of several MRML versions while minimising maintenance efforts. New features should be mapped into novel implementation and new syntaxes should only be reflected in the augmented schema of the evolving version (ie no cross-impact on the other versions).
- It forces the understanding of the MRML definition at a functional level (ie definition of a MRML API/Interface). This is required when augmenting the MRML schema. The benefit is a clear understanding of the MRML structure at a functional level that will most likely help in bettering the design.

Identified shortcomings of this approach are as follows:

- It creates a rigid relationship between MRML element and the implementation. This is characterised by the fact that the MRML API may have difficulty to evolve.



## Annex 2: Why MRML and not …?

Below is a list of good reasons for using MRML and not … (something else) in the context of Multimedia Search and Retrieval. If you feel that these reasons are non-sense, we would be extremely happy to have your views.

### *A2.1 Why MRML and not SQL?*

As stated above MRML is not a query language and SQL is one. This alone answers the question. The aims are different although both could achieve the same goal. Rather, the handling of SQL requests is planned into MRML.

### *A2.2 Why MRML and not SOAP?*

MRML could obviously be seen as a specialisation of SOAP. However, it is different enough that you would have to rewrite MRML in SOAP. So why not directly MRML? SOAP is nevertheless a good guide for clean recommendation.
Further, SOAP could be used as a container for MRML messages.

### *A2.3 Why MRML and not XML-RPC?*

XML-RPC being an embryonic version of SOAP (or, rather, SOAP being an overwhelming version of XML-RPC – according to its authors), the above readily argument applies to XML-RPC.

### *A2.4 Why MRML and not HTTP?*

MRML and HTTP are obviously two different things and cannot be compared or taken one for another. The right question is rather: “why MRML does not go through the HTTP port (80)?” This is a good question that need further thinking. Reasons that can be sated are:

- Why would it? What advantages would that bring. Obviously then MRML would be Firewall immune (like SOAP) but is that really needed (and possible)?
- If it would, what would be the implication of that. Then the MRML compliant client or server would have to listen to the right port and be in competition with httpd or equivalent. In that case, the client could be a servlet or equivalent (eg Cocoon XSP). As said above, this all need further thinking…

*A2.5 Why MRML and not Z39.50?*

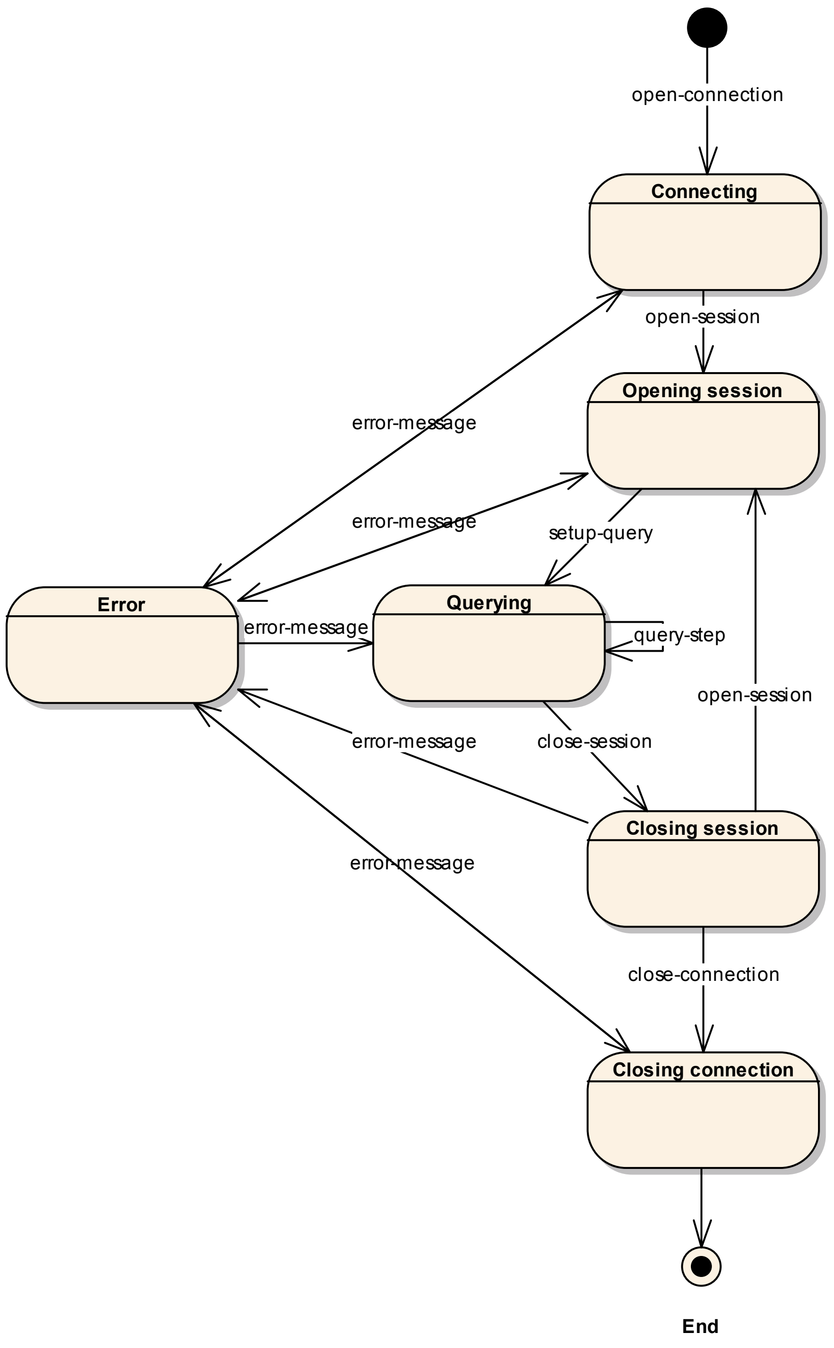
To keep things simple.
I am not an Z39.50-expert and would by no means criticize it. Simply, here are few points that motivated us to define MRML:

- Z39.50 is mostly defined for databases supporting “explicit queries”, as defined in this document. MRML wishes to be wider;
- Z39.50 is a form of binary exchange protocol. It makes sense to move onto XML;
- Reading this document is meant to be simpler than getting the whole bunch of docs accompanying Z39.50

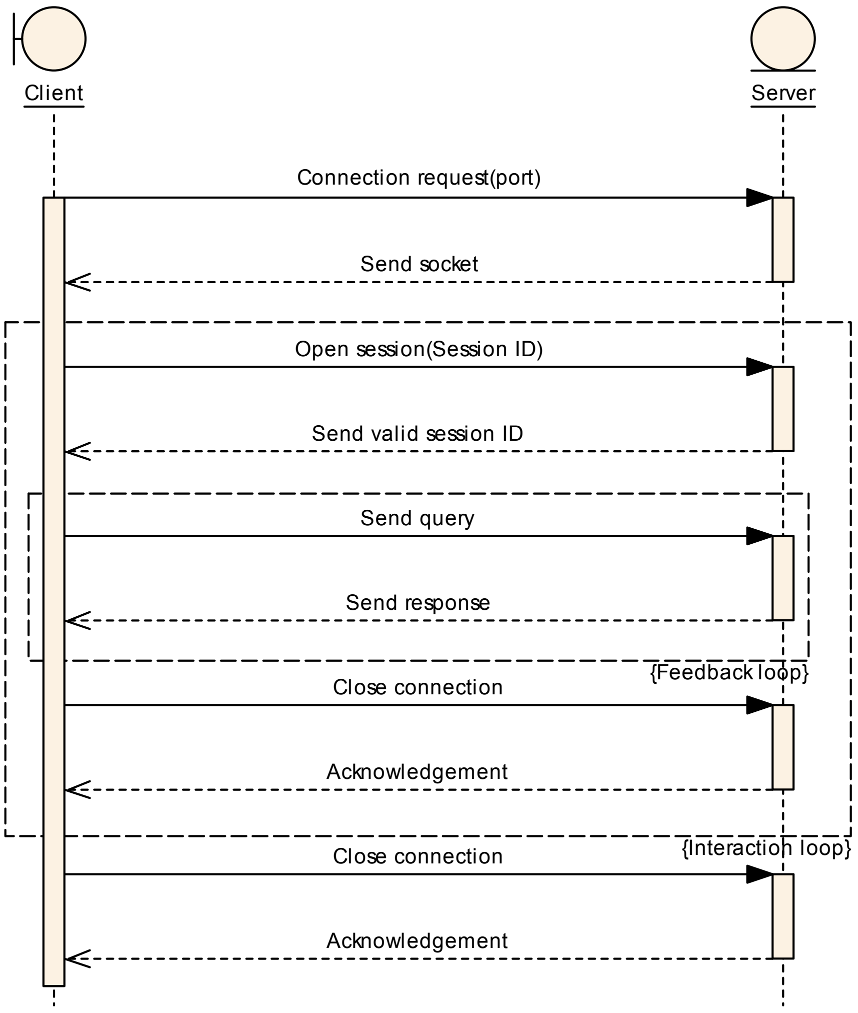
In any cases, we recognise that Z39.50 contains a number of good useful features and thus forms definitely a source of inspiration.

## Annex 3 : Formal MRML analysis

*A3.1 MRML State diagram*



*A3.2 MRML Messaging*


Client
Server
Connection request(port)
Send socket
Open session(Session ID)
Send valid session ID
Send query
Send response
{Feedback loop}
Close connection
Acknowledgement
{Interaction loop}
Close connection
Acknowledgement