



## **I-MAESTRO: Interactive Multimedia Environment for Technology Enhanced Music Education and Creative Collaborative Composition and Performance**

[www.i-maestro.org](http://www.i-maestro.org), [www.i-maestro.net](http://www.i-maestro.net)

### **DE5.1.1**

## **Model and Support for Cooperative Work and SMR for Music Education**

**Version:** 2.9

**Date:** 14/11/2006

**Responsible:** DSI (Francesco Frosini, Giovanni Liguori, Nicola Mitolo, Paolo Nesi)

Project Number: 026883  
Project Title: I-MAESTRO  
Deliverable Type: Public  
Visible to User Groups: Yes  
Visible to Affiliated: Yes  
Visible to Public: Yes

Deliverable Number: DE5.1.1  
Contractual Date of Delivery: M12  
Actual Date of Delivery: 14 November 2006  
Work-Package contributing to the Deliverable: WP1, WP2, WP3, WP4, WP5, WP10  
Task contributing to the Deliverable: WP5  
Nature of the Deliverable: Report  
Author(s): DSI, IRCAM, UNIVLEEDS

#### **Abstract:**

This document reports the specification of the Model and Support for Cooperative Work and Symbolic Music Representation (SMR) for music education.

#### **Keyword List:**

MPEG-SMR, Cooperative Work, Music Editor Support, MAX/MSP jitter, requirements, use cases, test cases, content, interactive multimedia, education, music, creative, collaborative

# Table of Contents

<b>1</b>	<b>EXECUTIVE SUMMARY AND REPORT SCOPE .....</b>	<b>4</b>
<b>2</b>	<b>GENERAL OVERVIEW OF I-MAESTRO ARCHITECTURE .....</b>	<b>4</b>
2.1	I-MAESTRO CLIENT GENERAL OVERVIEW .....	5
<b>3</b>	<b>INTEGRATED MUSIC SCORE EDITOR AND VIEWER TOOL.....</b>	<b>7</b>
3.1	MPEG-4 MODEL .....	8
3.2	MPEG SMR AND MULTIMEDIA .....	9
<b>4</b>	<b>GENERAL ASPECTS OF COOPERATIVE SUPPORT FOR MUSIC TRAINING .....</b>	<b>10</b>
<b>5</b>	<b>COMPUTER SUPPORTED COOPERATIVE WORK SERVICE.....</b>	<b>13</b>
5.1	CLASS DIAGRAM OF COMPUTER SUPPORTED COOPERATIVE WORK SERVICE .....	14
5.2	CSCW SERVICE .....	14
5.3	MESSAGE LOG SERVICE .....	17
5.4	ERROR LOG SERVICE.....	18
<b>6</b>	<b>P2P SERVICE.....</b>	<b>20</b>
6.1	CLASS DIAGRAM OF P2P SERVICE.....	20
6.2	SEND MESSAGE SERVICE .....	21
6.3	RECEIVE MESSAGE SERVICE .....	22
6.4	SEND FILE SERVICE .....	23
6.5	RECEIVE FILE SERVICE.....	24
6.6	DISCOVERY SERVICE.....	24
6.7	SYNCHRONISATION SERVICE.....	26
<b>7</b>	<b>CLIENT MANAGER: STARTING LESSONS/WORKGROUPS.....</b>	<b>27</b>
7.1	CLASS DIAGRAM OF CLIENTMANAGER AND ITS SERVICES .....	32
7.2	DISTRIBUTE LESSON SERVICE .....	33
7.3	COOPERATIVE SESSION DATA .....	34
<b>8</b>	<b>CLIENT MANAGER: MONITORING AND CONTROLLING COOPERATIVE WORK .....</b>	<b>36</b>
8.1	VIEWING AND CHECKING THE LOG MESSAGES .....	36
8.2	VIEWING AND CHECKING THE ERROR MESSAGES .....	37
<b>9</b>	<b>COOPERATIVE MUSIC EDITOR FOR SMR.....</b>	<b>38</b>
9.1	STAND ALONE SMR MUSIC EDITOR .....	38
9.1.1	List of available functionalities.....	39
9.2	STAND ALONE SMR MUSIC PLAYERS MPEG.....	42
9.2.1	List of available functionalities.....	42
9.3	COOPERATIVE MUSIC EDITOR FOR SMR.....	43
9.4	SMR MUSIC PLAYER INTO MPEG4 .....	43
9.4.1	Integration of SMR in the MPEG-4 player .....	43
9.4.2	Authoring an MPEG-4 SMR application.....	46
<b>10</b>	<b>MAX AND THE MUSIC EDITING SERVICE (MED) AND THE MUSIC EXECUTION SERVICE (MEX) 49</b>	
10.1	MAX/MSP DATA TYPES.....	50
10.2	MAX/MSP COOPERATIVE LESSON .....	50
<b>11</b>	<b>MED: SMR MUSIC EDITOR GENERAL CONTROLS FOR MAX.....</b>	<b>51</b>
11.1	INLETS FOR SET METHODS .....	51
11.2	INLETS FOR GET METHODS.....	56
11.3	INLETS FOR OTHER METHODS .....	60
11.4	EXAMPLES OF MED IN MAX/MSP FOR MUSIC GENERAL CONTROLS .....	65

<b>12</b>	<b>MED: SMR MUSIC EDITOR, MUSIC NOTATION ACCESS SUPPORT FOR MAX.....</b>	<b>67</b>
12.1	SCORE NAVIGATION METHODS .....	69
12.2	MEASURE INFO.....	71
12.3	NOTE INFO .....	77
12.4	REST INFO .....	79
12.5	CHORD INFO .....	80
12.6	REFRAIN INFO.....	80
12.7	KEYCHANGE INFO .....	81
12.8	CLEFCHANGE INFO.....	81
12.9	ERROR CODES DESCRIPTION .....	81
12.10	MED IN MAX/MSP FOR MUSIC EDITING .....	82
12.11	EXAMPLES OF MED IN MAX/MSP FOR MUSIC NOTATION ACCESS .....	84
<b>13</b>	<b>MEX: MUSIC EXECUTION SERVICE FOR COOPERATIVE WORK FOR MAX.....</b>	<b>86</b>
13.1	TIME CRITICAL COMMANDS.....	87
13.2	GENERIC MESSAGES.....	88
13.3	CSCW SPECIFIC CONTROLS.....	89
13.4	EXAMPLES OF MEX FOR COOPERATIVE WORK AMONG MAX TOOLS .....	90
<b>14</b>	<b>SCORE-FOLLOWING INTEGRATION MODEL (IRCAM) .....</b>	<b>91</b>
14.1	MUTUAL POSITION CONTROL BETWEEN THE MED AND THE SCORE-FOLLOWER.....	92
14.2	ALIGNED RENDERING OF PERFORMANCE DATA .....	92
<b>15</b>	<b>MODELS FOR NON-SYMBOLIC PERFORMANCE DATA (IRCAM).....</b>	<b>93</b>
<b>16</b>	<b>MODELS FOR COLLABORATIVE INTERACTIVE AUDIO PROCESSING (IRCAM).....</b>	<b>94</b>
16.1	SIMPLIFIED MODULAR INTERACTIVE AUDIO PROCESSING FRAMEWORK.....	94
16.2	SYNTHESIS CONTROL AND SYNCHRONISATION NETWORK PROTOCOL (IRCAM) .....	97
<b>17</b>	<b>ACRONYMS.....</b>	<b>98</b>
<b>18</b>	<b>BIBLIOGRAPHY .....</b>	<b>99</b>

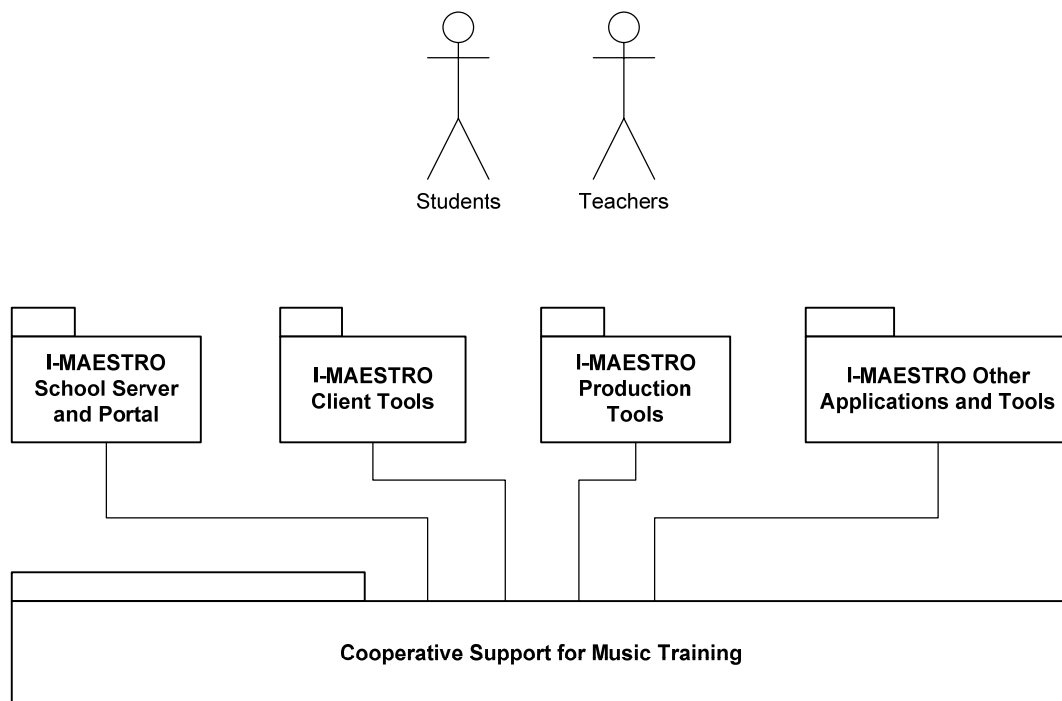
## 1 Executive Summary and Report Scope

This document reports the specification for the first 12 months of the Model and Support for Cooperative Work and SMR for Music Education. It takes information from the WP2 of user requirements and from the preliminary work performed by project partners and summarised in this proposal document. The specification takes into account the general structure of the I-MAESTRO framework.

This document is focused on mapping the research and development work on the real needs provided by the requirements, use cases and test cases.

## 2 General Overview of I-MAESTRO Architecture

In this section, the I-MAESTRO architecture and main components are described. The general architecture of the I-MAESTRO is shown in the following figure:



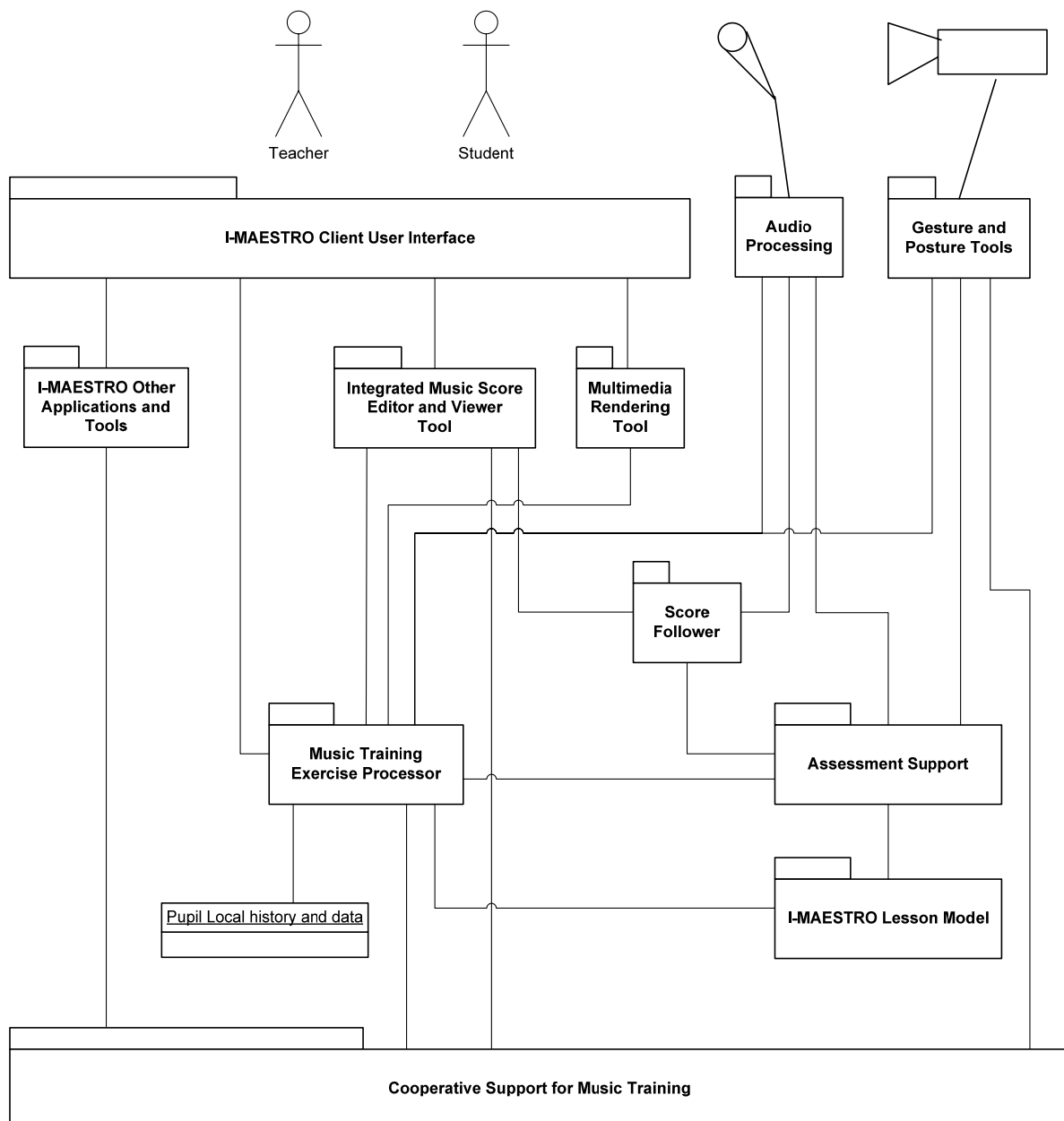
At a high level, the I-MAESTRO architecture is composed of the following components:

- **I-MAESTRO Client Tools** which support Teacher and Students in following Lesson and performing Exercises.
- **I-MAESTRO Production Tools** which are used by Teacher to create Exercises and package Lessons.
- **I-MAESTRO Other Applications and Tools** which include applications such as: a Metronome, a Tuner for instrument tuning, the Audio Recording Tool for audio analysis, the Video Recording Tool and Tools for gesture and posture support, etc.
- **I-MAESTRO School Server and Portal** which contains all available I-MAESTRO contents for the I-MAESTRO tools and all the information about registered users (Students and Teacher profiles). The Portal also provides an external access for registration, and anonymous users for downloading I-MAESTRO tools, manuals, user guides, information etc.
- **Cooperative Support for Music Training** which allows I-MAESTRO Tools to work in a cooperative manner and to exchange information and contents.

The list of I-MAESTRO tools, as described above, can access the cooperative support to exchange messages and files during a cooperative session. The Client Tools are used to configure the lesson chosen for the cooperative session and to manage user roles defined inside the lesson. Other I-MAESTRO Applications and Tools represent instead any Max/MSP lesson with cooperative service which gives the cooperative capability to each tool (video rendering, editor, metronome ...) depending on the lesson structure defined during the authoring phase.

The I-MAESTRO School Server and the I-MAESTRO Production Tools do not directly use the peer-to-peer layer but they are available on the same network for the other tools to provide and to store lessons and user-profiles (e.g. it is possible to search and download a lesson from the School Server using a Web Browser).

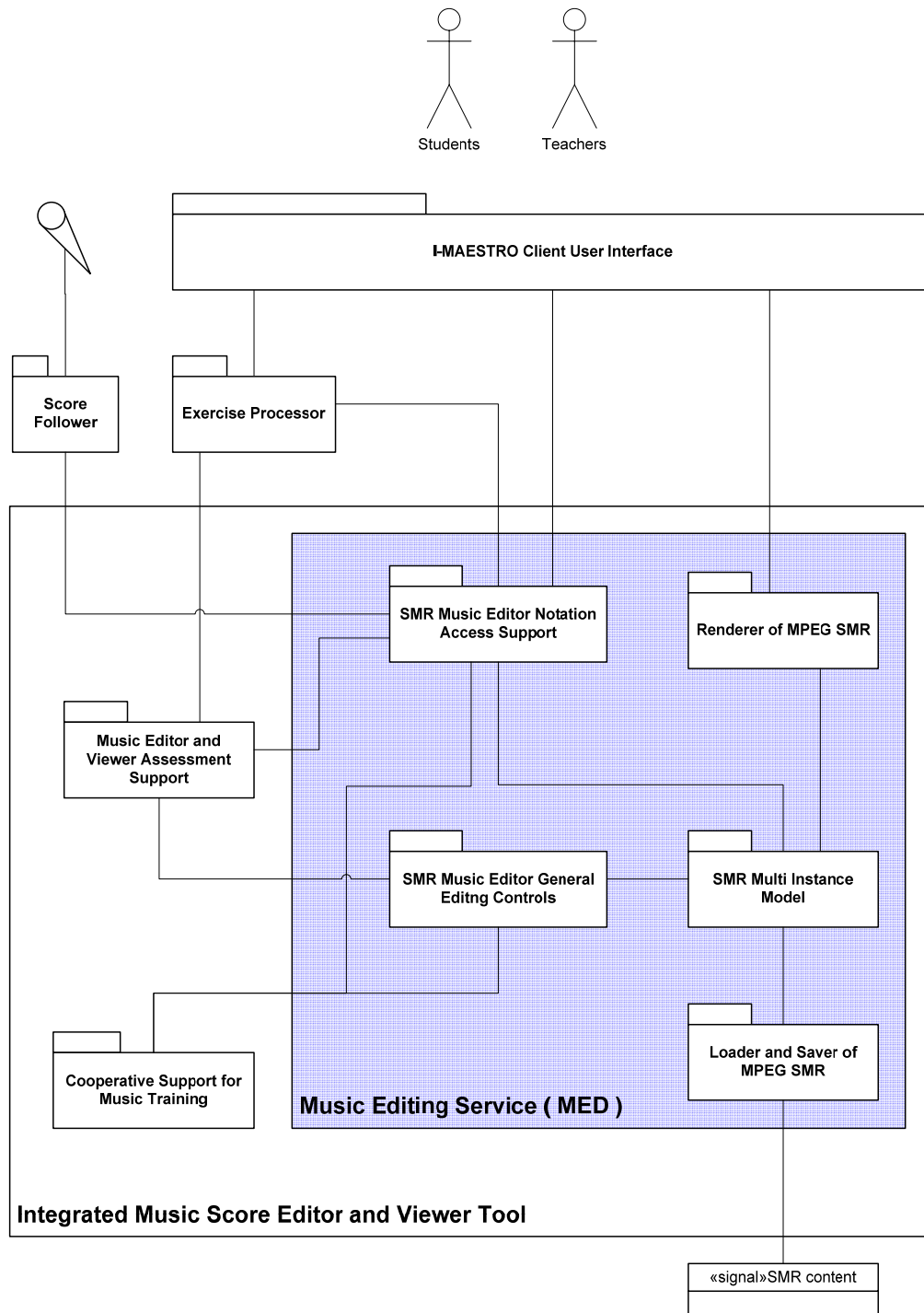
## 2.1 I-MAESTRO Client General Overview



**The I-MAESTRO Client** is the tool used by Teachers and Students to execute Lessons and Exercises. It includes these modules:

- **I-MAESTRO Client User Interface**, used directly by Teacher and Students to interact with I-MAESTRO Applications (e.g. Tuner, Metronome, etc.), the Integrated Music Score Editor and Viewer Tool, the Multimedia Rendering Tool and Music Training Exercise Processor.
- **Music Training Exercise Processor**. This is the core of I-MAESTRO Client and it is able to manage (sending and receiving commands) the other modules of I-MAESTRO Client (i.e. Integrated Music Score Editor and Viewer Tool, Multimedia Rendering Tool, Audio Processing and Gesture and Posture Tools, I-MAESTRO Lesson Model). It also manages the Pupil Local History and data and it can give assessments on performances depending on the evaluations of the Assessment Support.
- **I-MAESTRO Lesson Model**. The model represents the Lesson content and it is used by the Music Training Exercise Processor to follow the correct step during the performance. It also linked to the Assessment Support to assess Student executions.
- **Audio Processing**. It provides the capability to acquire sound to record a performance and/or to assess it.
- **Gesture and Posture Tools**. This is a set of tools to provide support and assess the Students' performance (playing the instrument), using data from sensors and 3D Motion Data capturing system.
- **Score-follower**. This is used in some type of exercises and it gives the capability to upload the score viewer depending on the selected Tempo. Also it can upload in real time the score viewer depending on the speed of the execution. In both cases it is able to collect information from Audio Processing, sends command to the Integrated Music Score Editor and Viewer Tool and passes information to the Assessment Support for performance assessment.
- **Assessment Support**. It collects information from the Lesson Model, Audio Processing and Gesture and Posture Tools and it is able to give a response on some measure of the performance's quality.

### 3 Integrated Music Score Editor and Viewer Tool



The Integrated Music Score Editor and Viewer Tool is a tool developed to edit and execute a score in a cooperative and single user way.

Students and Teachers use the I-MAESTRO client interface to interact with the Integrated Music Score Editor and Viewer Tool.

The Integrated Music Score Editor and Viewer Tool comprise of a set of functionalities that are:

- Renderer of MPEG SMR to update use GUI depending on changes made to the SMR
- Music Editor and Viewer Command Support used by Exercise Processor
- Music Editor and Viewer Command Manager, which collect commands given to manipulate SMR Multi Instance Model
- Music Editor and Viewer Assessment Support, which provides assess support during the execution and editing to highlight Students mistakes.
- SMR Multi Instance Model: the SMR model of the score used.

Also the Integrated Music Score Editor and Viewer Tool can receive command managed by the Music Editor and Viewer Command Manager directly from:

- a user, through I-MAESTRO Client User Interface,
- the Exercise Processor, through the Music Editor and Viewer Command Support ,
- the Cooperative Support for Music Training, during a cooperative session,
- the Score-Follower, during a score execution,
- the SMR Multi Instance Model which represents the score loaded from the Loader and Saver of MPEG SMR.

### 3.1 MPEG-4 Model

In the MPEG-4 model, audio-visual objects have both a spatial and a temporal extent. Temporally, all AV objects have a single dimension. Each AV object has a local coordinate system in which the object has a fixed spatio-temporal location and scale. AV objects are positioned in a scene by specifying one or more coordinate transformations from the object's local coordinate system into a common, global coordinate system, or scene coordinate system.

BIFS is an abbreviation for "BInary Format for Scenes". BIFS provides a complete framework for the presentation engine of MPEG-4 terminals. BIFS enables to mix various MPEG-4 media together with 2D and 3D graphics, handle interactivity, and deal with the local or remote changes of the scene over time. An audio-visual object in a BIFS scene is usually represented by one BIFS node or a sub-tree of the BIFS scene graph.

Scene description information is a property of the scene's structure rather than of particular AV objects. Consequently, it is transmitted as a separate stream. This is an important feature for bitstream editing and one of the essential content based functionalities in MPEG-4. For bitstream editing, one can change the composition of AV objects without having to decode their bitstream and change their content. If the position of the object were part of the object's bitstream, this would become very difficult.

The scene description can be dynamically changed at any time. An initial scene description is provided at the beginning of an MPEG-4 stream. It can be as simple as a single node, or as complex as one wants (within limits that are established for ensuring conformance). BIFS-Commands are used to modify a set of properties of the scene at a given time. It is possible to insert, delete and replace nodes, fields and ROUTEs as well as to replace the entire scene. For continuous changes of the parameters of the scene, BIFS-Anim can be used; it specifically addresses the continuous update of the fields of a particular node. BIFS-Anim is used to integrate different kinds of animation, including the ability to animate face models as well as meshes, 2D and 3D positions, rotations, scale factors, and colour attributes. The BIFS-Anim information is conveyed in its own elementary stream.



## 3.2 MPEG SMR and Multimedia

MPEG-4 permits the encoding of multimedia content, including many different kinds of object types and a scene description allowing precise synchronisation among them and specifying object composition rules.

Symbolic representations of music have a logical structure consisting of: symbolic elements that represent audiovisual events; the relationship between those events; and aspects of rendering those events. There are many symbolic representations of music including different styles of Chant, Renaissance, Classic, Romantic, Jazz, Rock, Pop, and 20<sup>th</sup> Century styles, percussion notation, as well as simplified notations for children, Braille, etc.

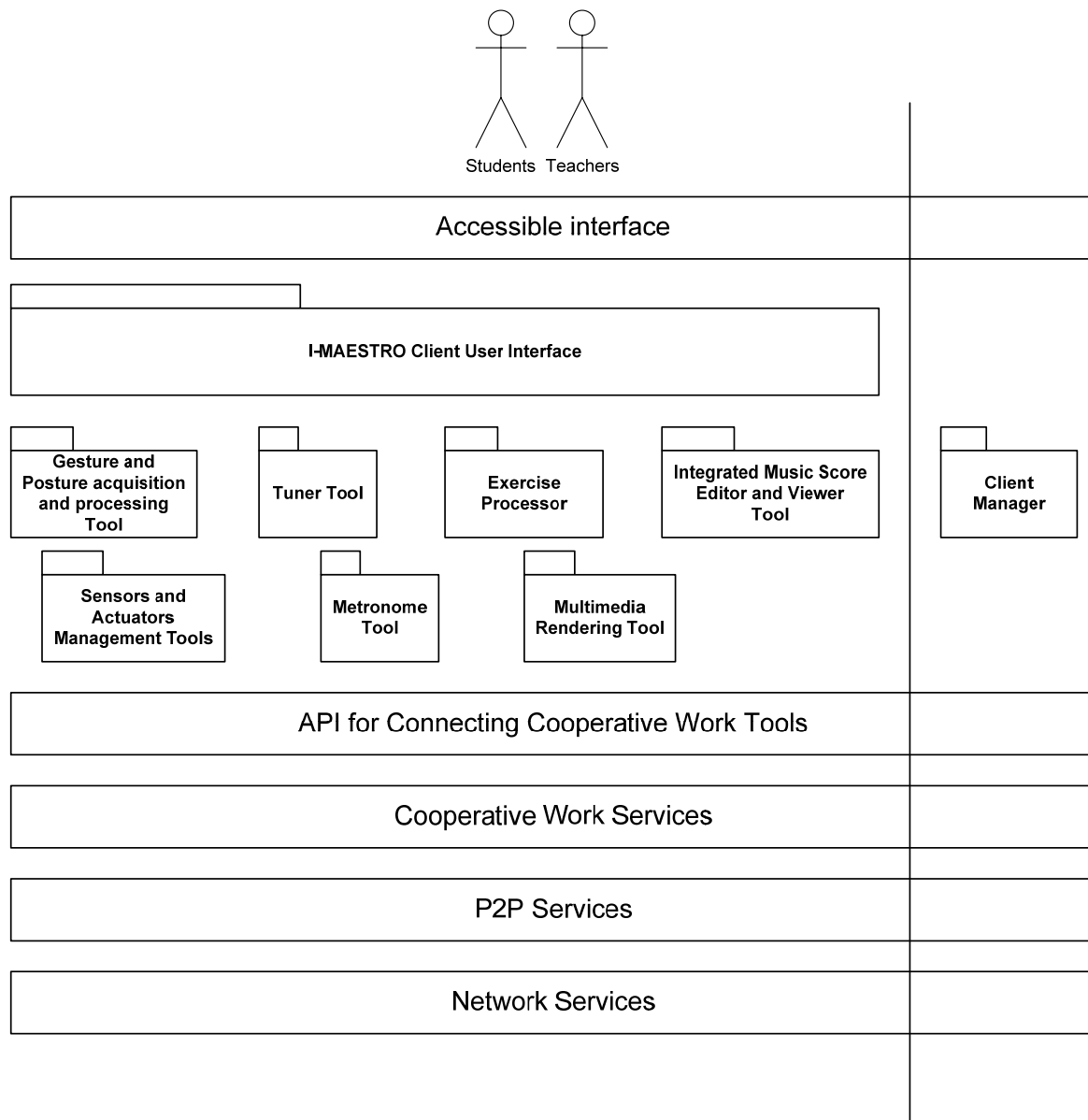
Many music-related software and hardware products are currently available in the market. Some integrate symbolic representations of music with multimedia content. Examples include:

- Interactive music tutorials
- Play training, performance training
- Ear training
- Compositional and theory training
- Multimedia music publication
- Software for music management in libraries (music tools integrating multimedia for navigation and for synchronisation),
- Software for entertainment (mainly synchronisation between sound, text and symbolic information),
- Piano keyboards with symbolic music representation and audiovisual capabilities,
- Mobile devices with music display and editing capabilities.

MPEG Symbolic Music Representation (SMR) enables the synchronisation of symbolic music elements with audio-visual events that are represented and rendered using existing MPEG technology.

The breadth of MPEG standards for multimedia representation, coding, and playback, when integrated with symbolic representations of music provides content interoperability and an efficient high quality, peer reviewed, standardized toolset for developers of these products.

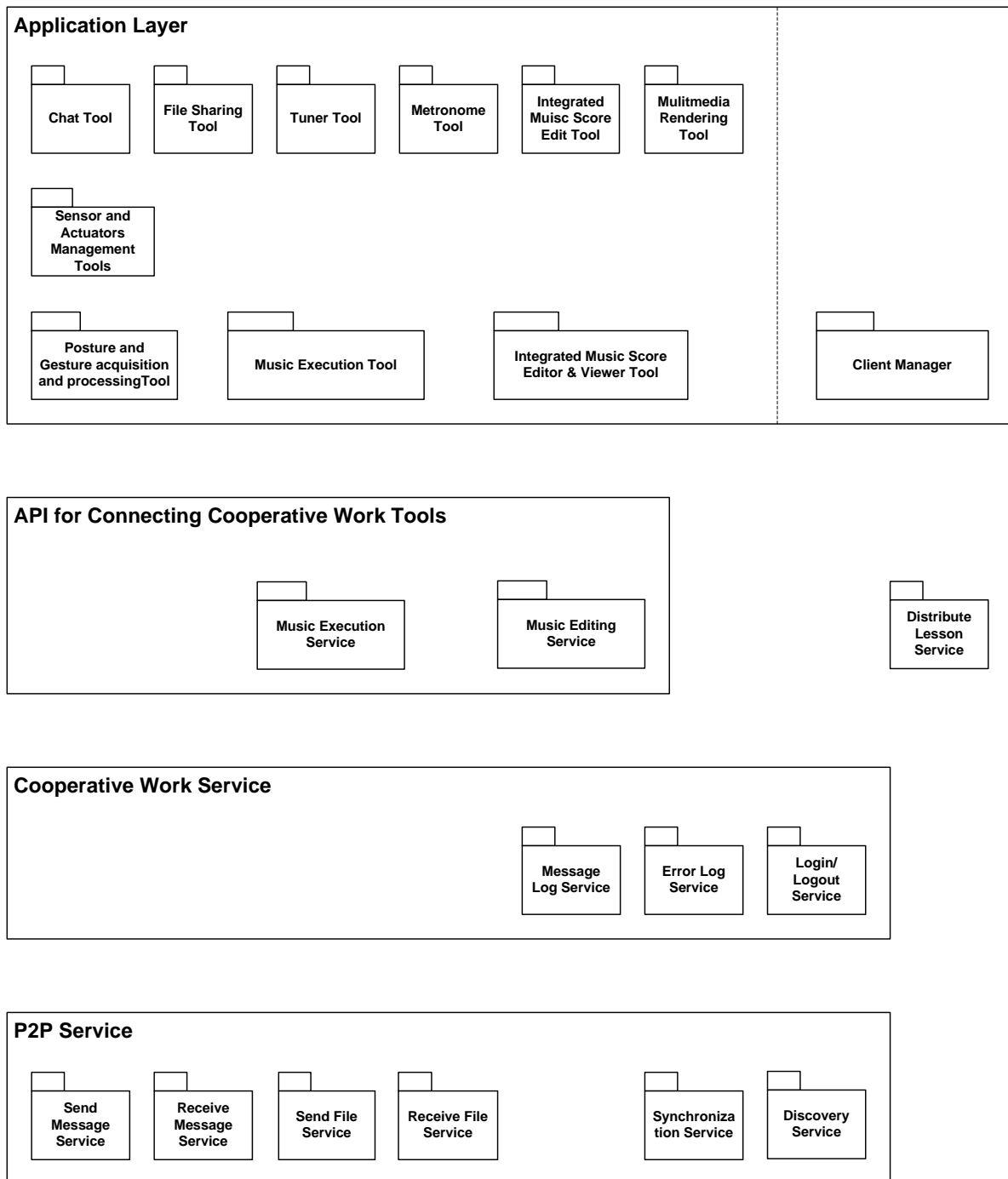
## 4 General Aspects of Cooperative Support for Music Training



Teachers and Students use I-MAESTRO Client User Interface or Accessible Interface (for impaired people) to use tools and applications as the Integrate Music Score Editor and Viewer Tool, the Gesture and Posture Tool, the Multimedia Rendering Tool and so on.

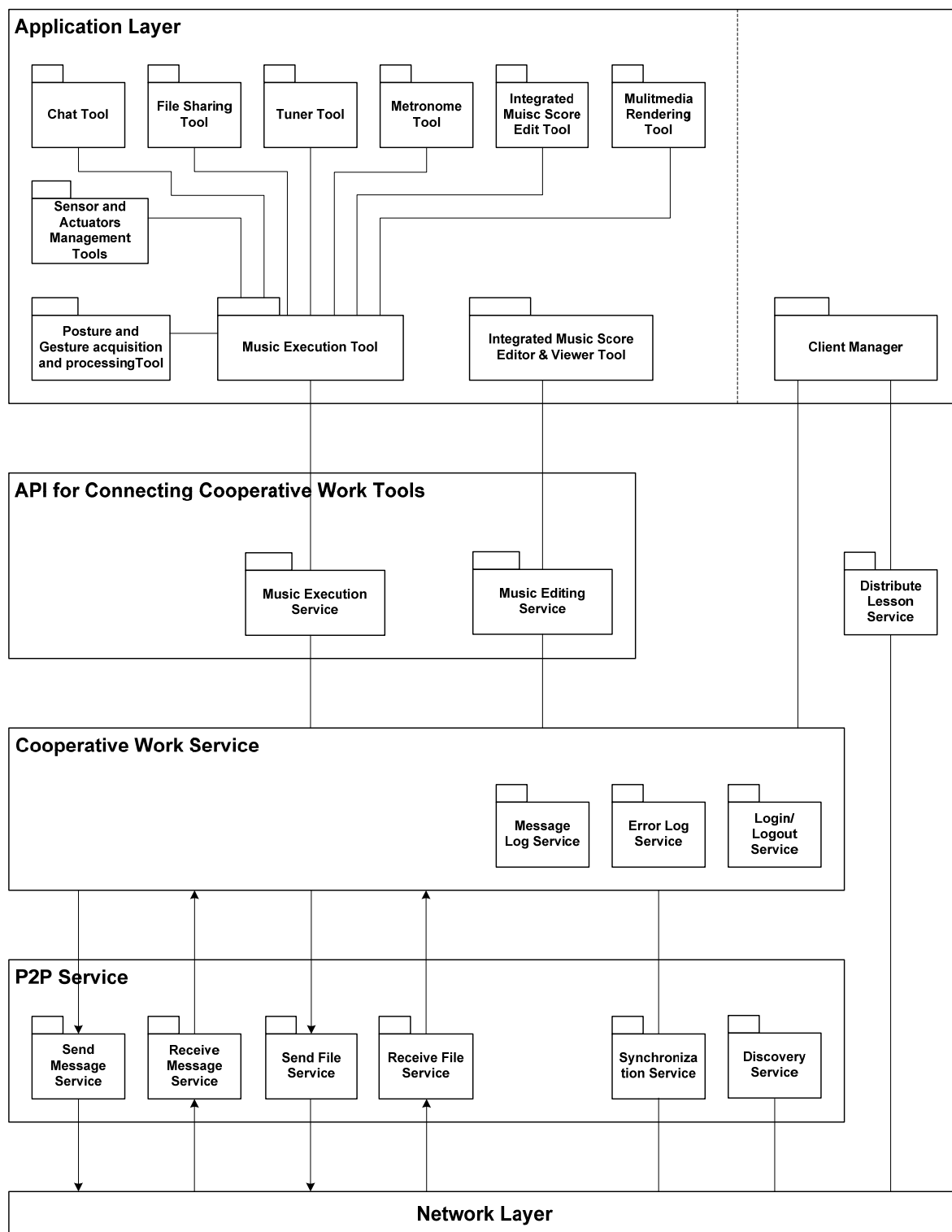
As shown in the figure above, **Cooperative Support for Music Training** contains of three layers:

- **API for Connecting Cooperative Work Tools:** each specific tool, which needs to work cooperatively, has its own API to exploit Cooperative work services and exchange messages and files with each other.
- **Cooperative Work Service:** services of this layer are used from the API to communicate in a distributed system and they create log about message exchanged and error occurred.
- **P2P Services:** manages the low level P2P Network functionalities used by the higher layers. These services are launched when user logs on to P2P Network and it runs independently until the end of cooperative session.



This figure presents a more detailed view of the Cooperative Support layers. The **Application Layer** represents the tools with cooperative functionalities. The **API for Connecting Cooperative Work Tools** is a layer providing the cooperative functionalities which are implemented by the **Cooperative Work Service** and the **P2P Service**.

The following figure shows in detail the logic connection between services.



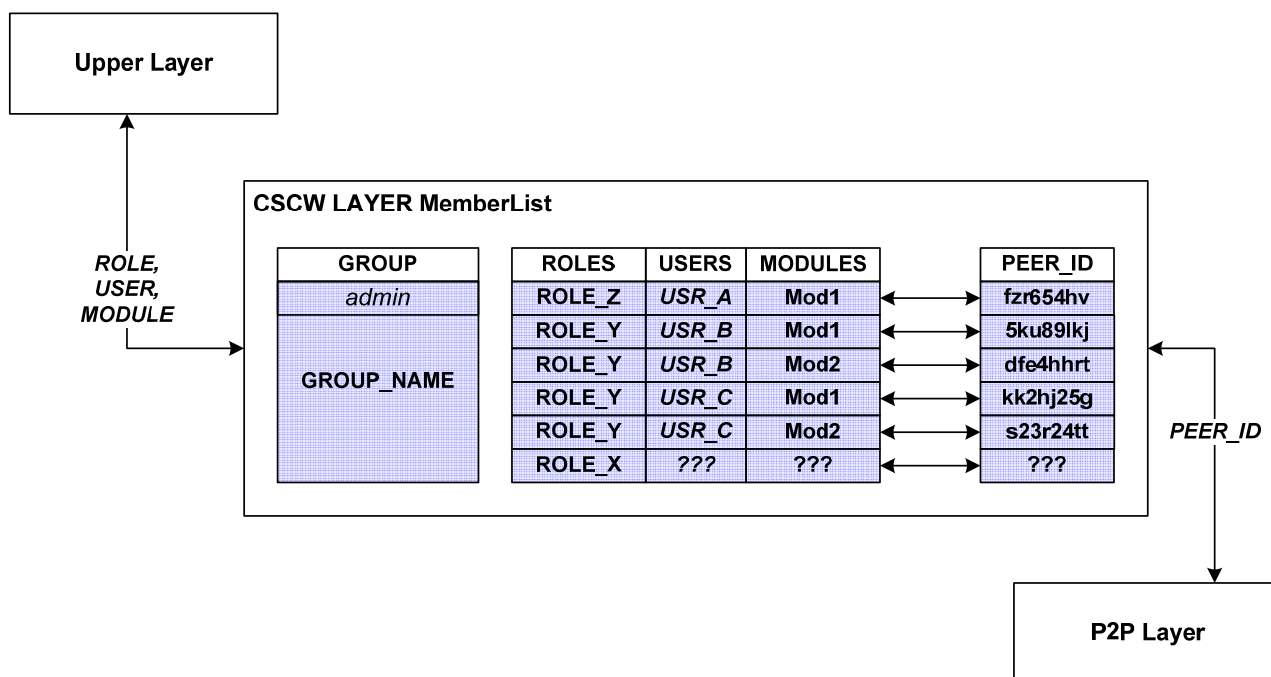
## 5 Computer Supported Cooperative Work Service

Computer Supported Cooperative Work (CSCW) Service keeps information about active work groups inside P2P network and manages the information exchange between peers of a specific work group. Applications that use CSCW Service can join or leave work groups at any time and the information about members are updated automatically whenever a configuration change happens.

At the moment of work group creation, it is possible to specify if the group must have an internal role structure using an array of role names. In that way the work group can contain at most a number of members equal to the size of the array.

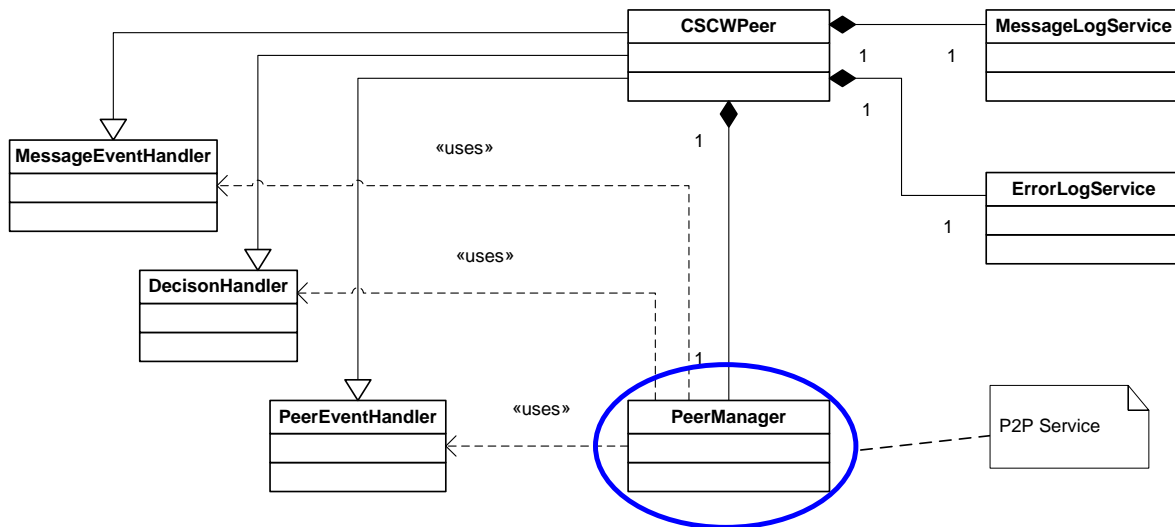
If a new member tries to join the group with a wrong role name or if the role selected is busy, the join procedure fails. Creating a work group without roles specification causes no limit on the maximum number of group members.

CSCW Service uses an internal Member List for routing group information coming from and addressing the upper layer. The following figure shows a generic example of CSCW Member List.



CSCW Service has the capability to support time consistent execution of commands through the network using the P2P synchronisation service that periodically starts a synchronisation procedure. For time critical applications, such as distributed sound playing, the user can also request an immediate synchronisation procedure at any time.

## 5.1 Class Diagram of Computer Supported Cooperative Work Service



CSCW Service involves the following classes:

- **CSCWPeer Class:** it wraps CSCW Service and provides methods for groups management. It uses P2P Services to exchange information between peers.
- **MessageLogService:** registers the information of every message that flows through CSCW Layer
- **ErrorLogService:** registers the information of every error that occurs in the CSCW Layer
- **PeerManager:** a class which links the CSCW work service with P2P service.

CSCWPeer also inherits from **MessageEventHandler**, **DecisionHandler** and **PeerEventHandler** to manage event occurred in the beneath layer.

## 5.2 CSCW Service

login	
Method	login
Description	Starts the services of P2P Layer.
Input parameters	String module : the name of Module chosen (e.g CM, MEX, ...) String user : the User (nick name) chosen
Output parameters	Boolean - TRUE if correctly logged in, FALSE otherwise

logout	
Method	logout
Description	Disconnects the user stopping all the active services of the P2P Layer.
Input parameters	None
Output parameters	Boolean - TRUE if correctly logged out, FALSE otherwise

<b>createGroup</b>	
Method	createGroup
Description	Creates a new work group and optionally defines a set of roles. If roles are present, each member must select a role for joining in the group.
Input parameters	String group : the name of the work group to create String Array roleList (optional): the list of possible roles
Output parameters	Boolean - TRUE if correctly created, FALSE otherwise (e.g. the group already exists)

<b>joinGroup</b>	
Method	joinGroup
Description	Joins in a work group as new member
Input parameters	String group: the name of the work group to create String role (optional): the Role chosen (empty means all roles)
Output parameters	Boolean - TRUE if correctly joined, FALSE otherwise (e.g. the group does not exists)

<b>leaveGroup</b>	
Method	leaveGroup
Description	Removes member from work group
Input parameters	String group : the name of the work group to leave
Output parameters	Boolean - TRUE if correctly removed, FALSE otherwise

<b>getAvailableGroups</b>	
Method	getAvailableGroups
Description	Returns the list of active work groups on the P2P Network
Input parameters	None
Output parameters	String Array groupList : the active work groups list

<b>getAvailableRoles</b>	
Method	getAvailableRoles
Description	Returns the list of available roles of a specific work group
Input parameters	String group : the name of the selected work group
Output parameters	String Array roleList : the available roles list (empty means all roles busy)

<b>sendFile</b>	
Method	sendFile
Description	Uses the send file service of P2P Layer to deliver a file over the P2P Network. The file is sent to every member of the group that matches with role, module and user parameters.
Input parameters	String file : the path of the file to send String role (optional): the Role the file is directed to (empty means all roles) String module (optional): the Module the file is directed to (empty means all modules)

	String user (optional): the User the file is directed to (empty means all users)
Output parameters	Boolean - TRUE if correctly sent, FALSE otherwise

<b>sendCommand</b>	
Method	sendCommand
Description	Uses the send message service of P2P Layer to deliver a command over the P2P Network. The command is sent to every member of the group that matches with role, module and user parameters.
Input parameters	String cmd : command to send String anti_cmd : command that revokes the effect of cmd String role (optional): the Role the command is directed to (empty means all roles) String module (optional): the Module the command is directed to (empty means all modules) String user (optional): the User the command is directed to (empty means all users)
Output parameters	Boolean - TRUE if correctly sent, FALSE otherwise

<b>setCommandHandler</b>	
Method	setCommandHandler
Description	Sets a user defined function that will be called on network command receive.
Input parameters	Function pointer pF, that points to a function with no return values and these arguments <ul style="list-style-type: none"> <li>- String command : the name of the command</li> <li>- int64 timestamp : the time stamped value</li> </ul>
Output parameters	None

<b>setFileReceiveHandler</b>	
Method	setFileReceiveHandler
Description	Sets a user defined function that will be called on network file receive.
Input parameters	Function pointer pF, that points to a function with no return values and this arguments <ul style="list-style-type: none"> <li>- String filePath : the path where the file has been saved</li> </ul>
Output parameters	None

<b>setFileReceivedDir</b>	
Method	setFileReceivedDir
Description	Sets a user defined directory that will be used for saving receive files.
Input parameters	String dir : the path to the specified dir
Output parameters	None

<b>setErrorLogDir</b>	
Method	setErrorLogDir
Description	Sets a user defined directory that will be used for saving error log files.
Input parameters	String dir : the path to the specified dir
Output parameters	None



<b>setMessageLogDir</b>	
Method	setMessageLogDir
Description	Sets a user defined directory that will be used for saving message log files.
Input parameters	String dir : the path to the specified dir
Output parameters	None

<b>doSynchNow</b>	
Method	doSynchNow
Description	Requests an immediate execution of synchronisation round.
Input parameters	None
Output parameters	None

<b>getSyncInterval</b>	
Method	getSyncInterval
Description	Returns the time interval between two consecutive synchronisations
Input parameters	None
Output parameters	Long Interval – time interval in milliseconds

<b>setSyncInterval</b>	
Method	setSyncInterval
Description	Sets the time interval between two consecutive synchronisations
Input parameters	Long Interval – time interval in milliseconds
Output parameters	None

### 5.3 Message Log Service

Message Log Service maintains trace of all information passed through Cooperative Work Service Layer. It is possible to set and get the file name where info will be saved. If no file name is set, the Service creates a new file automatically with a default name with the current hour and date.

An instance of this class is created inside the PeerManger and it is used to log each message passed through the services of the Cooperative Work Service layer. The file is a text one where each line is the log of a message

<b>MessageLogService</b>
+setFilename() +getFilename() +writeMessage()

<b>setFileName</b>	
Method	setFileName
Description	Sets the name of the file used to save messages passed through Cooperative Work
Input parameters	String filename - name of the file where the log are saved
Output parameters	None

<b>getFileName</b>	
Method	getFileName
Description	Gets the name of the file set to save messages
Input parameters	None
Output parameters	String File Name - name of the file where the log are saved

<b>writeMessage</b>	
Method	writeMessage
Description	Adds a line to the log file every time that a message leaves from Send Message Service or arrives to Receive Message Service.
Input parameters	String AddressPeer – ip address of the peer that has sent the message, String service type – service recipient of the message String message – the message sent Timestamp – time and date of the message arrival
Output parameters	Boolean: TRUE if the writing succeeds FALSE otherwise

## 5.4 Error Log Service

Error Log Service maintains trace of all errors occurred in the Cooperative Work Service Layer and in the P2P Service Layer. It is possible to set and get the file name where error messages will be saved. If no file name is set, the Service creates a new file automatically with a default name with the current hour and date. The Error file is a text file where each line is an error occurred inside the services of Cooperative Work Service layer.

An instance of this class is created inside the PeerManager and it is used to log each error message happened inside the services of Cooperative Work Service layer. The file is a text one where each line is the log of a message.

<b>ErrorLogService</b>
+setFilename() +getFilename() +writeMessage()

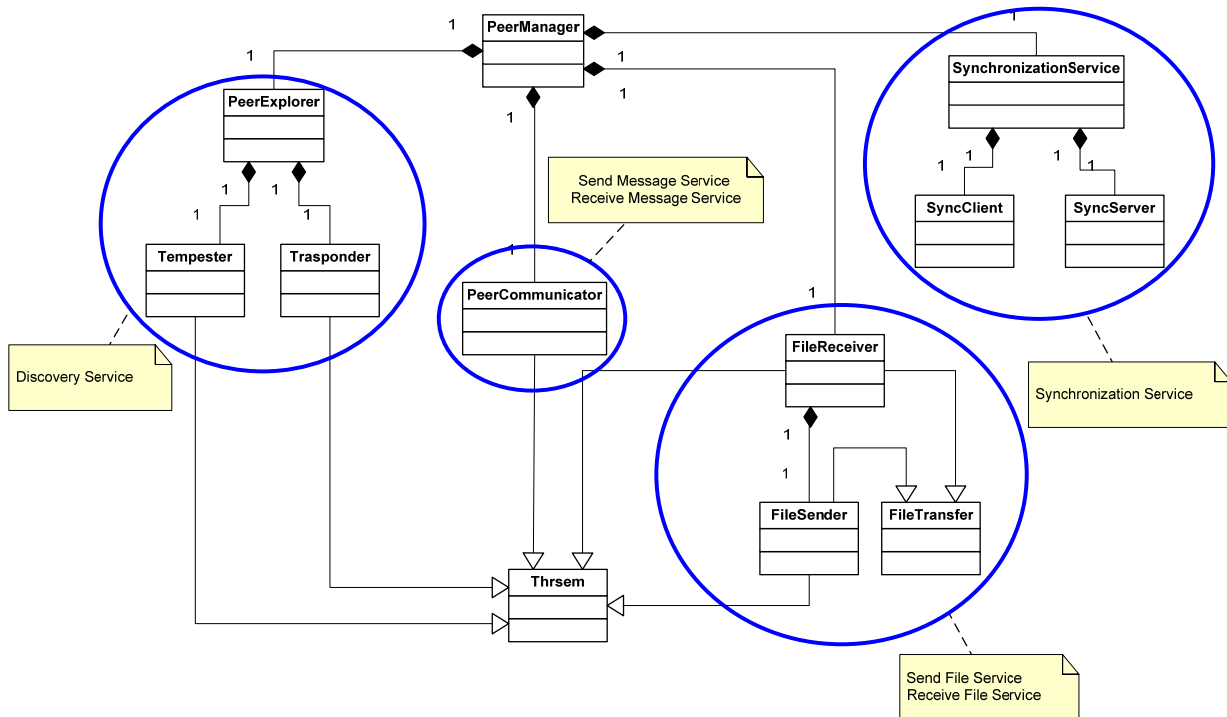
<b>setFileName</b>	
Method	setFileName
Description	Sets the name of the file used to save error happened inside Cooperative Work Service
Input parameters	String filename - name of the file where the log are saved
Output parameters	None

<b>getFileName</b>	
Method	getFileName
Description	Gets the name of the file set to save messages
Input parameters	None
Output parameters	String File Name - name of the file where the log are saved

<b>writeMessage</b>	
Method	writeMessage
Description	Adds a line to the error log file every time that an error happen during the execution of a service inside a Cooperative Work Service.
Input parameters	String service type – service which has generated the error, String error type – type of the error generated, String Timestamp – hour and date of error
Output parameters	None

## 6 P2P Service

### 6.1 Class Diagram of P2P Service

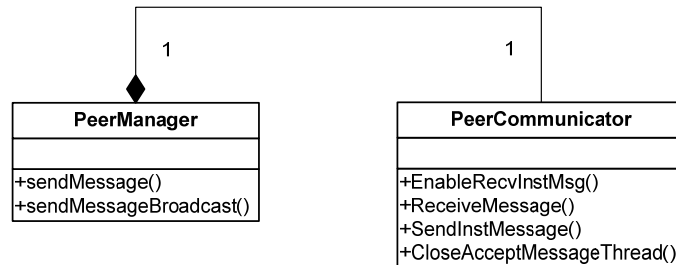


P2P Service involves the following classes:

- **PeerManager Class:** it wraps P2P Service and separates it from the Cooperative Work Layer. It has methods to start and stop services and to exchange information between layers.
- **PeerExplorer:** define the behaviour of the Discovery Service using the classes Transponder and Tempester.
- **PeerCommunicator:** for sending and receiving messages (Send Message Service and Receive Message Service).
- **FileReceiver** and **FileSender:** for file sending and receiving
- **SynchronisationService:** it implements the algorithm and the thread to keep peers synchronised
- **ThrSem:** it provides support to threads implementation

## 6.2 Send Message Service

PeerManager leans on PeerCommunicator Send InstMessage method to provide methods to send message. Here, the methods of PeerManager are described.



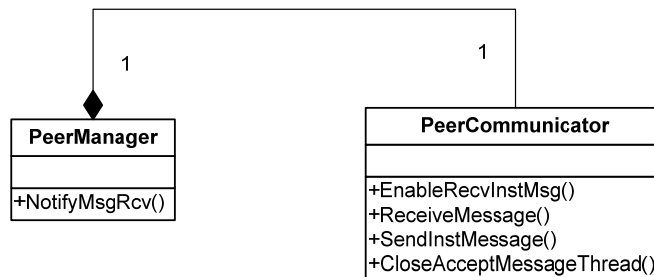
This service takes a message arriving from the API layer and sends it to the specified peer, to all peers in a Workgroup or to all peers in the P2P Network.

When a service wants to send a message to all peers over the network, it uses sendMessageBroadcast, which sends a message to all registered peers in the P2P Network.

sendMessageBroadcast	
Method	sendMessageBroadcast
Description	Method of PeerManager class: It sends a broadcast message to all peer in the P2P Network
Input parameters	String Message – message to deliver
Output parameters	Int 0 if the message is sent correctly, -1 otherwise

sendMessage	
Method	sendMessage
Description	Method of PeerManager class: It sends a message to a specific Recipient
Input parameters	String RecipientPeerID String message – the message to deliver
Output parameters	Int 0 if the message is sent correctly, -1 otherwise

### 6.3 Receive Message Service



This module waits for any messages arriving from the peer of the network. Each message has to be notified and passed to the Service of API Layer to process it. In the figure above we can see the methods of PeerManger and PeerCommunicator involved in the Receive Message Service.

NotifyMsgRcv	
Method	NotifyMsgRcv
Description	It notify the arrive of a new message
Input parameters	String PeerIDSender
Output parameters	None

EnbleRecvInstMsg	
Method	EnbleRecvInstMsg
Description	It starts the thread waiting for message coming from peers of the P2P Network. When a new message arrives it collects the message and it calls NotifyMsgRcv.
Input parameters	None
Output parameters	None

CloseAcceptMessageThread	
Method	CloseAcceptMessageThread
Description	It stops the thread waiting for message coming from peers of the P2P Network
Input parameters	None
Output parameters	None

ReceiveMessage	
Method	ReceiveMessage
Description	It provides the capability to receive the message coming from other peers.
Input parameters	None
Output parameters	String message - return the message arrived

## 6.4 Send File Service

This service is a client module used to exchange files between peer in the P2P Network. A peer can request a file to another peer and it is possible to receive the file automatically.

The connection between peer to exchange files, is a point to point connection using TCP protocol. If a peer need to transfer the same file to more than one peer, it has to send a sequence of file one at time or it can create more threads (one for each connection) and transfers file in parallel.

<b>acceptingConnection</b>	
Method	acceptingConnection
Description	Accept the connection to send a file
Input parameters	None
Output parameters	TRUE if the file can provided FALSE otherwise

<b>trasferringFile</b>	
Method	trasferringFile
Description	Send a file to a specific peer
Input parameters	String ipaddress - Ip address of the recipient, String file_name - the complete path of the file to be transferred
Output parameters	TRUE if file is correctly transferred FALSE otherwise

<b>getFileName</b>	
Method	getFileName
Description	It return the name of the file transferred
Input parameters	None
Output parameters	String file_name - the name of the file trasferred

<b>getFileSize</b>	
Method	getFileSize
Description	It returns the size of the file trasferred
Input parameters	None
Output parameters	Long size – size of the file transferred

## 6.5 Receive File Service

This service is the server side used to exchange files between peer in the P2P Network. It waits for connection arriving from the peer of the P2P Network. Each request is granted by a specific thread and it can create multiple threads to manage multiple requests.

<b>startReceiveFile</b>	
Method	startReceiveFile
Description	It starts the thread waiting for connection coming from peers of the P2P Network for transferring files.
Input parameters	None
Output parameters	TRUE if the service starts successfully FALSE otherwise

<b>stopReceiveFile</b>	
Method	stopReceiveFile
Description	It stops the thread waiting for connection coming from peers of the P2P Network
Input parameters	None
Output parameters	TRUE if the service stops successfully FALSE otherwise

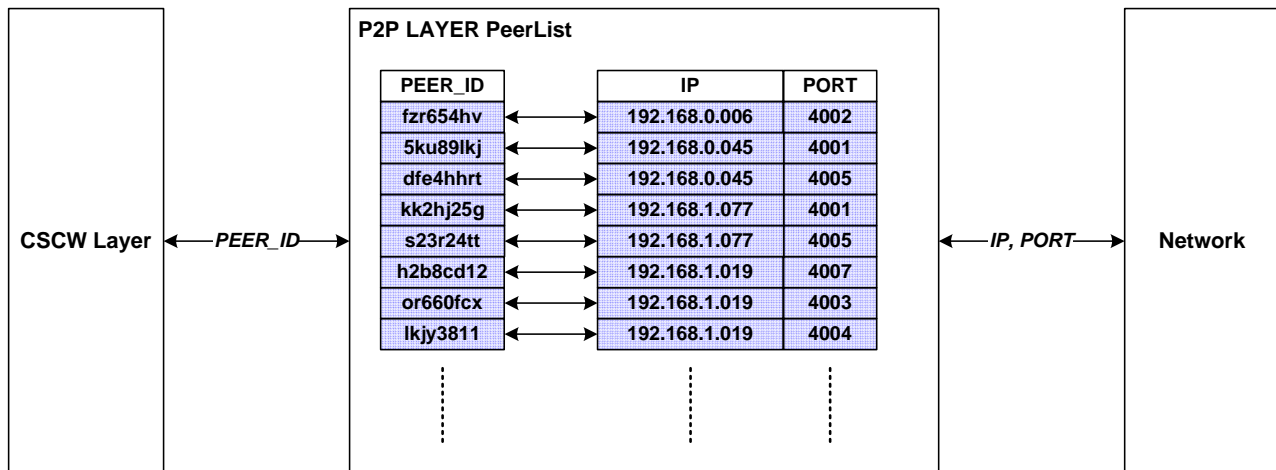
<b>receivingFile</b>	
Method	receivingFile
Description	It receives the file sent.
Input parameters	String – path of the file to receive
Output parameters	TRUE if file is correctly received FALSE if some error occur

## 6.6 Discovery Service

Discovery Service is used to interrogate the P2P Network to find all connected peer. When the service discovers a new peer on the net, gets the IP address and PORT number, assigns an univocal PEER\_ID and adds this new information in the PeerList. It also communicates the new PEER\_ID to the upper Layer.

PeerList allows the upper layers (e.g. CSCW) to refer to all peers only using their ID and not the IP-PORT data, as shown in the following figure:

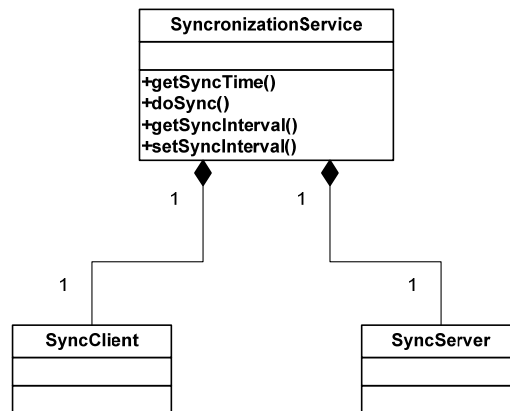




startStopScan	
Method	startStopScan
Description	It starts Tempester thread which scans the range of host address and it start Transponder thread waiting for connection from Discovery Client of the other peer of the network. When a new peer is found, the address IP and his port number is added to the peer list.
Input parameters	None
Output parameters	None

disconnect	
Method	disconnect
Description	Stop Transponder and Tempester
Input parameters	None
Output parameters	None

## 6.7 Synchronisation Service



This module uses a specific synchronisation algorithm to maintain synchronisation between peers of P2P Network. Each tool relies upon this service to execute cooperative, time critical, commands.

<b>getSyncTime</b>	
Method	getSyncTime
Description	It returns the synchronised time.
Input parameters	None
Output parameters	int64 time – time in microseconds

<b>doSync</b>	
Method	doSync
Description	It starts execution of the synchronisation algorithm.
Input parameters	None
Output parameters	None

<b>getSyncInterval</b>	
Method	getSyncInterval
Description	It returns the time interval between two consecutive sincronisations
Input parameters	None
Output parameters	Long Interval – time interval in milliseconds

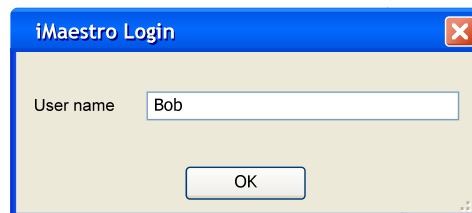
<b>setSyncInterval</b>	
Method	setSyncInterval
Description	Sets the time interval between two consecutive sincronisations
Input parameters	Long Interval – time interval in milliseconds
Output parameters	None

## 7 Client Manager: Starting Lessons/Workgroups

The Client Manager (CM) is the application required to start and manage a work session that can be single user or cooperative. The Client Manager is a Win32 application providing the capability to allow:

- User login
- Displaying a list of Lessons available in the P2P network
- Displaying a list of Lessons available in the local PC
- Displaying a list of roles for selected lesson
- Selecting a role and downloading a cooperative lesson file linked to that role
- Automatic start of the lesson downloaded

After Client Manager starts, the User sees the login form and insert the username that identify the user into the P2P network.



After the login the User has the possibility to see a list of lessons saved in the local machine (left list) and the list of lessons shared in the P2P network (right list). To see the second one it is necessary to connect to the P2P network using the menu “Connection”.

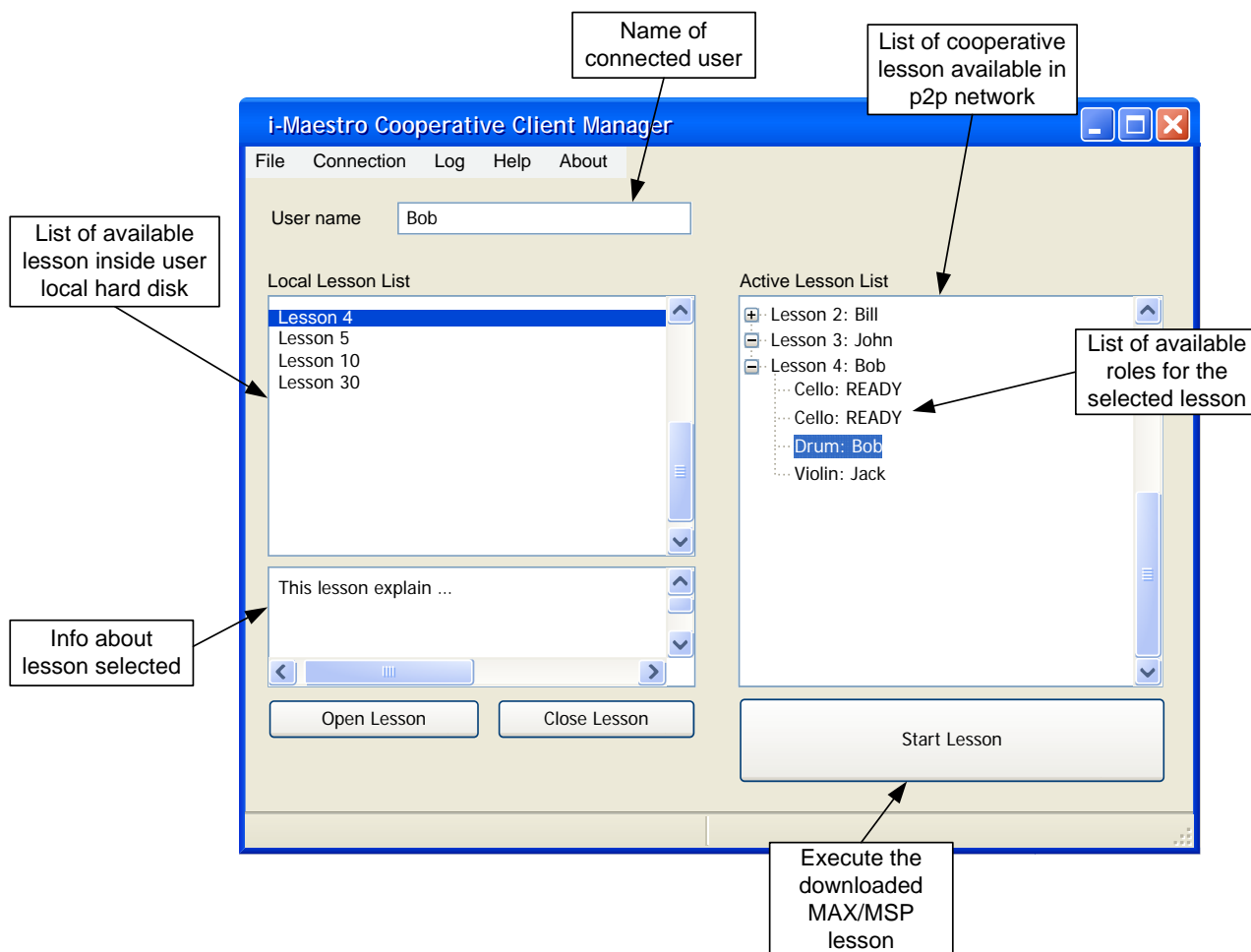
Using the “Local Lesson List” the User has the capability to see a description of the lesson selected and to start a new cooperative lesson.

After this choice, the Client Manager communicates to the other peers that the role is not available and waits until Users have covered their roles. When all Users have filled all available roles and depending on the structure of the lesson, the cooperative lesson can start. Actors are free to leave the lesson simply closing the active Lesson.

A cooperative Lesson contains generic roles (parts of a score, some type of exercise, other types of substructures) and each of them can be assigned to one User. Each Lesson contains inside information of the Workgroup and the max number of members of the Workgroup is given by the number of available roles in the lesson.

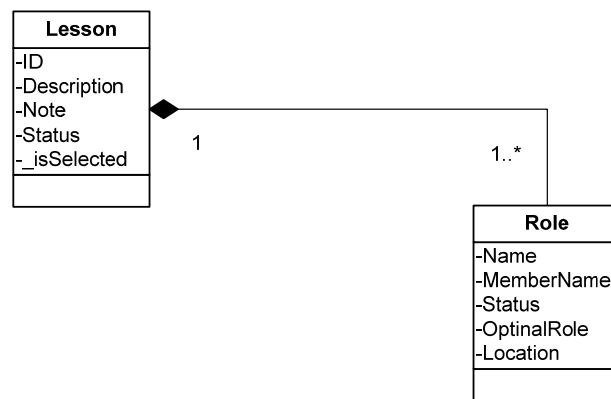
When a User has chosen a role inside a lesson, the Client Manager provides the features to download the content linked to the role, retrieving it using the URI specified into the XML file of the Lesson.

Lesson downloaded is automatically started by the Client Manager as an independent process. Now, the execution of the lesson doesn't depend from the Client Manager anymore, but it follows the logic decided by the lesson creator. Because we talk about cooperative lessons, each lesson has to have a Music Execution Service to exchange information between peers.

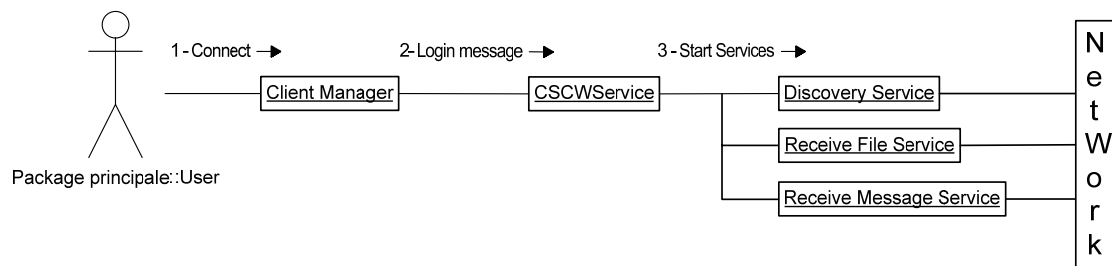


### Sequence of operations to setup a cooperative Lesson

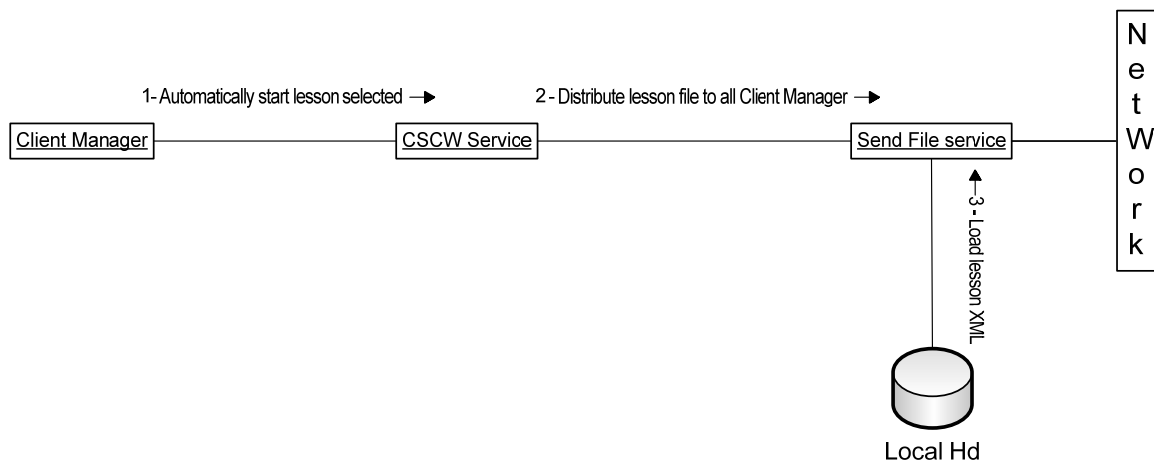
- 1) The User uses a Web Interface to search lesson inside a School Server and download its metadata information in xml format. Information is saved in a specific directory. This point doesn't concern the Client Manager and P2P Network.
- 2) The User starts Client Manager giving its own user name
- 3) The Client Manager automatically searches for the xml file of the lesson inside a specific local folder. When the xml file is found and loaded, the ClientManager creates a Lesson object in the Local Lesson List. The Lesson objects are made by Roles objects and they are put in a Lesson list. Note that lesson and role info are the same info described in the cooperative lesson metadata as in §13.5.



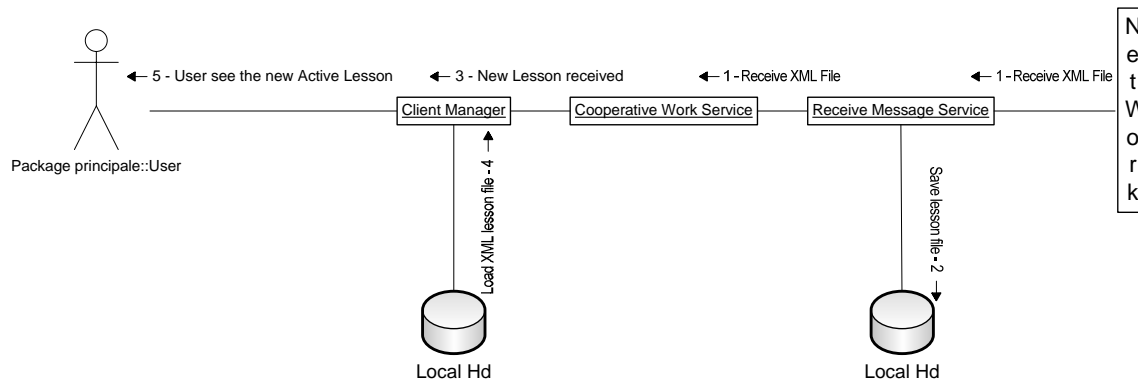
4) When the User chooses “Connect”, from the “Connection” menu, all the P2P services start.



If the User starts one of the local lesson the Client Manager add this lesson in the Active Lesson List and sends the xml Lesson file to all peers in the P2P Network using CSCW Service. In this way each Client Manager on the network can see the available lessons for cooperative work.



5) When a peer receives this xml file it copies in a specific folder, parses the xml and creates the correspondent lesson and roles objects in the Active Lesson List. Now the User receives the lesson arriving from the other Client Manager peer and s/he can see roles linked to the lesson in her/his GUI.



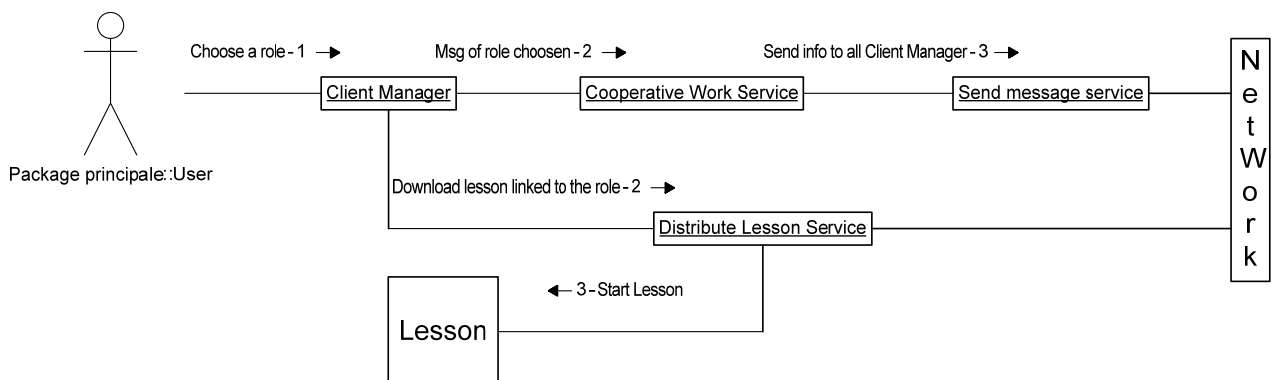
6) The User can select a lesson and see the list of available roles and can decide to associate only to one of these roles (the roles with the status READY). When User selects his role, the information is sent in the P2P network and all CM-peers update the role info. The role pass from the status READY to BUSY and the lesson file linked to the role is downloaded from the URI specified inside the correspondent field in the instance of the class of the Role chosen. When the lesson is downloaded, it is unzipped (if it is compressed) and then it is automatically started by the Client Manager, launching a new process executing the correspondent executable file.

User is now ready to take part to the cooperative lesson depending on the logic specified by the lesson creator. The format of the message is:

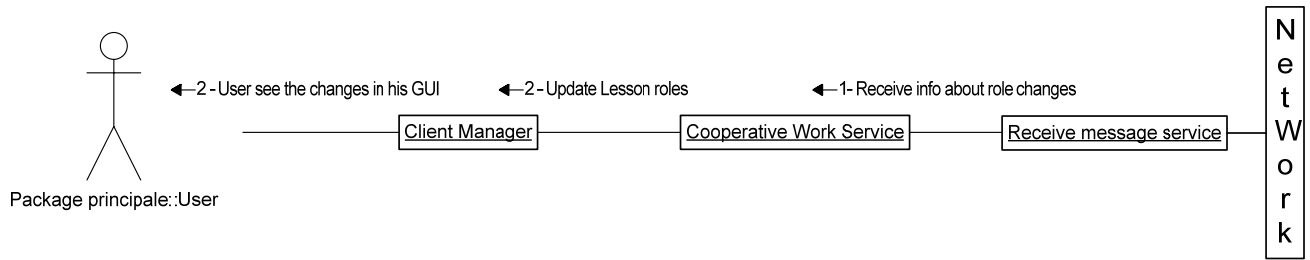
ROLECHOOSE <lessonId::teacher> <rolename> <username>.

The <lessonId::teacher> parameter corresponds to the “ID” parameter of the Cooperative Lesson description metadata and name of the teacher that starts the lesson (see § 13.5); <rolename> and <username> are the “RoleName” and “MemberName” parameters of the Role description.

When a peer receives a message as the previous one, it searches the lesson and role and updates the username info.



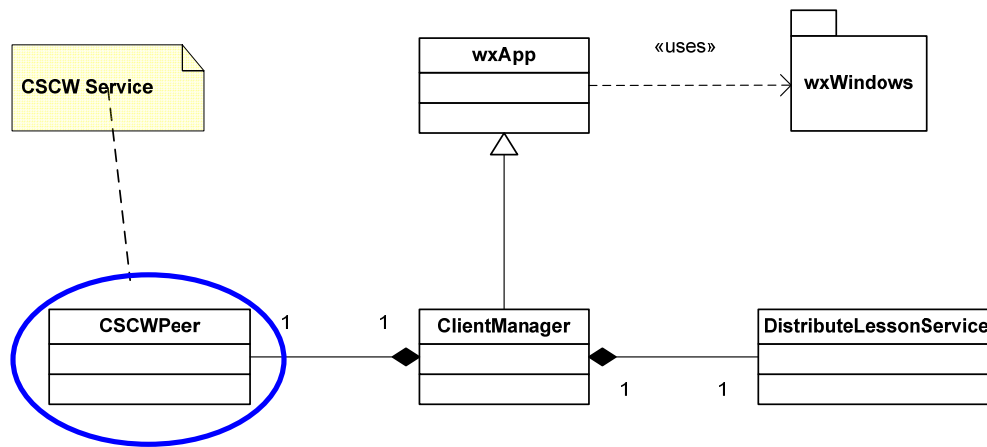
7) All Client Manager peers receive info about roles status. Earlier the role was available for all, now the role is booked and nobody can choose it again.



The previous collaboration diagram shows the message exchange during the setup of a cooperative lesson. As regards the single user lesson, the setup step is simpler: a single user lesson is a cooperative lesson with only one role.

In this way user loads his lesson and choose the only one role available, then system download and/or open the lesson. The lesson is not cooperative but it can contain Music Execution Service with CSCW Service and P2P service for localhost communication.

## 7.1 Class Diagram of ClientManager and its services



The figure shows the class diagram of the Client Manager and the services of the API layer that it uses.

Client Manager is a Wx application and it is composed by:

- **CSCWPeer Class**: it provides methods and functionalities to support cooperative work, such as messages delivering for adding user to a selected role, removing user from a role, updating available cooperative lesson in P2P Network
- **DistributeLessonService**: it gives the capability to download lesson after role user chosen and to start it as an independent process.



## 7.2 Distribute Lesson Service

This service is realized by the class `DistributeLessonService` and it provides methods to download lesson given a specific Uri and to open it specifying the lesson path.

<b>DistributeLessonService</b>
+downloadLesson() +openLesson() +decompressLesson()

<b>downloadLesson</b>	
Method	downloadLesson
Description	Download a lesson from the Uri specified in the input parameter
Input parameters	String location – the Uri where it is possible to find the lesson
Output parameters	TRUE if lesson is downloaded successfully FALSE otherwise

<b>openLesson</b>	
Method	openLesson
Description	Open the lesson using the path specified in the input parameter.
Input parameters	String path – local path of the lesson to open
Output parameters	TRUE if lesson is opened successfully FALSE otherwise

<b>decompressLesson</b>	
Method	decompressLesson
Description	It decompresses the lesson and recreates the original tree of lesson content.
Input parameters	String path – local path of the lesson to decompress
Output parameters	TRUE if lesson is opened successfully FALSE otherwise

### 7.3 Cooperative Session Data

In this section the information used to define a cooperative Lesson are described. Information is written into lesson metadata and they are used during lesson setup. This information is present in every lesson, both cooperative and single user lesson. For the single user lesson, there is only one role and user can complete the lesson alone.

<b>CooperativeLesson Description</b>		
<b>Data</b>	<b>Type</b>	<b>Description</b>
ID	String	Unambiguous code of the lesson
Description	String	Small description of the lesson
Note	String	Useful information to perform the lesson
Status	String	It specifies if the lesson is READY to start or if it is just started, bringing back the USERNAME who has started the lesson.
<b>Role description</b>		
RoleName	String	Role name
MemberName	String	Name of the Student who has joined the role
OptionalRole	Boolean	It specifies if the lesson role is optional or not to perform the lesson. If it is mandatory, lesson cannot start until the person has joined that role.
RoleStatus	String	The status of the role is READY if the role is free; otherwise it is BUSY because somebody has chosen it.
Location	String	URL or path where it is possible to find the lesson linked to the role

**Cooperative Lesson XML formalisation**

The set of metadata defined previously formalised by means of the following XML Schema:

```
<xs:schema ...>
  <xs:element name="CooperativeLesson">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="ID" type="xs:string"/>
        <xs:element name="Description" type="xs:string"/>
        <xs:element name="Note" type="xs:string"/>
        <xs:element name="Status" type="xs:string"/>
        <xs:complexType>
          <xs:element name="Role" maxOccurs="unbounded">
            <xs:sequence>
              <xs:element name="RoleName" type="xs:string"/>
              <xs:element name="MemberName" type="xs:string"/>
              <xs:element name="OptionalRole" type="xs:boolean"/>
              <xs:element name="Location" type="xs:string"/>
            </xs:sequence>
          </xs:element>
        </xs:complexType>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

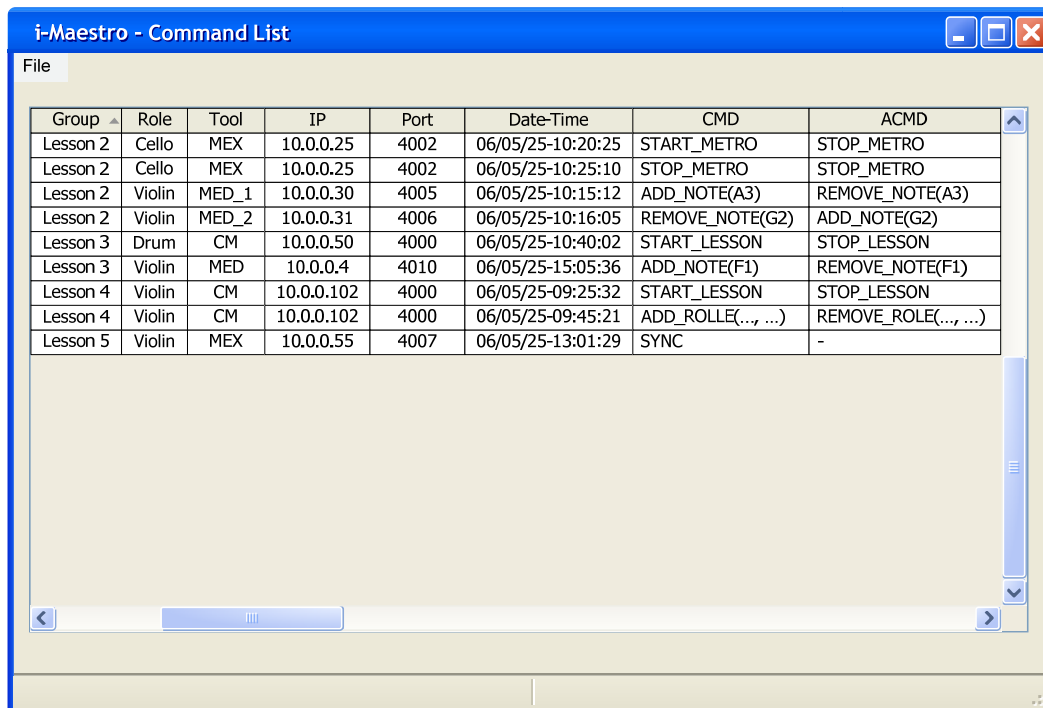
Example of XML file:

```
<CooperativeLesson>
  <ID>Unambiguous code of the lesson</ID>
  <Description>Description of the lesson</Description>
  <Note> Some note about the lesson</Note>
  <status>READY</status>
  <Role>
    <RoleName>Violin</RoleName>
    <MemberName>Tom</MemberName>
    <OptionalRole>TRUE</ OptionalRole >
    <Status>BUSY</Status>
    <Location>http://...</Location>
  </Role>
  < Role >
    <RoleName>Cello</RoleName>
    <MemberName></MemberName>
    <OptionalRole>TRUE</ OptionalRole >
    <Status>READY</Status>
    <Location>http://...</Location>
  </ Role >
  ...
</CooperativeLesson>
```

## 8 Client Manager: Monitoring and Controlling Cooperative work

### 8.1 Viewing and checking the log messages

Using an Administrator version of Client Manager it is possible to monitoring all the information passed through the Cooperative Work Service Layer (CSCW). For example it is possible to visualise the “Command list”, as shown in the figure below.



The screenshot shows a window titled "i-Maestro - Command List" with a menu bar containing "File". Below the menu bar is a table with the following data:

Group	Role	Tool	IP	Port	Date-Time	CMD	ACMD
Lesson 2	Cello	MEX	10.0.0.25	4002	06/05/25-10:20:25	START_METRO	STOP_METRO
Lesson 2	Cello	MEX	10.0.0.25	4002	06/05/25-10:25:10	STOP_METRO	STOP_METRO
Lesson 2	Violin	MED_1	10.0.0.30	4005	06/05/25-10:15:12	ADD_NOTE(A3)	REMOVE_NOTE(A3)
Lesson 2	Violin	MED_2	10.0.0.31	4006	06/05/25-10:16:05	REMOVE_NOTE(G2)	ADD_NOTE(G2)
Lesson 3	Drum	CM	10.0.0.50	4000	06/05/25-10:40:02	START_LESSON	STOP_LESSON
Lesson 3	Violin	MED	10.0.0.4	4010	06/05/25-15:05:36	ADD_NOTE(F1)	REMOVE_NOTE(F1)
Lesson 4	Violin	CM	10.0.0.102	4000	06/05/25-09:25:32	START_LESSON	STOP_LESSON
Lesson 4	Violin	CM	10.0.0.102	4000	06/05/25-09:45:21	ADD_ROLE(..., ...)	REMOVE_ROLE(..., ...)
Lesson 5	Violin	MEX	10.0.0.55	4007	06/05/25-13:01:29	SYNC	-

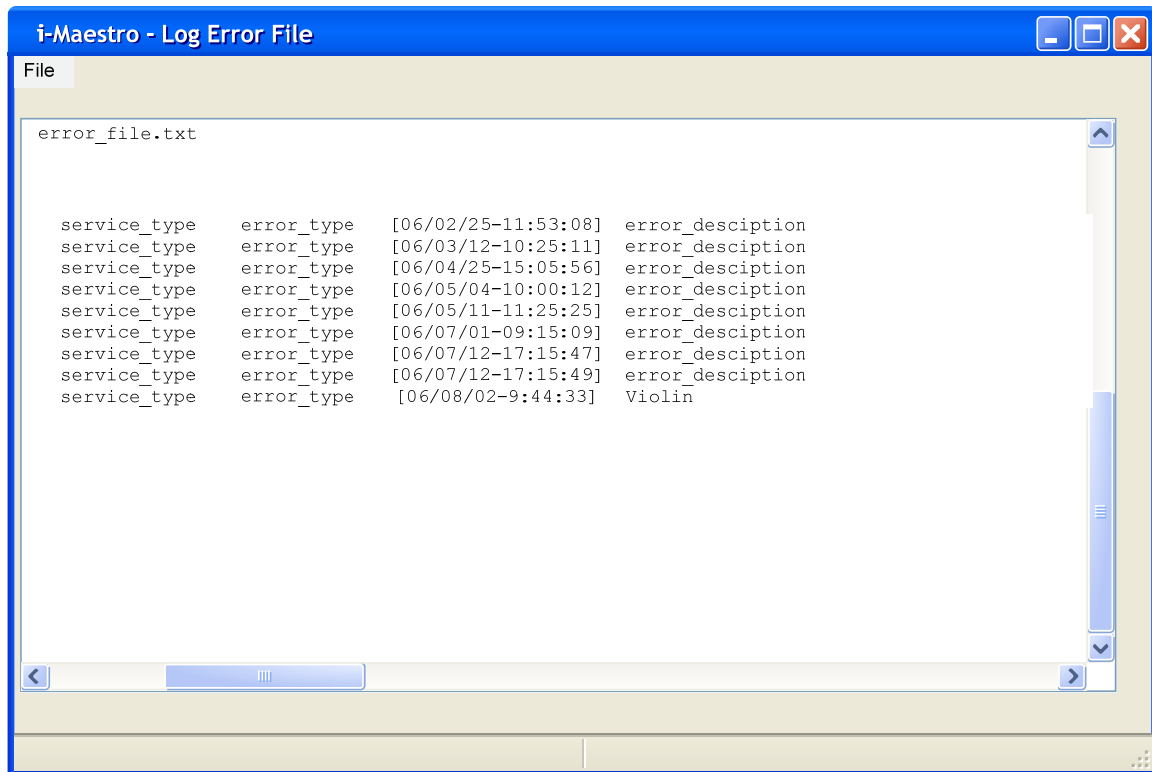
Each line represents a message passed through the CSCW. Columns show the information contained in each messages. A shortly description is shown below:

- Group: the name of the group that a user has joined;
- Role: the role chosen by the user;
- Tool: the name of the tool that is exchanging the message;
- IP: the sender IP address;
- Port: the communication port number used for connection;
- Date-Time: The date and the time of the message sent;
- CMD: command name;
- ACMD: the command to undo the previous one.

It is also possible to visualise the messages in a particular order clicking on the top of each column. For example, ordering the messages for IP address or for tool type, etc.

## 8.2 Viewing and checking the Error messages

In the same way it is possible to visualise the error log file created by the Cooperative Work Service and containing both the errors of this layer and the P2P errors.

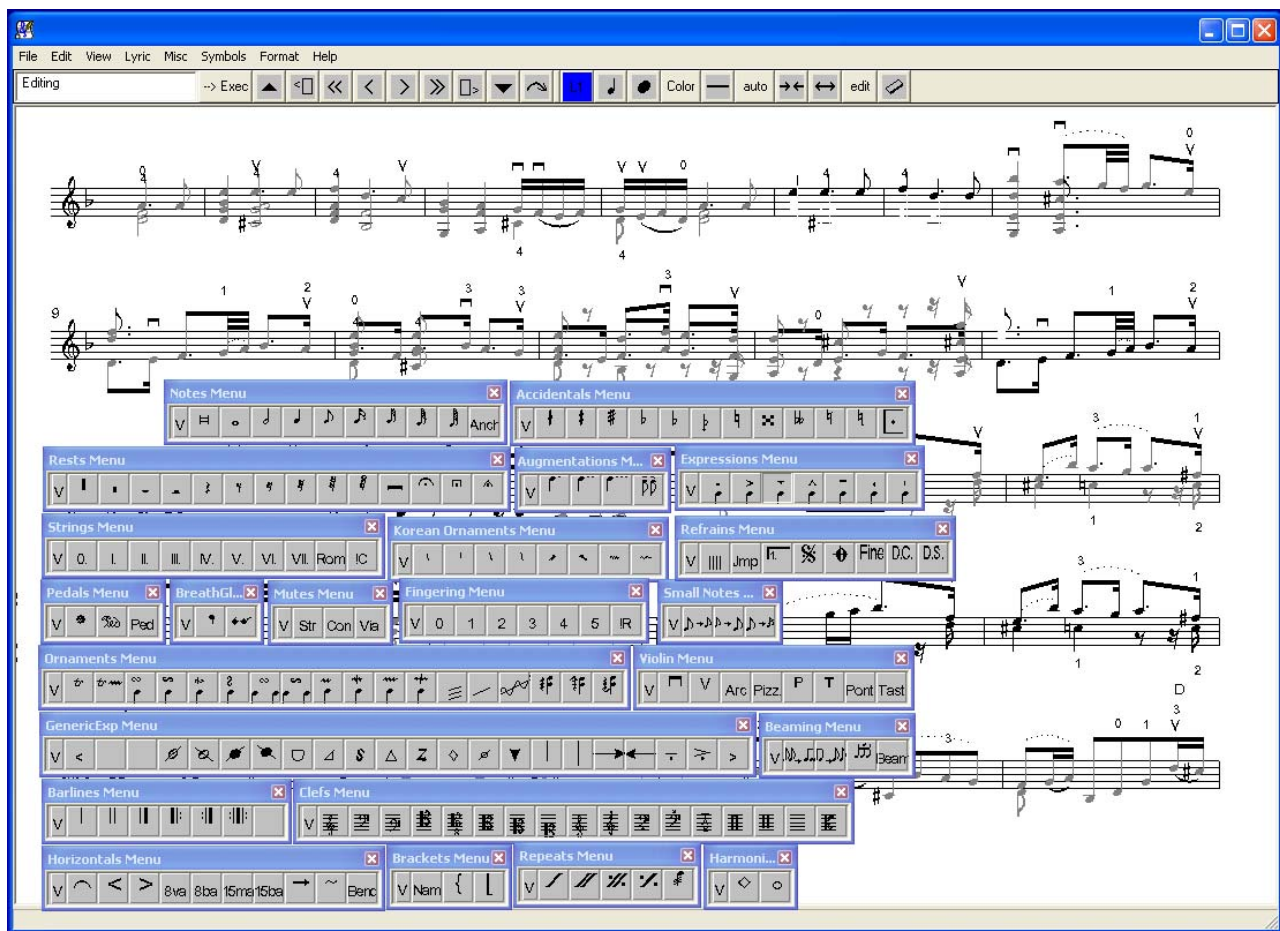


## 9 Cooperative Music Editor for SMR

### 9.1 Stand Alone SMR Music Editor

The stand alone SMR Music Editor is the music editor used to produce music score in SMR format. The functionalities of the SMR Editor are substantially the same of the Music Editor in MAX explained in the following Section 10 and have the following major capabilities:

- Music editing and reading, Writing the music notation and trying to play, with some constraints
- music score editing, main score and parts. Main score it is useful for producing music to be played for accompanying the pupil.
- music score printing, main score and parts. For producing the score that can be played without the computer.
- Navigation in the Music Editor Help
- Change of Metronomic indication directly in the reference Music Notation Model
- Navigating from music notation symbols to other multimedia aspects and content contained in the lesson. Active notes to get further explanation about the exercise: for example the explanation of an expression symbol (how it has to be executed), a link to a theoretical description document, a link to a video, to an animation, etc.
- Import from MIDI, from paper by using Optical Music Recognition, etc.
- Audio production from Music Notation, via MIDI, Play a music piece with different MIDI orchestrations
- Understand music notation structures and symbols, simulating their effects on expressive MIDI: expressions, articulations, ornaments, etc.
- Split and merge of voices to staves
- Transposition exercise and verification with the algorithm
- Piano reduction exercise and verification with the algorithm
- Guitar: converting notes to tablatures and vice versa
- Music notation, definition of interpretation semantics and performance evaluation rules

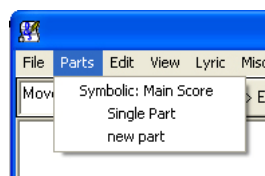


### 9.1.1 List of available functionalities

#### Menu “File”

- New
- Load
- Save
- Import MIDI
- Settings
- Encode SMXF
- Decode BitStream
- Close

Menu “Parts”: shows the list of the available single part of the score. The user can select which part visualise.



Menu “Edit”

- New Part
- Cut a Part
- Copy a Part
- Paste a Part
- Move Parts
- Transposition
- Splitting Layers -> Part
- Merging Parts -> Layers
- Joining Parts -> Multistaff
- Disjoin Multistaff -> Parts
- Move a Layer
- New Measure Column Before
- New Measure Column After
- Cut a Measure Column Before
- Copy a Measure Column
- Paste a Measure Column After
- New Single Measure Before
- New Single Measure After
- Cut a Single Measure
- Copy a Single Measure
- Paste a Single Measure Over
- Cut a Page Box
- Copy a Page Box
- Paste a Page Box

Menu “View”

- Computer View
- Print View
- Computer View Parameters
- Print Parameters
- Hide a Part
- Show Hidden Parts
- View/Identify Layer
- See Behaviour
- Fit Image to window

Menu “Lyric”

- New Lyric
- Edit Lyric
- Hide Lyric
- Select Lyric

Menu “Misc”

- Open All Menu Symbols
- Close all Menu symbols
- Jump to Symbolic
- See Links
- Edit Links



Menu “Symbols”

- Time Signature
- Key Signature
- Metronome
- Dynamics
- Fretboards
- Annotations
- Generic Text
- Insert Page Text
- Insert Page Image
- Expressions
- Generic Symbols
- Ornaments
- Korean Ornaments
- Mutes
- Harmonics
- Fingering
- Strings
- Pedals
- Violin
- Percussions
- Breath/glass
- Beaming
- Repeats
- Barlines
- Clafs
- Brackets
- Horizontals
- Refrains

Menu “Format”

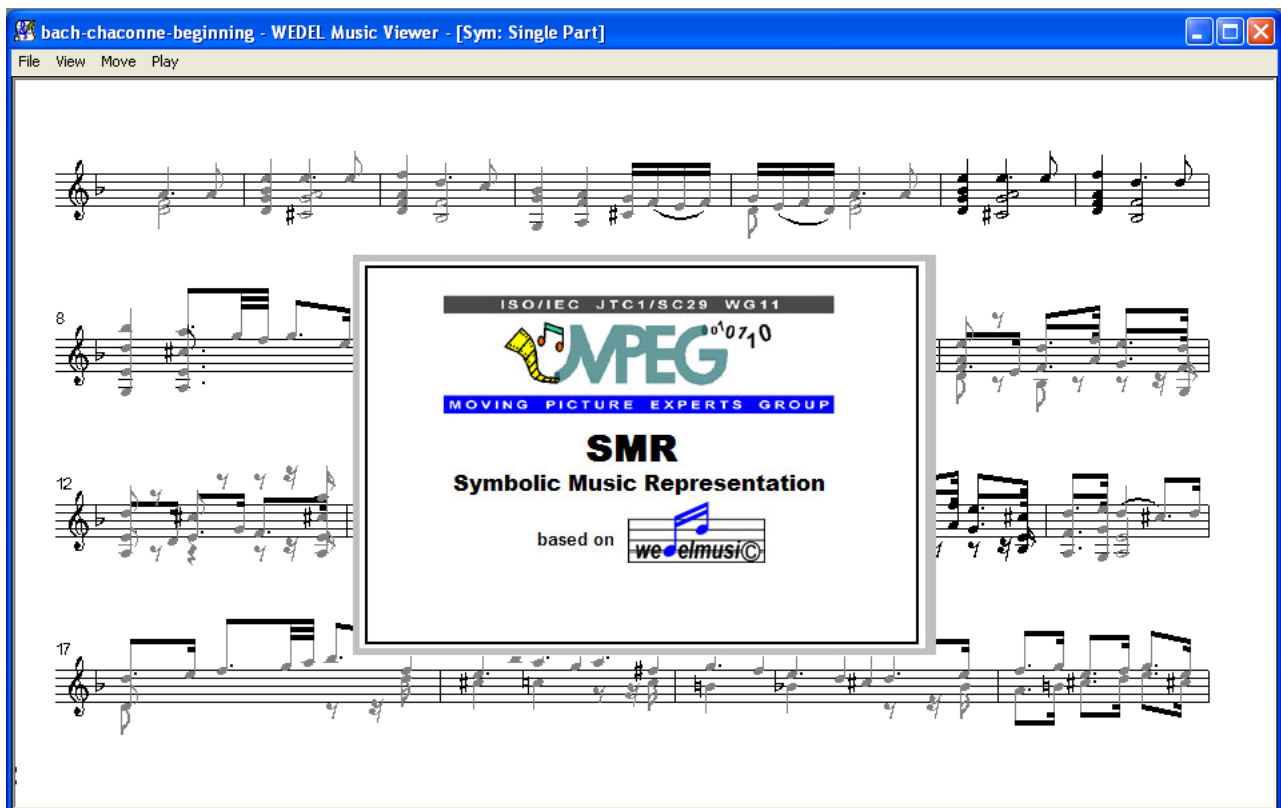
- Checking
- Justification
- Line Breaking
- Auto Beaming
- Beaming
- Up/Down Stem
- Edit Milla
- ReLoad Milla
- Debug Milla
  - Debug Milla Insertion
  - Debug Milla Positioning
- See Invisible Figures
- See Anchorages
- Edit Font Table
  - Music Font 1
  - Music Font 2
  - Music Font 3
  - Custom Font 1
  - Custom Font 2
  - Custom Font 3

- Custom Percussions
- Custom Pedals
- Score Excerpt

Menu “Help”

- Help
- About

## 9.2 Stand Alone SMR Music Players MPEG



### 9.2.1 List of available functionalities

Menu “File”

- Load, to load a music score in SMR format
- Import MIDI, to import a MIDI file
- Key Change, to change and transpose a part of the score
- Exit, to close the SMR Viewer

Menu “View”: shows the list of the available parts. It is possible to select the part to be visualised

Menu “Move”

- Top (Ctrl-T), to go at the first measure of the score
- Forward 1 measure (Ctrl-F)
- Forward 5 measures (Ctrl-Shift-F)
- Backward 1 measure (Ctrl-B)

- Backward 5 measures (Ctrl-Shift-B)
- Bottom (Ctrl-M), to go at the last measures of the score
- Jump (Ctrl-J), to go at the chosen measure

Menu “Play”

- Play MIDI (Ctrl-P), to play the MIDI version of the score
- Pause (Ctrl-U), to pause the playing
- Stop (Ctrl-S), to stop the playing

### 9.3 Cooperative Music Editor for SMR

Cooperative work functionalities will be implemented for the stand alone Music Editor.

### 9.4 SMR Music Player into MPEG4

MPEG-4 technology covers a huge media domain through the concept of synthetic and natural hybrid coding (SNHC) audio, and “symbolic” audio (like e.g. MIDI) content can be rendered and synchronised with other forms of media: images, video, graphic animations, etc. It further allows structured descriptions of audio content through a normative algorithmic description language associated with a score language more flexible than the MIDI protocol (MPEG-4 Structured Audio, SA). All these tools, though allowing to *derive* in someway a symbolic representation out of the information they carry, are to a large extent not enough to guarantee a correct coding of music notation as they lack for instance all kind of information about visual and graphic aspects, many symbolic details, a thorough music notation modelling, and many necessary hooks for a correct human-machine interaction. MPEG-7 also provides some symbolic music related descriptors; but they are not meant to be a means for coding symbolic music representation as a form of content. On the other hand, MPEG SMR content stream contains a complete Symbolic Music Representation and it may be rendered in synchronisation with other audio-visual elements, video, audio, images, animations, 2D and 3D scenes, etc.

MPEG SMR permits the realization of new applications in which multimedia and music notation may take advantage and enrich each other in terms of functionalities. For example in the areas of edutainment, entertainment, courseware production, music notation subtitles during concerts and operas, music rendering in archives, piano keyboards with symbolic music representation and audiovisual capabilities, mobile devices with music display capabilities, etc.

All these applications may take advantage from the MPEG-4 technology with SMR.

#### 9.4.1 Integration of SMR in the MPEG-4 player

Symbolic Music Representation (SMR) is going to be integrated into MPEG-4 format by:

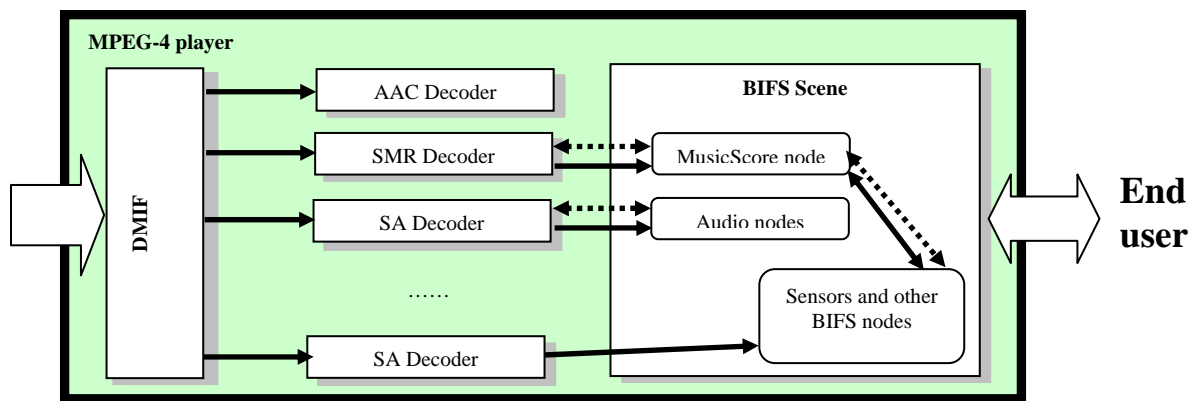
- defining a toolset including as its main part an XML format (Symbolic Music eXtensible Format or SM-XF) for a text based symbolic music representation, to be used for interoperability with other symbolic music representation/notation formats and as a source for the production of an equivalent binary information that may be stored in files and/or streamed through a suitable transport layer as the other MPEG-4 data; the other tools are a Symbolic Music Formatting Language (SM-FL) and the simple Symbolic Music Synchronisation Information (SM-SI);
- adding an SMR Object Type for the delivery of a binary stream containing SMR, synchronisation information, and rendering rules; the associated SMR decoder allows to manage the received information to add the necessary “musical intelligence” for the interaction with the user according to the spirit of MPEG-4 in terms of interactive capabilities;

- specifying the interface and the behaviour for the symbolic music representation decoder and its relationship with the MPEG-4 synchronisation and interaction layer (MPEG-4 BIFS nodes, where BIFS stands for Binary Format for Scenes description, a mark-up binary language with an equivalent textual format defined in MPEG-4 XMT).

The MPEG SMR content can be produced by using an appropriate converters and/or a native SMR music editor which is a coder for the SMR format. One instance is already available in the MPEG forum as derived from the WEDELMUSIC editor.

Then, an MPEG-4 SMR-enabled encoder tool can multiplex the different SMR files into an MPEG-4 binary stream (standard XML binarization is also available in MPEG). The SMR binary stream contains in one stream information about music symbols (SM-XF), their synchronisation with other media in time and space (SM-SI), and possible rendering rules for formatting music symbols (SM-FL).

It is also possible to enhance the flexibility of the SMR toolset by means of the direct usage of MIDI, being this latter a protocol based on symbolic information. A native support inside MPEG-4 Structured Audio (through which MIDI information can be carried over MPEG-4) allows this synergy: the SMR decoder provides a direct support of SA streams containing MIDI objects.



**MPEG-4 Player Architecture**

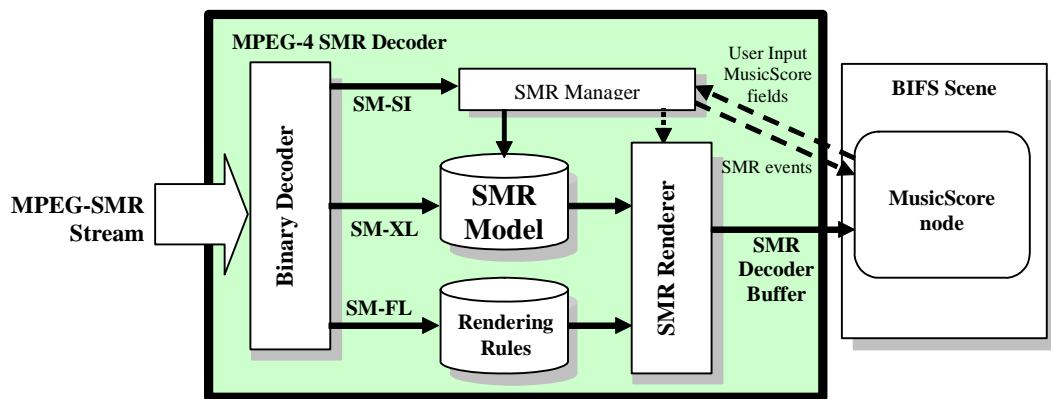
The figure above represents the structure of an MPEG-4 player endowed of MPEG-4 SMR decoder. The player uses the MusicScore SMR Node (a node is the basic element of MPEG-4 Scene Description hierarchical trees) to attach the symbolic music information to the scene (or even by exploiting functionality of other BIFS nodes) as decoded by the SMR decoder. The user can interact with the SMR content (to change a page, view, transpose, and so on) using sensors in association with other nodes defining the audiovisual, interactive content and routing them to the music score. The user sends commands from the SMR node fields to the SMR decoder (dashed lines in the figure), which generates a new view to be displayed in the scene. In addition, user client tool automatically converts MIDI files (through a specific algorithm) into SMR on the client side and render them. Similarly, the server might only deliver the SMR.

The SMR decoder is activated when an SMR elementary stream or file is received. The configuration of the SMR decoder is in its stream or file header.

The SMR decoder is capable to cope with different kind of chunks: main score (the SM-XL coding itself), single parts (SM-XL coding of the single part), formatting rules (SM-FL), synchronisation information (coded according to SM-SI), lyrics (coded as SMR multilingual lyric SMR code in XML), and fonts.

Internally the decoder contains the SMR object oriented model and an engine for the automatic formatting of SMR on the visual domain. That engine is capable of processing statements written Symbolic Music Formatting Language to learn the rules that have to be applied to format the music information [2]. The

formatting engine of the SMR decoder for SMR rendering includes two main subsystems and algorithms. One for the positioning of symbols and a second for the arrangement of music symbols according to the justification and line-breaking parameters. The information coded into Symbolic Music Formatting Language allows encoding and thus providing specific rules for each SMR information. Thus the decoder can take decisions about the positioning and the justification of music notation symbols in the visual domain, according to the shape of it. This is very important in an interactive environment such as the MPEG-4 player in which the size of the view may change, since the internal window in which the SMR is show may be different from one case to another from one instant to another. The SMR decoder architecture is shown in the figure below:



MPEG-4 SMR Decoder Architecture

### 9.4.2 Authoring an MPEG-4 SMR application

In order to create an MPEG-4 application based on SMR a specific BIFS (MPEG-4 Scene Description) coder is needed. It may include the simple integration of the SMR information to represent it in the visual domain as well as a set of other BIFS nodes and commands to create and enable the user interface and the related interaction prepared to create the needed feedback from the user to the SMR decoder, via the SMR node.

For example, the music editing score application reported below presents a set of elements that have been created in BIFS as described in the following coding example.

```
#MUSIC SCORE
  Transform2D {
    translation 0 0
    children [
      DEF SCORE_TS TouchSensor {}
      ScoreShape {
        score DEF SCORE MusicScore {
          url 5
          size 800 600
          startTime -1
          stopTime -1
          loop true
          hyperlinkEnable true
        }
        geometry Bitmap {}
      }
    ]
  }

#PLAY Button
  Transform2D {
    translation -550 -270
    children [
      DEF PLAY_TS TouchSensor {}
      Shape {
        appearance Appearance
        { Texture ImageTexture { url 9 } }
        geometry Bitmap {}
      }
    ]
  }
  DEF PLAY_SEL Conditional {
    buffer {
      REPLACE SCORE.stopTime BY -1
    }
  }

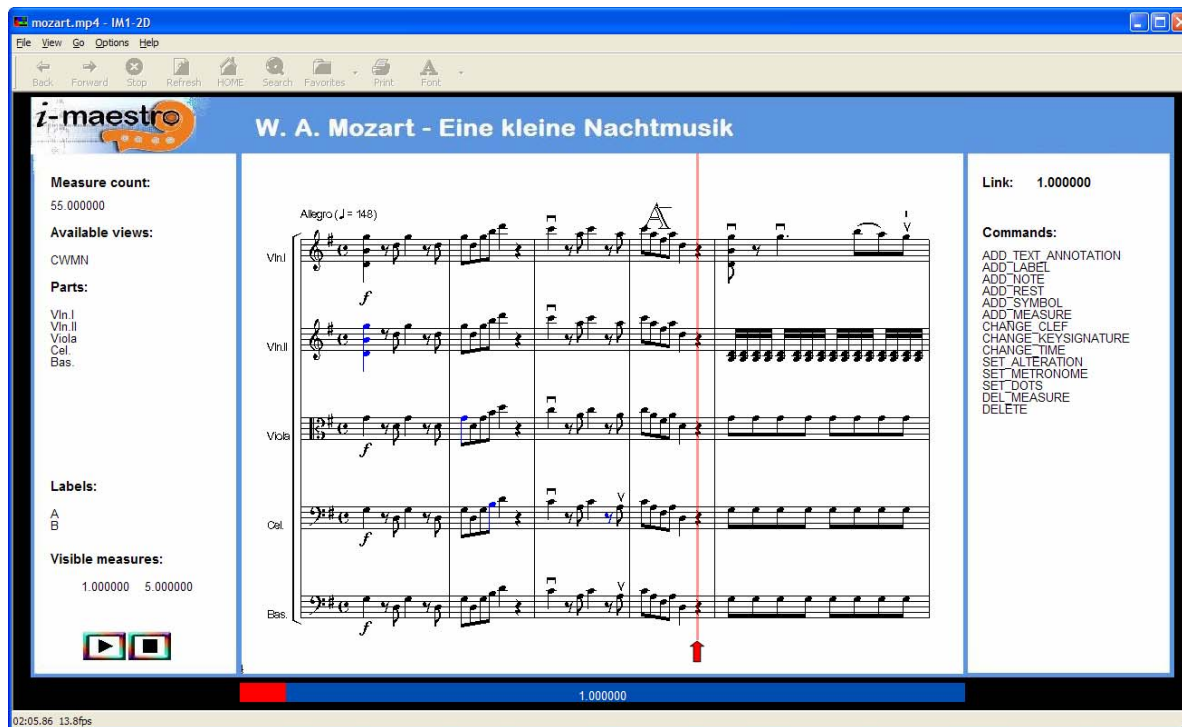
#PARTS
  Transform2D {
    translation -610 120
    children [
      Shape {
        appearance Appearance {
          material Material2D {
            emissiveColor 0 0 0
            filled TRUE
          }
        }
        geometry Text { string ["Parts:"]
          fontStyle FontStyle {
            family ["Arial"]
            justify ["LEFT"]
            size 18
            style "BOLD"
          }
        }
      }
    ]
  }
  Transform2D {
    translation -610 90
    children [
      Shape {
        appearance Appearance {
          material Material2D {
            emissiveColor 0 0 0

```

```

        filled TRUE
    }
}
geometry DEF PARTS Text {
    string ["Elaborating..."]
    fontStyle FontStyle {
        family ["Arial"]
        justify ["LEFT"]
        size 16
    } } }
}
}

```



### MPEG-4 SMR applications coded in BIFS

For example, with the MPEG-4 SMR application of SMR it is possible to perform several interactive operations with the SMR model. For example, other BIFS elements are accessible by means of the mouse click; such as the Play button for music education with integrated music score editing/interaction capabilities reported in Figure above which presents a set of elements that have been created in MPEG-4 BIFS as described in the associated coding segments. In this example, there are other elements coded in BIFS, such as the slider, the measure count, links, commands, etc., in practical all the graphics is obtained by using BIFS coding.

In the above described MPEG-4 application of SMR it is possible to perform several interactive operations with the SMR model. The BIFS elements which are accessible by means of the mouse click are the

- Play button to start the execution (it is synchronised with the scene using the synchronisation information stored in the SM-SI), the "current" playing position is shown by using a BIFS made indicator (a red line and a red arrow);
- Stop button to stop the player execution;
- Slider (on the bottom) to browse the score by scrolling the measures from the first to the last;
- Labels on the left to jump at the corresponding label assigned to the score measures, the visual representation is redrawn consequently;
- Parts on the left to select different views of the same information, in this case, the view corresponding to the rendering of the single parts (in the figure the main score is presented);

- Available views on the let can be used for activating the rendering of the SMR information according to different modalities, for example: Braille format, tablature representation, piano roll, etc.;
- Commands list on the rights allows the user to send commands to the SMR decoder and model. For example to add or change music notation symbols, or to set some parameters. A quarter note can be inserted into the score selecting the command and then clicking in the score position in which the note has to be placed.

The user may interact with the visual representation of SMR in the visual frame with mouse clicking. This interaction allows sending to sending at the SMR decoder the position of the mouse that is used to identify, for example, the selected position for a symbolic element to be added, or the selected symbolic element to interact with it, and/or to provoke other events. For example, clicking on a note that present an additional link (such as notes marked in blue in the example) the link associated with a note is shown on the right side. The provoked event can be used to start other audio visual effects such as the play of a video, the play of an audio or of an animation, to change the scene, to navigate in a tutorial, or a as a menu button, etc.

With the proposed example we have demonstrated that the usage of the MPEG-SMR allows producing a large range of music related applications coded in MPEG-4. This permits to avoid addressing the need of creating/coding different software tools every time a new application has to be created. In fact, with this approach the focus is move from the tools to the content. The application is in effect a simple MPEG-4 file, which can be streamed from a broadcast service provider to your TV set top box or any other compliant decoder.

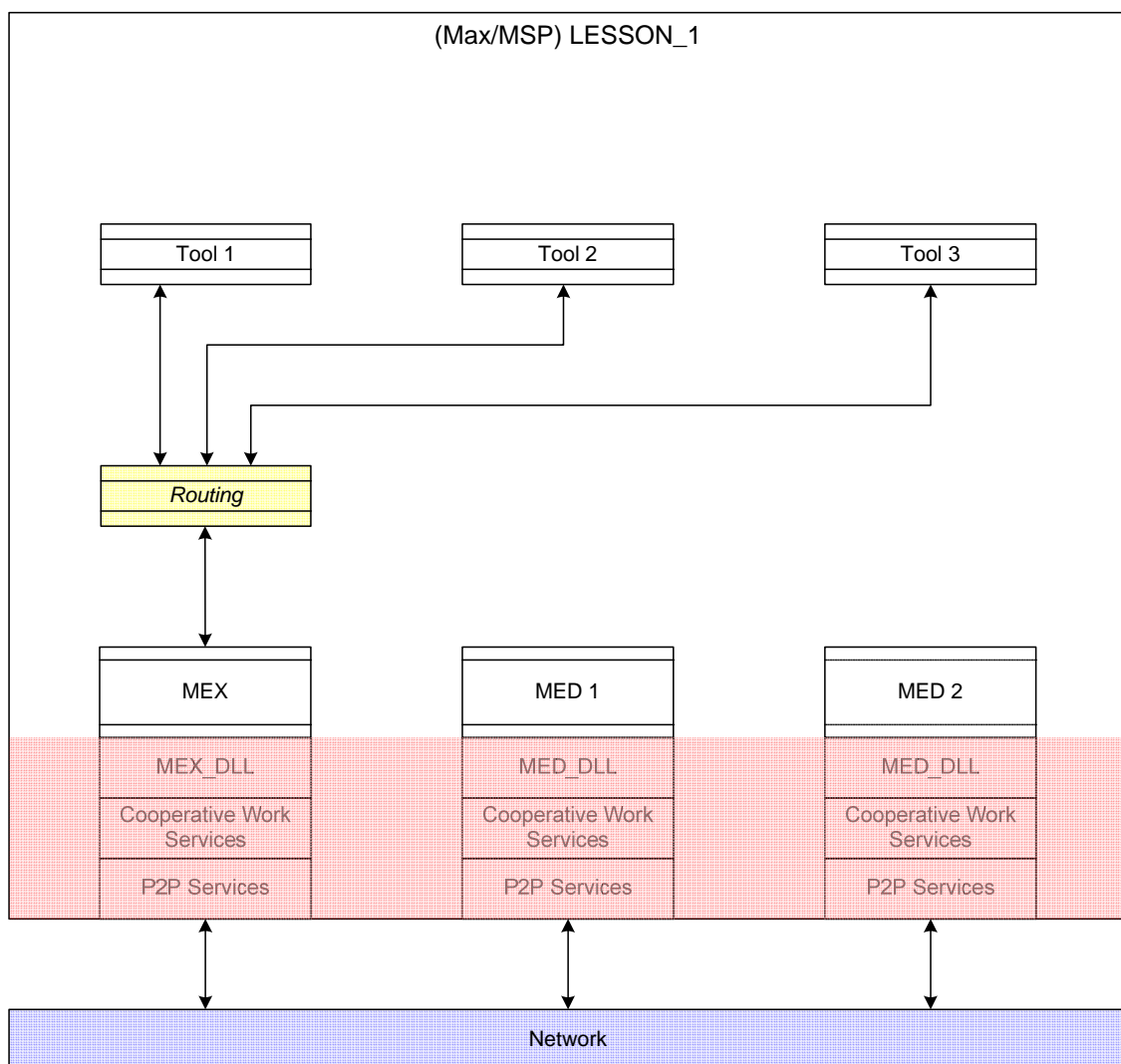


## 10 MAX and the Music Editing Service (MED) and the Music Execution Service (MEX)

Music Editing Service and Music Execution Service are two Max/MSP externals that provide cooperative work support for Max/MSP environment. The MEX external is use for binding cooperatives tools on different peers, such as distributed metronome, so each cooperative Max/MSP external can exchange messages with other tools on the P2P network. It is mandatory the presence of one and only one MEX external for each cooperative lesson.

The MED external provides distributed music editing and viewing support, including a set of commands for score navigation and information handling. Each lesson can contain up to eight MED externals, numbered with unique ID in order to avoid addressing problems.

The following figure shows a basic cooperative lesson configuration in Max/MSP environment.



## 10.1 Max/MSP Data Types

The following table shows the four basic atomic Max types (bang, int, float, symbol) and the list type used as multi-type collector. A data of any type is like a message that flows from an object to another, bringing information. The bang message has no information. It tells the receiver only to execute immediately its functionality.

Max Type	“C” Type	Use
Bang	-	Action
Int	Long	Integer number
Float	Float	Real number
Symbol	Char* (string)	“One word” string
List	-	MultiType Array

## 10.2 Max/MSP Cooperative Lesson

First of all it is mandatory that P2P and CWS layer in the Music Execution Service has to be started using different ports for server modules, respect of the same modules of the Client Manager, otherwise there will be a conflict during connection start-up of the Music Execution Service.

After that a cooperative lesson is executed as an independent process by the Client Manager, the Music Execution Service inside the lesson starts its Computer Work Support Layer and P2P Layer.

The first operation of the Music Execution Service is sent a localhost message to the client manager to receive lessonId of the lesson chosen from the user in the Client Manager. The knowledge the lesson id is useful when the Music Execution Service starts up the services of CSCW layer. Client Manager replies with such message:

<lessonId::teacher>: ID values of the lesson loaded and defined inside the metadata followed by name of the user that activate the lesson (a teacher or student supervisor). This value represents the name of the group that Music Execution Service will join for group working. Moreover, at the time of group joining, Music Execution Service registers itself to a specific lesson role, provided by the lesson that uses it.

The Music Execution Service can also ask CSCW the list of active roles of the lesson. With these info, it is possible to manage the lesson execution, for example waiting for the arrival of all role-users. Obviously this feature depends on the lesson logic defined by the lesson creator.

Then, when message comes, it is parsed to understand:

- 1) The type of the command (START or STOP command, generic SENDMESSAGE command)
- 2) the recipient role (RoleName parameter inside role description metadata)
- 3) the recipient tool name
- 4) and possible tools parameters

Then the command is sent to the outlet of the Music Execution Service and a routing Max/MSP object is used for delivering message to the correct cooperative tool (Max object).

## 11 MED: SMR Music Editor General Controls for MAX

This module is an External Max/MSP graphic object, which gives the capability of score editing and score-following. This module supplies the synchronisation support to edit music score between workgroup members. The module doesn't use the Music Execution Service to exchange information between peers; the structures and the services used to manage the cooperative editing are embedded in the module itself.

Cooperative editing can involve mainly scores where user can insert, delete, modify notes and symbols. It manages a commands log executed by each user to keep trace of notes and symbols modified.

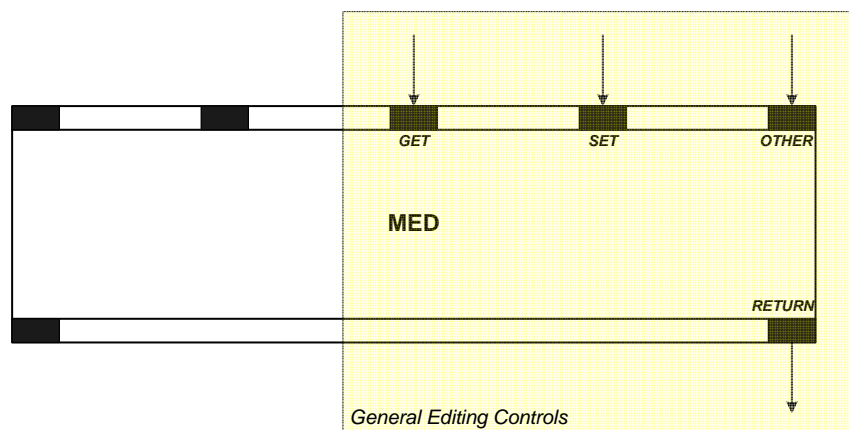
Teacher or Coordinator Student can choose to save cooperative work at any time; in this case all group members receive new document version and the change log is reset.

The API to interact with the Music Editor is defined by Max/MSP Inlets which represent the input to send to the Music Editor.

The following methods are available to interact with the Music Editor. They can be grouped in three principal groups:

1. "Get" methods to recover information about parameters and info of the score
2. "Set" methods to set parameters of the score
3. "other" methods

All three groups are linked to a specific inlet and there is an outlet to send info from the Music Editor to the other objects. So, there is an inlet for each group and only one outlet as shown in the following figure:



### 11.1 Inlets for set methods

The inlet for "Set" methods (the fourth from the left) receives as input the name of the method (without the "set" prefix) and the list of the parameters. If there are more than one parameter (for example an array string), the call to method is send as

`<methodname without "set" prefix> <value1> <value2> <value3> ...`

For example, to set the first visible measure, the message sent to the "Set" inlet is:

**FirstVisibleMeasure 9**

Or to set the size of the score the message is:

**Size 640 480**

<b>setArgumentOnExecute</b>	
Method	setArgumentOnExecute
Description	It indicates arguments for the commandOnExecute command.
Input parameters	List of arguments (depending on the type of command)
Output parameters	List ("ArgumentOnExecute" args): - args: List of arguments (depending on the type of command)

<b>setCommandOnExecute</b>	
Method	setCommandOnExecute
Description	It indicates the command to be executed by the user
Input parameters	Symbol command_sym – command to execute
Output parameters	List ("CommandOnExecute" command_sym): - Symbol command_sym – command to execute
List of possible commands	<ul style="list-style-type: none"> <li>• "ADD_TEXT_ANNOTATION" the first value in <i>argumentsOnExecute</i> contains the text to be added to the score in the position where the user will click via the <i>mouseOnExecute</i></li> <li>• "ADD_LABEL" the first value in <i>argumentsOnExecute</i> contains the label text to be added to the measure where the user will click via the <i>mouseOnExecute</i> if the measure already has a label the label is substituted</li> <li>• "ADD_NOTE" the first value in <i>argumentsOnExecute</i> contains the note duration: D1, D1_2, D1_4, D1_8, D1_16, D1_32, D1_64; the second value indicates the notehead type: "CLASSIC", ... the note is inserted where the user clicks or it is added to a chord if sufficiently near to another note/chord.</li> <li>• "ADD_REST" the first value in <i>argumentsOnExecute</i> contains the rest duration: D1, D1_2, D1_4, D1_8, D1_16, D1_32, D1_64; the rest is inserted where the user clicks via the <i>mouseExecute</i>.</li> <li>• "SET_ALTERATION" the first value in <i>argumentsOnExecute</i> contains the alteration to be set on the note, it can be: "SHARP", "DSHARP", "FLAT", "DFLAT", "NATURAL". The alteration is set to the note where the user clicks via the <i>mouseExecute</i>.</li> <li>• "SET_DOTS" the first value in <i>argumentsOnExecute</i> contains the number of dots to be set on the note, it can be: "0", "1", "2". The dots are set to the note where the user clicks via the <i>mouseExecute</i>.</li> <li>• "ADD_SYMBOL" the first value in <i>argumentsOnExecute</i> contains the symbol to be added on the note/rest/measure, it can be: "STACCATO", "TENUTO" or any symbol defined using the formatting language. The symbol is added where the user clicks via the <i>mouseExecute</i>.</li> <li>• "ADD_MEASURE" adds a measure to the score, the first value in <i>argumentsOnExecute</i> can be: "BEFORE", "AFTER" or "APPEND", the second value in <i>argumentsOnExecute</i> indicates the measure with respect to the new measure is added. The second value is necessary only for <i>execute</i> eventIn and it is not necessary for <i>mouseClickExecute</i> in fact in this case the measure where the user clicks indicates the measure with respect to the new measure is added.</li> <li>• "DEL_MEASURE"</li> </ul>

	<p>removes a measure of the score; the first value in <i>argumentsOnExecute</i> indicates the measure number to be removed. The value is necessary only for <i>execute</i> eventIn and it is not necessary for <i>mouseClickExecute</i> in fact in this case the measure where the user clicks indicates the measure to be removed.</p> <ul style="list-style-type: none"> <li>• "CHANGE_CLEF" changes the clef of a measure and for all the following until another clef change or to the end. The first value in <i>argumentsOnExecute</i> contains the clef type, it can be: "TREBLE", "SOPRANO", "BASS", "TENOR"..., The clef change applies to the measure where the user clicks via the <i>mouseExecute</i></li> <li>• "CHANGE_KEYSIGNATURE" changes the key signature of a measure and for all the following until another key signature change or to the end. The first value in <i>argumentsOnExecute</i> contains the key signature type, it can be: "DOdM", "FAdM", "SIM", ... The key signature change applies to the measure where the user clicks via the <i>mouseExecute</i></li> <li>• "CHANGE_TIME" changes the time of a measure and for all the following until another time change or to the end. The first value in <i>argumentsOnExecute</i> contains the time, it can be: "4/4", "3/4", "2/4", "C" or "C/". The time change applies to the measure where the user clicks via the <i>mouseClickExecute</i></li> <li>• "SET_METRONOME" sets the metronome for the whole piece. The first value in <i>argumentsOnExecute</i> contains the reference note duration (D1, D1_2, D1_4,...) the second value contains "TRUE" if the reference note is with augmentation dot ("FALSE" or empty otherwise), the third value indicates the number of reference notes in one minute. For example ["D1_4", "TRUE", "100"] sets a metronome with 100 dotted quarters in one minute. The metronome is set using the <i>execute</i> eventIn.</li> <li>• "DELETE" allows deleting any symbol, note, rest, alteration, label and annotation added by the user in the position where the user clicks via the <i>mouseExecute</i></li> </ul>
--	---

setFirstVisibleMeasure	
Method	setFirstVisibleMeasure
Description	It sets the first measure currently visible
Input parameters	Int number_of_measure
Output parameters	List ("FirstVisibleMeasure" number_of_measure): - Int number_of_measure

setHyperlinkEnable	
Method	setHyperlinkEnable
Description	When it is set to 1 hyperlinks are shown; when the user clicks on a link an <b>activatedLink</b> is generated
Input parameters	Int flag: 1 to activate Hyperlink 0 otherwise
Output parameters	List ("HyperlinkEnable" flag): - Int flag

<b>setPartLyrics</b>	
Method	setPartLyrics
Description	It is a list of symbols indicating for which part to view the lyrics and in which language (e.g. [ita eng ""]) to view lyrics for part 1 in Italian and for part 2 in English and no lyric for part 3)
Input parameters	List of symbols of language for part lyrics (the position represents the number of part, the symbol represents the language)
Output parameters	List ("PartLyrics" language_list): - List(symbols) language_list

<b>setPartShown</b>	
Method	setPartShown
Description	It is a list of integers indicating which parts have to be shown; the number is the position in the array of parts names; if <b>partShown</b> is empty all parts will be invisible (e.g. [ ] to view empty main score, [2] to view single part number 2, [1,3] view main score with parts 1 and 3, etc.).
Input parameters	List (int): list of numbers of the parts to show
Output parameters	List ("PartShown" parts_list): - List(int) parts_list

<b>setScoreOffset</b>	
Method	setScoreOffset
Description	It indicates the initial (or point 0) offset from the beginning of the score; it may be used to change page or move inside the score before starting it, or in pause etc. <b>scoreOffset</b> is indicated in seconds from the beginning of the score. <b>scoreOffset</b> can be used only if synchronisation information is provided or a metronome indication is present in the score.
Input parameters	Float time
Output parameters	List ("ScoreOffset" time): - Float time

<b>setChronometricPosition</b>	
Method	setChronometricPosition
Description	It sets the chronometric position (in millisecond) in the SMR object
Input parameters	Float position
Output parameters	List ("ChronometricPosition" position): - Float position

<b>setComputerViewParams</b>	
Method	setComputerViewParams
Description	It sets params for ComputerView
Input parameters	List (Top Bottom Left Right Staff): - Int Top - Int Bottom - Int Left - Int Right - Int Staff
Output parameters	List ("ComputerViewParams" param): - List (Int) param

<b>setMetricPosition</b>	
Method	setMetricPosition
Description	It sets the position depending on a metric (a metric is a time interval defined as a specific fraction of a minute) in the SMR object.
Input parameters	Int position
Output parameters	List (“MetricPosition” position): - Int position

<b>SetPrintViewParams</b>	
Method	SetPrintViewParams
Description	It sets params for PrintView
Input parameters	List (NPage Top Bottom Left Right Magnify Linelenght NStaffs NSystems Distance): - Int NPage - Int Top - Int Bottom - Int Left - Int Right - Float Magnify - Int Linelenght - Int NStaffs - Int NSystems - Int Distance
Output parameters	List (“PrintViewParams” param): - List param

<b>setSize</b>	
Method	setSize
Description	It expresses the width and height of the music score in pixels.
Input parameters	List (width height): - Int width - Int height
Output parameters	List (“Size” dim): - List (Int) dim

<b>setSpeed</b>	
Method	setSpeed
Description	It indicates how fast the score shall be played. It can be a positive tempo multiplier (>0), so a speed of 2 indicates the score plays twice as fast the tempo metronomic indication.
Input parameters	Float value
Output parameters	List (“Speed” value): - Float value

<b>setViewType</b>	
Method	setViewType
Description	It indicates the kind of view to be used (one of the <i>availableViewTypes</i> ).
Input parameters	Symbol value
Output parameters	List (“ViewType” value): - Symbol value

## 11.2 Inlets for get methods

The inlet for “Get” methods (the third from the left) receives as input only the name of the method (without “get” prefix). The method is executed by the Music Editor and it returns from the Return outlet (the second from the left) the name of the method (without the “get” word) and the parameters defined in the Output Parameters. If there are more than one parameter (for example, an array string), the Return outlet returns a sequence as:

<methodname without “get” prefix> <value1> <value2> <value3> ...

For example, to get the first visible measure, the message sent to the “Get” inlet is:

**FirstVisibleMeasure**

And the output of Return outlet is: **FirstVisibleMeasure 25**

<b>getArgumentOnExecute</b>	
Method	ArgumentOnExecute
Description	It returns arguments for the current commandOnExecute command.
Input parameters	None
Output parameters	List (“ArgumentOnExecute” args): - args: List of arguments (depending on the type of command)

<b>getAuthor</b>	
Method	getAuthor
Description	It returns the author of the SMR loaded
Input parameters	None
Output parameters	List (“Author” name): - Symbol name

<b>getAvailableCommands</b>	
Method	getAvailableCommands
Description	It gives a list of commands that can be performed on the score by the user when the user clicks on the score (e.g. ["ADD_LABEL", "ADD_TEXT_ANNOTATION", "DELETE"])
Input parameters	None
Output parameters	List (“AvailableCommands” commands): - List (symbol) commands

<b>getAvailableLabels</b>	
Method	getAvailableLabels
Description	It gives a list of strings with labels (e.g. ["A", "B", "SEGNO", "CODA"]).
Input parameters	None
Output parameters	List (“AvailableLabels” labels): - List (symbol) labels



<b>getAvailableLyrics</b>	
Method	getAvailableLyrics
Description	It gives a list of symbols where for each part there is the list of (e.g. ["eng ita" ita ""]) (this field may or may not be filled by the scene author, which is supposed to know the SMR content and thus languages that are available).
Input parameters	None
Output parameters	List ("AvailableLyrics" lyrics): - List (symbol) lyrics

<b>getAvailableViewTypes</b>	
Method	getAvailableViewTypes
Description	It gives an array of strings describing which view types are available for the score and for the decoder (e.g. [CWMN braille neumes]).
Input parameters	None
Output parameters	List ("AvailableViewTypes" views): - List (symbol) views

<b>getChronometricPosition</b>	
Method	getChronometricPosition
Description	It provides the present chronometric position (in millisecond) in the SMR object
Input parameters	None
Output parameters	List ("ChronometricPosition" position): - Float position

<b>getCommandOnExecute</b>	
Method	getCommandOnExecute
Description	It returns the current command set with the setCommandOnExecute method
Input parameters	None
Output parameters	List ("CommandOnExecute" command): - Symbol command
List of possible commands	<ul style="list-style-type: none"> <li>• "ADD_TEXT_ANNOTATION"</li> <li>• "ADD_LABEL"</li> <li>• "ADD_NOTE"</li> <li>• "ADD_REST"</li> <li>• "SET_ALTERATION"</li> <li>• "SET_DOTS"</li> <li>• "ADD_SYMBOL"</li> <li>• "ADD_MEASURE"</li> <li>• "DEL_MEASURE"</li> <li>• "CHANGE_CLEF"</li> <li>• "CHANGE_KEYSIGNATURE"</li> <li>• "CHANGE_TIME"</li> <li>• "SET_METRONOME"</li> <li>• "DELETE"</li> </ul>

<b>getFirstVisibleMeasure</b>	
Method	getFirstVisibleMeasure
Description	It returns the first measure currently visible
Input parameters	None
Output parameters	List (“FirstVisibleMeasure” number_of_measure): - Int number_of_measure

<b>getLastVisibleMeasure</b>	
Method	getLastVisibleMeasure
Description	It returns the last measure currently visible
Input parameters	None
Output parameters	List (“LastVisibleMeasure” number_of_measure): - Int number_of_measure

<b>getHighlightPosition</b>	
Method	getHighlightPosition
Description	It outputs the highlight position in local coordinates.
Input parameters	None
Output parameters	List (“HighlightPosition” position): - List (three floats) position

<b>getHyperlinkEnable</b>	
Method	getHyperlinkEnable
Description	When it is set to 1 hyperlinks are shown; when the user clicks on a link an <b>activatedLink</b> is generated
Input parameters	None
Output parameters	List (“HyperlinkEnable” active): - Int active: 1 if Hyperlink is activated, 0 otherwise

<b>getMetricPosition</b>	
Method	getMetricPosition
Description	It provides the position depending on a metric (a metric is a time interval defined as a specific fraction of a minute) in the SMR object.
Input parameters	None
Output parameters	List (“MetricPosition” position): - Int position:

<b>getMousePositionOnExecute</b>	
Method	getMousePositionOnExecute
Description	It is used to indicate the point where the user has clicked, the position will be considered when the <b>executeCommand</b> will be issued.
Input parameters	None
Output parameters	List (“MousePositionOnExecute” position): - List (three floats) position

<b>getClick</b>	
Method	getClick
Description	It is used to indicate the point coordinates in pixels (x,y) where the user has clicked.
Input parameters	None
Output parameters	List (“Click” position): - List (two ints) position
<b>getNumMeasures</b>	
Method	getNumMeasures
Description	It gives the number of measures in the score.
Input parameters	None
Output parameters	List (“NumMeasures” number_of_measure): - Int number_of_measure

<b>getPartLyrics</b>	
Method	getPartLyrics
Description	It is a list of symbols indicating for which part to view the lyrics and in which language (e.g. [ita eng ""] to view lyrics for part 1 in Italian and for part 2 in English and no lyric for part 3)
Input parameters	None
Output parameters	List (“PartLyrics” language_list): - List(symbols) language_list

<b>getPartNames</b>	
Method	getPartNames
Description	It gives a list of symbols with part names (instruments, e.g. [soprano baritone piano])
Input parameters	None
Output parameters	List (“PartNames” names): - List(symbols) names

<b>getPartShown</b>	
Method	getPartShown
Description	It is a list of integers indicating which parts are shown; the number is the position in the array of parts names; if <b>partShown</b> is empty all parts will be invisible (e.g. [1,3] main score shows parts 1 and 3, etc.).
Input parameters	None
Output parameters	List (“PartShown” parts_list): - List(int) parts_list

<b>getScoreOffset</b>	
Method	getScoreOffset
Description	It returns the initial (or point 0) offset from the beginning of the score; it may be used to change page or move inside the score before starting it, or in pause etc. <b>scoreOffset</b> is indicated in seconds from the beginning of the score. <b>scoreOffset</b> can be used only if synchronisation information is provided or a metronome indication is present in the score.
Input parameters	None
Output parameters	List (“ScoreOffset” time): - Float time

<b>getSize</b>	
Method	getSize
Description	It returns the width and height of the music score in pixels.
Input parameters	None
Output parameters	List (“Size” width height): - Int width - Int height

<b>getSpeed</b>	
Method	getSpeed
Description	It returns the present speed of the score. It can be a positive tempo multiplier (>0), so a speed of 2 indicates the score plays twice as fast the tempo metronomic indication.
Input parameters	None
Output parameters	List (“Speed” value): - Float value

<b>getSymPartName</b>	
Method	getSymPartName
Description	It returns a description of a Symbolic Score
Input parameters	None
Output parameters	List (“SymPartName” desc): - Symbol desc

<b>getTitle</b>	
Method	getTitle
Description	It returns the title of the SMR loaded
Output parameters	List (“Title” desc): - Symbol title

<b>getViewType</b>	
Method	getViewType
Description	It returns the view used (one of the <i>availableViewTypes</i> ).
Input parameters	None
Output parameters	List (“ViewType” type): - Symbol type

### 11.3 Inlets for other methods

The inlet for “Other” methods (the fifth from the left) receives as input the name of the method following (if required) by the parameter declared in the Input Parameters.

The method is executed by the Music Editor and, if the method has some parameters to return, it returns from the Return outlet (the second from the left) the name of the method and the parameters defined in the Output Parameters. If there are more than one parameters (for example an array string), the Return Outlet returns a sequence as:

<methodname> <value1> <value2> <value3> ...

For example, to load a file, the message sent to the “Other” inlet is:

### Load nomefile

And the output of Return outlet could be: **Load 1** (to confirm the correct file loading) or **Load 0** (if there is a problem to load the file)

Another example is the sequence of messages to execute a command to insert a note in the score (“Set” and “Other” inlets and Return outlet are involved):

step	Get	Set	Other	Return
1		“CommandOnExecute” ADD_NOTE		
2		“ArgumentOnExecute” D1_8 CLASSIC		
3			“ExecuteCommand”	
4				“ExecuteCommand” 1

In the first step it arrives at the “Set” Inlet the message to set the type of command to perform, then arrive also the command to set the parameters for the command itself. In the third step is sent the message to the “Other” inlet to execute the command previously set and at the end, from the Return Outlet, the message arrives with the confirmation of correct execution of the command.

ExecuteCommand	
Method	ExecuteCommand
Description	It is an input event indicating that the command set in commandOnExecute has to be performed.
Input parameters	None
Output parameters	List (“ExecuteCommand”flag): - Int flag: 1 if the command is correctly performed, 0 otherwise

GoBackward	
Method	GoBackward
Description	It shows previous score page
Input parameters	None
Output parameters	Symbol “GoBackward”

GoForward	
Method	GoForward
Description	It shows next score page
Input parameters	None
Output parameters	Symbol “GoForward”

<b>GoBottom</b>	
Method	GoBottom
Description	It shows the bottom of the score
Input parameters	None
Output parameters	Symbol “GoBottom”

<b>GoTop</b>	
Method	GoTop
Description	It shows the top of the score
Input parameters	None
Output parameters	Symbol “GoTop”

<b>GotoLabel</b>	
Method	GotoLabel
Description	It positions the score on the page containing the specified label (one of the <i>availableLabels</i> ).
Input parameters	Symbol label
Output parameters	List (“GotoLabel” label): - Symbol label

<b>GotoMeasure</b>	
Method	GotoMeasure
Description	positions the score on the page containing the specified measure
Input parameters	Int measure
Output parameters	List (“GotoMeasure” measure): - Int measure

<b>HighlightTimePosition</b>	
Method	HighlightTimePosition
Description	It highlights the time position indicated relative to the scoreOffset field
Input parameters	None
Output parameters	Symbol “HighlightTimePosition”

<b>Justify</b>	
Method	Justify
Description	It justifies (logarithmic or linear) the current score depending on entry parameters
Input parameters	List (fromMeasure toMeasure logarithmic): Int fromMeasure Int toMeasure Int logarithmic (1 is logarithmic, 0 is linear)
Output parameters	List (“Justify”, param): List(int) param

<b>Load</b>	
Method	Load

Description	Load a file from disk or from a URL
Input parameters	Symbol file – name or path of the file to load
Output parameters	List (“Load”flag): - Int flag: 1 if the file is correctly loaded, 0 otherwise

<b>Pause</b>	
Method	Pause
Description	It pauses the current file execution
Input parameters	None
Output parameters	Symbol “Pause”

<b>Play</b>	
Method	Play
Description	It plays the file previously loaded
Input parameters	None
Output parameters	Symbol “Play”

<b>PlaySync</b>	
Method	PlaySync
Description	It plays a synchronous file
Input parameters	None
Output parameters	Symbol “PlaySync”

<b>PlaySyncFromTo</b>	
Method	PlaySyncFromTo
Description	It executes an synchronous file from the measure x to the measure y
Input parameters	List(startMeasure endMeasure): -Int startMeasure -Int endMeasure
Output parameters	List (“PlaySyncFromTo”, param): List(int) param

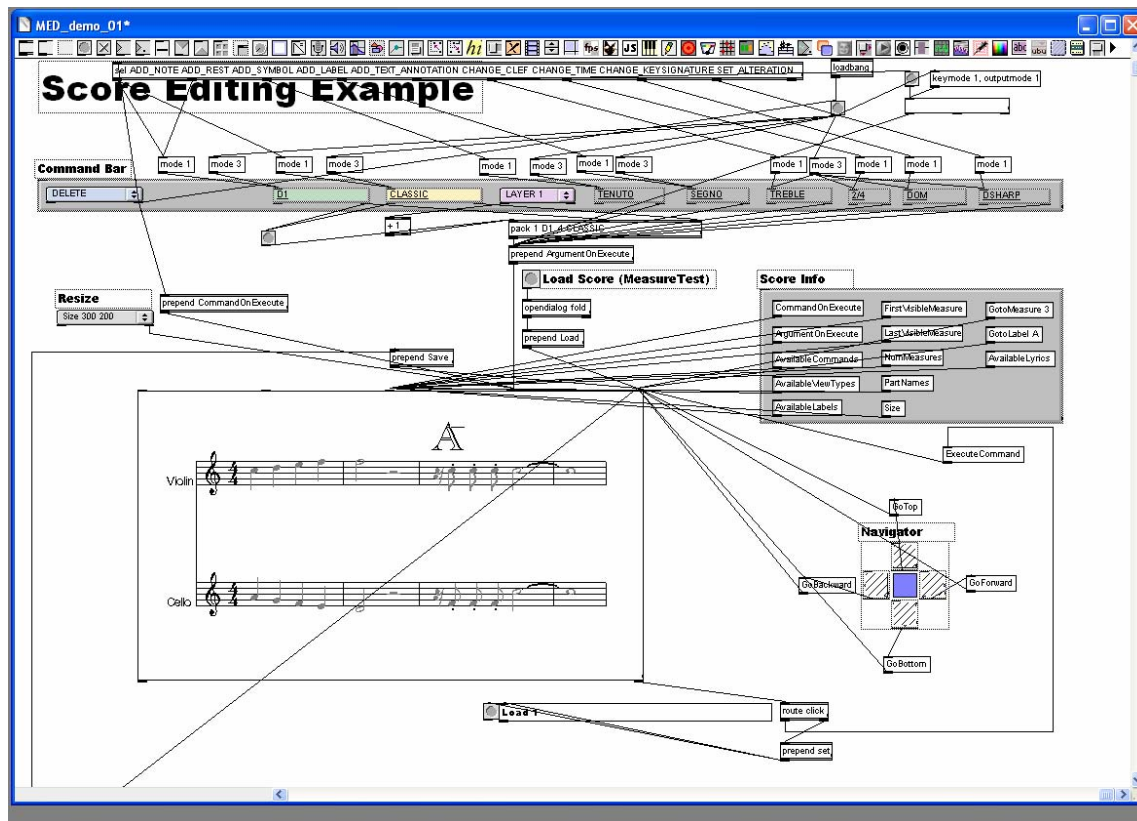
<b>Print</b>	
Method	Print
Description	Print the current score
Input parameters	None
Output parameters	Symbol “Print”

<b>Stop</b>	
Method	Stop
Description	It stops the execution of a file loaded
Input parameters	None
Output parameters	Symbol “Stop”

<b>Transpose</b>	
Method	Transpose
Description	It transposes notes in the score.
Input parameters	List (daBat aBat partnumber clef translation interval up numofstaff sharps adjust): -Int daBat – start measure to execute transposition -Int aBat – final measure to execute transposition -Int partnumber – number of the part to transpose -Int clef – number of the clef to change -Int translation – it point how much the note have to be tranlated -Int interval – type of interval -Int up – it points if the translation is up or down -Int numofstaff – it gives the capability to transpose using more then one staff -Int sharps -Int adjust
Output parameters	List (“Transpose”, param): List(int) param



## 11.4 Examples of MED in Max/MSP for Music General Controls



For a visual programmer in MAX, the model contained in the MED-SMR contains the music as modelled with a hierarchical organisation. A music score is made of parts and each part consists of a list of measures containing several voices. Each voice may contain events of various types: Notes, Chords made by notes, Rests, Refrains, Key Changes, Clef Changes. MED allows to access the SMR model by means of a set of functionalities.

For the MED, there are two groups of commands for accessing the SMR model and for its navigation. Each group of command can be accessed via a specific inlet and MED uses first outlet for sending information to other visual objects of the MAX visual environment.

The navigation/access to the SMR modelling information allows to realize patch programs in MAX to perform symbolic analysis of music and to read the SMR model that could be used to feed score-following algorithms.

A first command group is useful to recover information about score structure and events, such as the pitch of a specific note, the time signature of a measure or the number of parts inside the score. These commands have to be routed to the first inlet.

When a message is sent to MED through this inlet, MED takes this message as a getting information request.

The second group contains commands for setting the cursor position on a particular element of the score structure, such as moving cursor position on the third part of the score or at the beginning of the second measure of the first part. Once the cursor position is set, it is possible to retrieve element selected information using first inlet. These commands have to be routed to the second inlet.

On this basis it is possible to:

- get the number of elements of each structure's level;
- set the cursor position; in this case the cursor is only a position to start navigating into the SMR model;
- get measure information (metronome, barlines type, labels, key signature type, time signature data, clef type, etc.);
- get event type;
- get note info (pitch, duration, ties);
- get rest info (duration);
- get chord info (duration, pitches);
- get refrain info (type);
- get key change info (type);
- get clef change info (type).

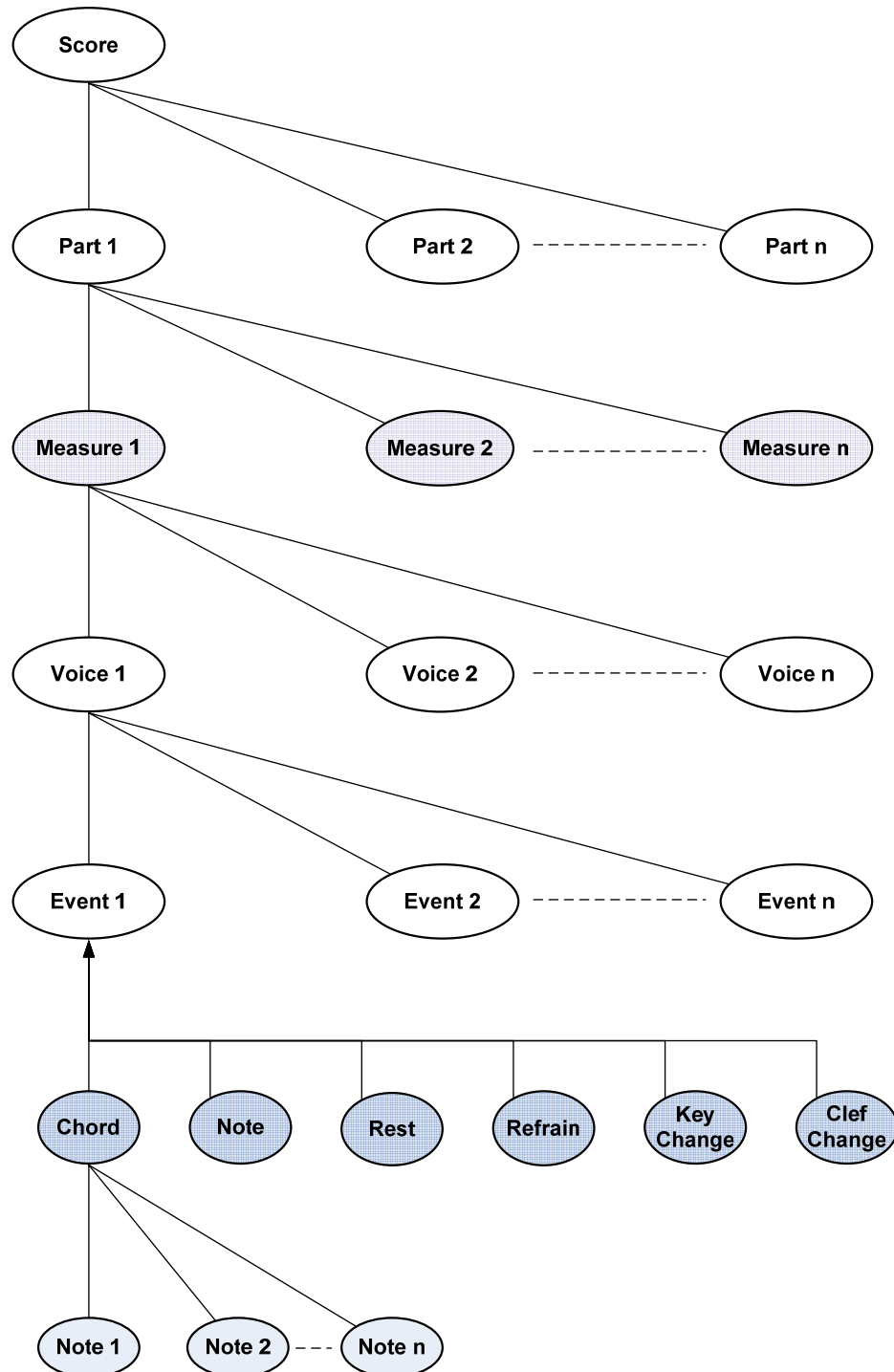
The following table shows an example of message sequence for getting the pitch of the fifth event (a note) on the second voice of the first measure in the third part of the score.

First Inlet	Second Inlet	First Outlet
	"Part 3"	
	"Measure 1"	
	"Voice 2"	
	"Event 5"	
"Pitch"		
		"Pitch G#2"

**Example to access the SMR model information**

## 12 MED: SMR Music Editor, Music Notation Access Support for MAX

The following is the schema of the structure of a score. Score is made by single parts (from one to n) and each part is made by voices. Each voice can contain one ore more events. There are various type of events: Note, Chord made by notes, MeasureChange, Rest, Refrain, KeyChange, ClefChange and TimeSignChange.



The following methods represent the interface provided by the Music Editor to the Score-Follower to interact with the score.



An example of access to the score viewed in the figure above can be the access to the duration of each note of lower voice in the “Violin” part.

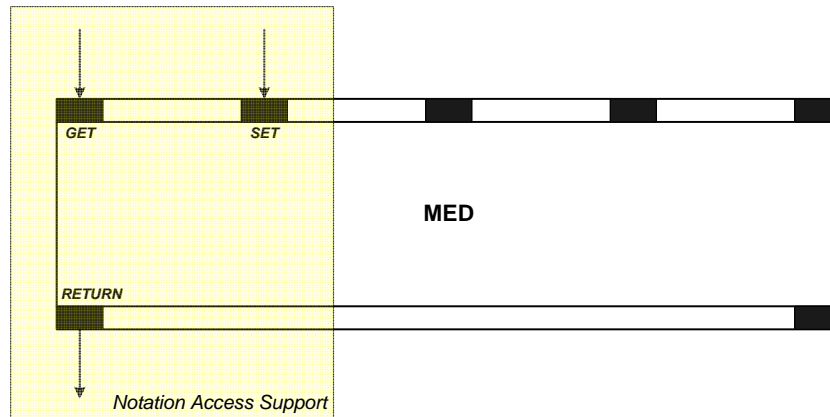
```
SetPart(3);
SetMeasure(2);
SetVoice(1);
for (i=1, i<getNumEvents(), i++)
{
    setCurrentEvent( i );
    event_type=getEventType();
    if (event_type==1) //the event is a note
    {
        currentNoteDuration=getDuration();
    }
}
```

Calling a setter method of a particular level causes cursor setting on the first element of lower levels, leaving upper levels unchanged (e.g. SetMeasure(2) moves the cursor on the first event of the first voice of measure 2). By default the score cursor is set on the first element of each level.

The following methods are available to access the Music Score. They can be grouped in two groups:

1. “Get” methods to recover information about score structure and events
2. “Set” methods to select a position inside the score structure

The groups are linked to a specific inlet and there is an outlet to send info from the Music Editor to the other objects. So, there is an inlet for each group and only one outlet as shown in the following figure:



## 12.1 Score navigation methods

getNumParts	
Method	getNumParts
Description	It returns the total parts number inside the score ordered bottom-up. There are at least one part in a score ordered from the lower to the higher.
Input parameters	None
Output parameters	List("NumParts" total_parts_number): -Int total_parts_number

setPart	
Method	setPart
Description	It selects the part inside the score
Input parameters	Int part_number
Output parameters	List("Part" flag): -Int flag : 0 if no error, -1 otherwise

getScoreName	
Method	getScoreName
Description	It returns the name of the score (if present)
Input parameters	None
Output parameters	List("ScoreName" score_name): Symbol score_name

getNumStaves	
Method	getNumStaves
Description	It returns the total staves number inside the part.
Input parameters	None
Output parameters	List("NumStaves" total_staves_number): -Int total_staves_number, -2 if error occurs

<b>setStaff</b>	
Method	setStaff
Description	It selects the staff inside the part
Input parameters	Int staff number
Output parameters	List(“Staff” flag): -Int flag: 0 if no error, -1 otherwise

<b>getNumMeasures</b>	
Method	getNumMeasures
Description	It returns the total measures number inside the part.
Input parameters	None
Output parameters	List(“NumMeasures” total_measures_number): -Int total_measures_number, -2 if error occurs

<b>setMeasure</b>	
Method	setMeasure
Description	It selects the measure inside the part
Input parameters	Int measure_number
Output parameters	List(“Measure” flag): -Int flag : 0 if no error, -1 otherwise

<b>getNumVoices</b>	
Method	getNumVoices
Description	It returns the total voices number inside the part. There are from 1 to 4 voices in a part ordered from the lower to the higher.
Input parameters	None
Output parameters	List(“NumVoices” total_voices_number): -Int total_voices_number, -3 if error occurs

<b>setVoice</b>	
Method	setVoice
Description	It selects the voice inside the part
Input parameters	Int voice_number
Output parameters	List(“Voice” flag): -Int flag : 0 if no error, -1 otherwise

<b>getPartName</b>	
Method	getPartName
Description	It returns the name of the part (if present)
Input parameters	None
Output parameters	List(“PartName” name): Symbol name






















<b>getNumEvents</b>	
Method	getNumEvents
Description	It returns the total events number inside the voice. There is atleast one event in a voice.
Input parameters	None
Output parameters	List(“NumEvents” total_events_number): -Int total_events_number, -4 if error occurs

<b>setEvent</b>	
Method	setEvent
Description	It selects an event inside the voice
Input parameters	Int event_number
Output parameters	List(“Event” flag): -Int flag : 0 if no error, -1 otherwise







<b>getEventType</b>																	
Method	getEventType																
Description	It returns the type of the event selected																
Input parameters	None																
Output parameters	List(“EventType” event_type): -Int event_type, -5 if error occurs <table border="1" data-bbox="667 1043 1145 1346"> <thead> <tr> <th>eventType</th><th>Int</th></tr> </thead> <tbody> <tr> <td>Label</td><td>0</td></tr> <tr> <td>Note</td><td>1</td></tr> <tr> <td>Rest</td><td>2</td></tr> <tr> <td>Chord</td><td>3</td></tr> <tr> <td>Refrain</td><td>4</td></tr> <tr> <td>KeyChange</td><td>5</td></tr> <tr> <td>ClefChange</td><td>6</td></tr> </tbody> </table>	eventType	Int	Label	0	Note	1	Rest	2	Chord	3	Refrain	4	KeyChange	5	ClefChange	6
eventType	Int																
Label	0																
Note	1																
Rest	2																
Chord	3																
Refrain	4																
KeyChange	5																
ClefChange	6																




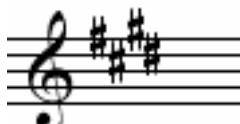








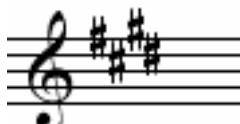








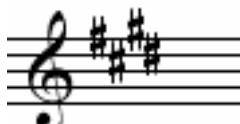





## 12.2 Measure Info






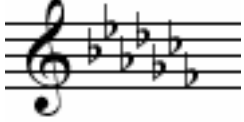
<b>getMetro</b>	
Method	getMetro
Description	It define the metronome speed of the current measure
Input parameters	None
Output parameters	List (”Metro”durationRef speed): - Int durationRef: symbolic positive note duration (whole note = 4096), -6 if error occurs - Int speed: beat per_minute, -6 if error occurs

<b>getBarType</b>																										
Method	getBarType																									
Description	It returns the type of the barline of the measure																									
Input parameters	None																									
Output parameters	List("BarType" bar_type): -Int bar_type, -6 if error occurs <table border="1"> <thead> <tr> <th>Bar_Type</th><th>Int</th><th>Example</th></tr> </thead> <tbody> <tr> <td>SINGLE</td><td>0</td><td></td></tr> <tr> <td>DOUBLE</td><td>1</td><td></td></tr> <tr> <td>END</td><td>2</td><td></td></tr> <tr> <td>END_REFRAIN</td><td>3</td><td></td></tr> <tr> <td>START_END_REFRAIN</td><td>4</td><td></td></tr> <tr> <td>START_REFRAIN</td><td>5</td><td></td></tr> <tr> <td>INVISIBLE</td><td>6</td><td></td></tr> </tbody> </table>		Bar_Type	Int	Example	SINGLE	0		DOUBLE	1		END	2		END_REFRAIN	3		START_END_REFRAIN	4		START_REFRAIN	5		INVISIBLE	6	
Bar_Type	Int	Example																								
SINGLE	0																									
DOUBLE	1																									
END	2																									
END_REFRAIN	3																									
START_END_REFRAIN	4																									
START_REFRAIN	5																									
INVISIBLE	6																									



<b>getLabel</b>																																			
Method	getLabel																																		
Description	It returns the label symbol																																		
Input parameters	None																																		
Output parameters	<p>List(“Label” label_sym):  - Symbol label_sym if no error, empty otherwise</p> <table> <tr> <th>Label</th><th>Symbol</th><th>Example</th></tr> <tr> <td>Coda</td><td>CODA</td><td></td></tr> <tr> <td>Segno</td><td>SEGNO</td><td></td></tr> <tr> <td>Da Capo</td><td>DC</td><td><b>D.C.</b></td></tr> <tr> <td>Da Capo al Fine</td><td>DCAF</td><td><b>D.C. al Fine</b></td></tr> <tr> <td>Dal Segno</td><td>DS</td><td><b>D.S.</b></td></tr> <tr> <td>Dal Segno al Fine</td><td>DSAF</td><td><b>D.S. al Fine</b></td></tr> <tr> <td>Da Capo al Segno</td><td>DCAS</td><td><b>D.C. al Segno</b></td></tr> <tr> <td>Dal Segno al Coda</td><td>DSAC</td><td><b>D.S. al Coda</b></td></tr> <tr> <td>Fine</td><td>FINE</td><td><b>Fine</b></td></tr> <tr> <td>Letter label</td><td><b>A...Z, 0...9</b></td><td><b>A</b></td></tr> </table>		Label	Symbol	Example	Coda	CODA		Segno	SEGNO		Da Capo	DC	<b>D.C.</b>	Da Capo al Fine	DCAF	<b>D.C. al Fine</b>	Dal Segno	DS	<b>D.S.</b>	Dal Segno al Fine	DSAF	<b>D.S. al Fine</b>	Da Capo al Segno	DCAS	<b>D.C. al Segno</b>	Dal Segno al Coda	DSAC	<b>D.S. al Coda</b>	Fine	FINE	<b>Fine</b>	Letter label	<b>A...Z, 0...9</b>	<b>A</b>
Label	Symbol	Example																																	
Coda	CODA																																		
Segno	SEGNO																																		
Da Capo	DC	<b>D.C.</b>																																	
Da Capo al Fine	DCAF	<b>D.C. al Fine</b>																																	
Dal Segno	DS	<b>D.S.</b>																																	
Dal Segno al Fine	DSAF	<b>D.S. al Fine</b>																																	
Da Capo al Segno	DCAS	<b>D.C. al Segno</b>																																	
Dal Segno al Coda	DSAC	<b>D.S. al Coda</b>																																	
Fine	FINE	<b>Fine</b>																																	
Letter label	<b>A...Z, 0...9</b>	<b>A</b>																																	

getKey																																																											
Method	getKey																																																										
Description	It returns the key signature type.																																																										
Input parameters	None																																																										
Output parameters	List(“Key” key_type): -Int key_type, -6 if error occurs																																																										
	<table><tr><th colspan="4">Key_Type</th><th>Example</th></tr><tr><th>Major</th><th>Int</th><th>Minor</th><th>Int</th><th></th></tr><tr><td>DOdM</td><td>7</td><td>LAdm</td><td>22</td><td></td></tr><tr><td>FAdM</td><td>6</td><td>REdm</td><td>21</td><td></td></tr><tr><td>SIM</td><td>5</td><td>SOLdm</td><td>20</td><td></td></tr><tr><td>MIM</td><td>4</td><td>DOdm</td><td>19</td><td></td></tr><tr><td>LAM</td><td>3</td><td>FAdm</td><td>18</td><td></td></tr><tr><td>REM</td><td>2</td><td>SIm</td><td>17</td><td></td></tr><tr><td>SOLM</td><td>1</td><td>MIm</td><td>16</td><td></td></tr><tr><td>DOM</td><td>0</td><td>Lam</td><td>15</td><td></td></tr><tr><td>FAM</td><td>8</td><td>REm</td><td>23</td><td></td></tr></table>				Key_Type				Example	Major	Int	Minor	Int		DOdM	7	LAdm	22		FAdM	6	REdm	21		SIM	5	SOLdm	20		MIM	4	DOdm	19		LAM	3	FAdm	18		REM	2	SIm	17		SOLM	1	MIm	16		DOM	0	Lam	15		FAM	8	REm	23	
Key_Type				Example																																																							
Major	Int	Minor	Int																																																								
DOdM	7	LAdm	22																																																								
FAdM	6	REdm	21																																																								
SIM	5	SOLdm	20																																																								
MIM	4	DOdm	19																																																								
LAM	3	FAdm	18																																																								
REM	2	SIm	17																																																								
SOLM	1	MIm	16																																																								
DOM	0	Lam	15																																																								
FAM	8	REm	23																																																								

		SIbM	9	SOLm	24		
		MIbM	10	Dom	25		
		LAbM	11	FAm	26		
		REbM	12	SIbm	27		
		SOLbM	13	MIbm	28		
		DObM	14	LAbm	29		






















getTimeSign	
Method	getTimeSign
Description	It returns the time signature of a measure
Input parameters	None
Output parameters	List ("TimeSign" numerator denominator): - Int numerator: the numerator part of the time signature, -6 if error occurs - Int denominator: the denominator part of the time signature, -6 if error occurs



getClef			
Method	getClef		
Description	It returns the clef signature type.		
Input parameters	None		
Output parameters	List(“Clef” clef_type): -Int clef_type, -6 if error occurs		
	</		

		ALTO	3		
		MEZZOSOPRANO	4		
		SOPRANO	5		
		TENOR	6		
		TENOR8	7		
		TREBLE	8		
		TREBLE8	9		
		8TREBLE	10		
		BASS8	11		
		8BASS	12		
		EMPTY(questo non è un cambio chiave)	13		
		PERCUSBOX	14		
		PERCUS2LINES	15		
		TAB	16		

### 12.3 Note Info































getPitch	
Method	getPitch
Description	It returns the pitch of note
Input parameters	None
Output parameters	List("Pitch" pitch_sym): Symbol pitch_sym (Format: XYZ; X is the char representing the basic pitch, Y is the optional sharp or flat, Z is the octave number – E.g. A#3, Bb4, E_1, C_3 = middle C) if no error, empty symbol otherwise

getDuration																	
Method	getDuration																
Description	It returns the symbolic positive note duration calculated depending on the duration of a whole note. (whole note duration = 4096)																
Input parameters	None																
Output parameters	List("Duration" dur): -Int dur, -6 if error occurs <table border="1" data-bbox="480 1014 1331 1912"> <thead> <tr> <th>Duration</th><th>Example</th></tr> </thead> <tbody> <tr> <td>8192</td><td></td></tr> <tr> <td>4096</td><td></td></tr> <tr> <td>2048</td><td></td></tr> <tr> <td>1024</td><td></td></tr> <tr> <td>512</td><td></td></tr> <tr> <td>256</td><td></td></tr> <tr> <td>128</td><td></td></tr> </tbody> </table>	Duration	Example	8192		4096		2048		1024		512		256		128	
Duration	Example																
8192																	
4096																	
2048																	
1024																	
512																	
256																	
128																	

		64		
		32		

<b>getIsTied</b>	
Method	getIsTied
Description	It returns 1 if the note is tied with the next one (with the same pitch).
Input parameters	None
Output parameters	List("IsTied" tie): -Int tie: 1 if the note is tied, 0 if not and, -6 if error occurs

## 12.4 Rest info

<b>getDuration</b>																							
Method	getDuration																						
Description	It returns the symbolic positive rest duration calculated depending on the duration of a whole rest. (whole rest duration = 4096)																						
Input parameters	None																						
Output parameters	<p>List(“Duration” dur): -Int dur, -6 if error occurs</p> <table> <tr> <th>Duration</th><th>Example</th></tr> <tr> <td>16384</td><td></td></tr> <tr> <td>8192</td><td></td></tr> <tr> <td>4096</td><td></td></tr> <tr> <td>2048</td><td></td></tr> <tr> <td>1024</td><td></td></tr> <tr> <td>512</td><td></td></tr> <tr> <td>256</td><td></td></tr> <tr> <td>128</td><td></td></tr> <tr> <td>64</td><td></td></tr> <tr> <td>32</td><td></td></tr> </table>	Duration	Example	16384		8192		4096		2048		1024		512		256		128		64		32	
Duration	Example																						
16384																							
8192																							
4096																							
2048																							
1024																							
512																							
256																							
128																							
64																							
32																							

## 12.5 Chord info

<b>getNumNotes</b>	
Method	getNumNotes
Description	It returns the total number of notes inside the chord.
Input parameters	None
Output parameters	List(“NumNotes” total_notes_number): -Int total_notes_number, -6 if error occurs

<b>setNote</b>	
Method	setNote
Description	It selects a note inside the chord.
Input parameters	Int note_number
Output parameters	List(“Note” flag): -Int flag : 0 if no error, -1 otherwise

## 12.6 Refrain info

getRefrain			
Method	getRefrain		
Description	It returns the type of refrain.		
Input parameters	None		
Output parameters	List(“Refrain” refrain_type): -Int refrain_type, -6 if error occurs		
	refrainType	Int	Example
	FIRSTTIME	7	1.
	SECONDTIME	8	2.
	THIRDTIME	9	3.
	ENDTIME	10	



## 12.7 KeyChange info

<b>getKeyChange</b>	
Method	getKeyChange
Description	It returns the key signature type.
Input parameters	None
Output parameters	List(“KeyChange” key_type): -Int key_type, -6 if error occurs(see 12.2)

## 12.8 ClefChange info

<b>getClefChange</b>	
Method	getClefChange
Description	It returns the clef signature type.
Input parameters	None
Output parameters	List(“ClefChange” clef_type): -Int clef_type, -6 if error occurs(see 12.2)

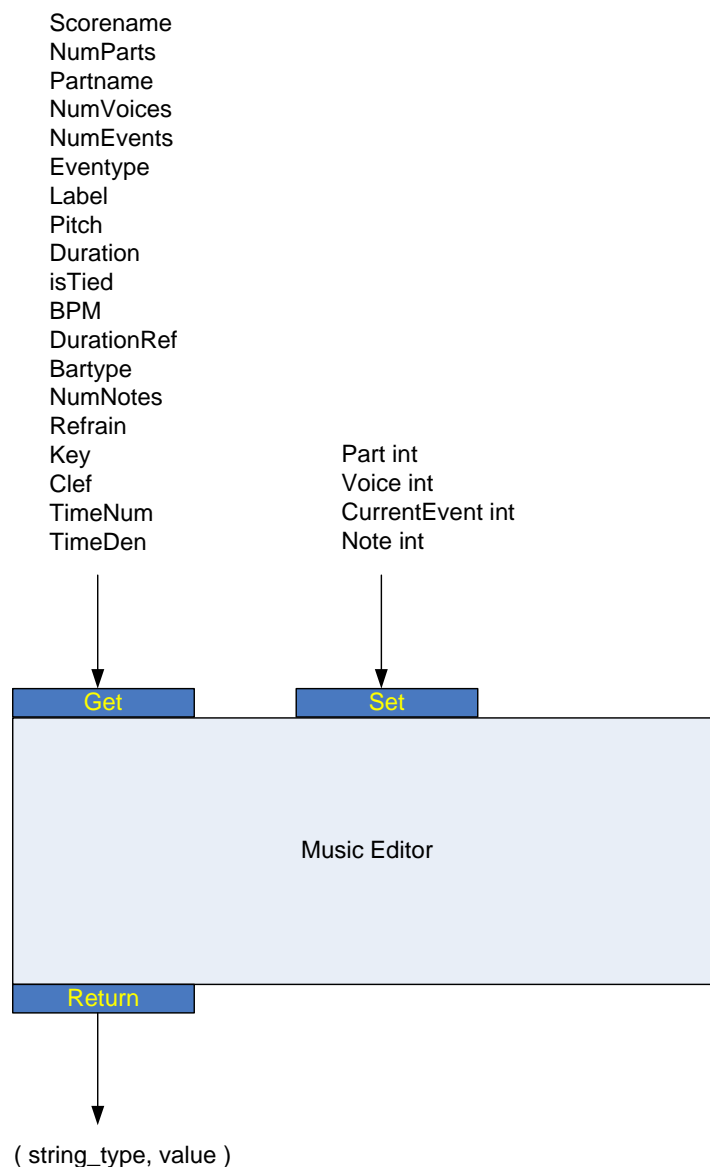
## 12.9 Error codes description

The following table shows the correspondence between error code number and their description.

<b>Description</b>	<b>Code (Int)</b>
Out of bounds	-1
Part not set	-2
Measure not set	-3
Voice not set	-4
Event not set	-5
Wrong type event	-6

## 12.10 MED in Max/MSP for Music Editing

The Music Editor Max module provides two specific inlets and one specific outlet, available for score exploring. The first inlet receives string commands corresponding to each 'get' methods and the second receives string commands corresponding to the 'set' methods previously described. The outlet returns a couple containing the name of the command sent and a value that can be also an error code if fault occurred.



The following command sequence is an example of interaction with Music Editor using the previously methods. The example score is made of 3 parts containing each a single voice, there are 10 figures (only notes and rests) per voice.

The table below presents the sequence of command to recover the duration of a note; the steps are explained to give an idea of the sequence progress. In the first step it is selected the part 3 of the score, and Return Outlet sends the method name with number of the part. Then in the step 3, it is selected the voice two (of the part 3 previously set). But now Return Outlet returns the name of the method with the number -1 which

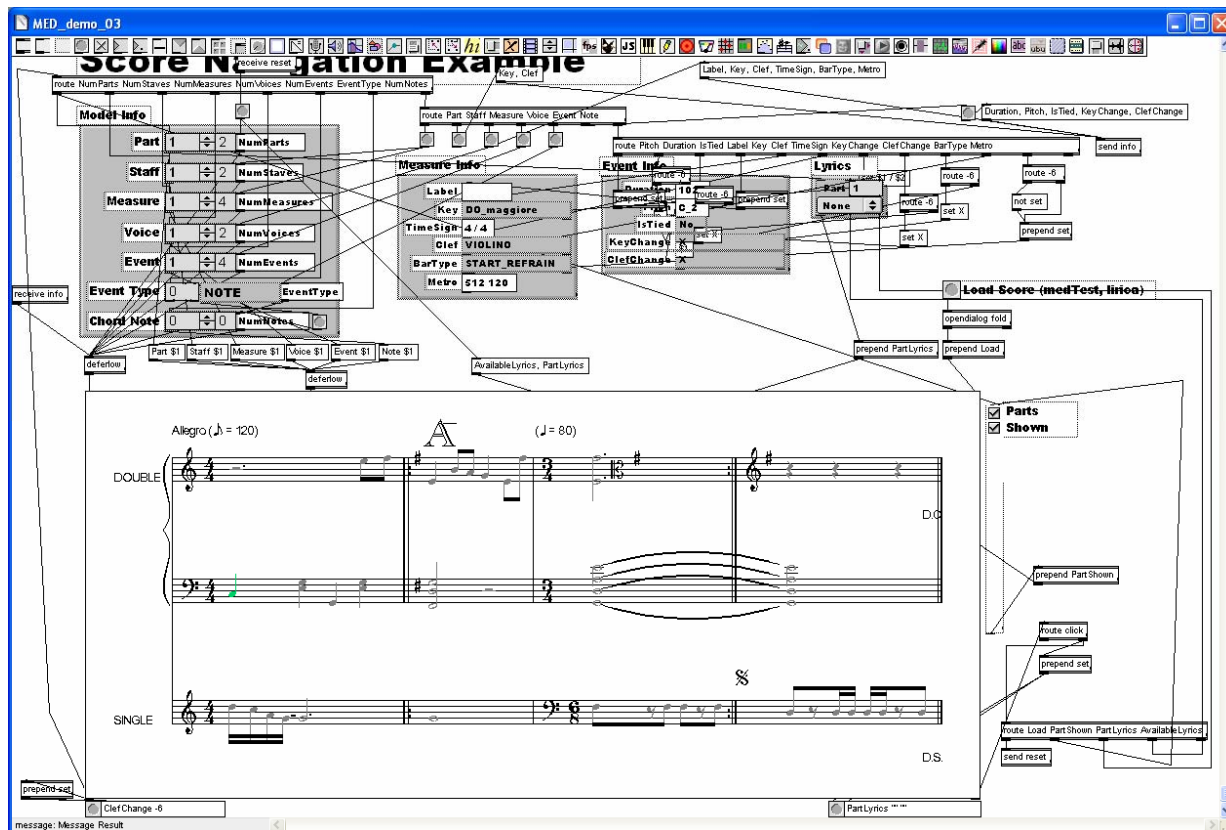
represents the presence of the error “Out of bounds” because there isn’t the voice number 2 in the part 3. In the step 5 the voice 1 is set and now the return value is correct.

The sequence continues with the step 7 where the command setCurrentEvent selects the event number 5 in the voice 1. The event type is get calling the Eventype method in the step 9. The return value from the Return outlet is “Eventype 1”. Looking at the event type code, the number 1 corresponds to the Note type.

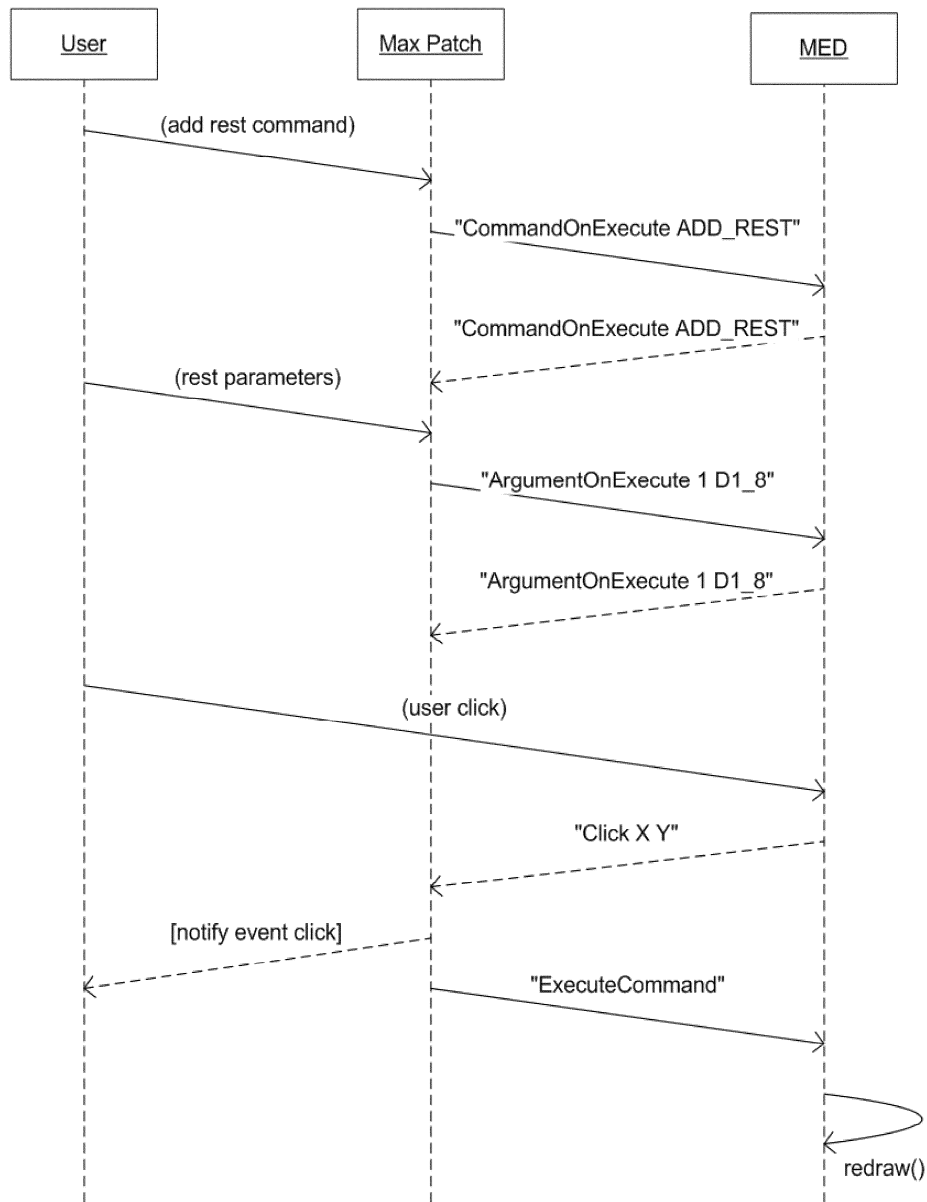
Then in step 11, the method Label is called to the current event (Note); the return value is an error (the number -5) because the note doesn’t have a label. In the end, the last method called is Duration which returns the duration of the note (step 14).

step	Get	Set	Return
1		“Part” 3	
2			“Part” 3
3		“Voice” 2	
4			“Voice” -1
5		“Voice” 1	
6			“Voice” 1
7		“CurrentEvent” 5	
8			“CurrentEvent” 5
9	“Eventype”		
10			“Eventype” 1
11	“Label”		
12			“Label” -5
13	“Duration”		
14			“Duration” 256
15			
16			
17			

## 12.11 Examples of MED in Max/MSP for Music Notation Access



MED has an interactive way of editing score symbols. It provides a set of user selectable commands (e.g., adding notes, rests, accidentals, deleting a symbol) with relative parameters. Once a programmer has specified a command to execute (e.g., add a quarter note), it is possible to set up a specific user interface in terms of coded MAX patch to interact with the MED by means of the mouse click (see the following figure). In the figure, the sequence diagram depicting the protocol established by the user, Max/MSP and the MED SMR model is reported. In this view, the user interacts with the MAX patch to send a command to the MED (for example the command to add a rest “CommandOnExecute ADD\_REST” of 1/8 of duration on voice 1 of the part selected by the user interaction). Once this command has reached the MED, it waits for a mouse click on the MED SMR visual rendering. This action provokes the sending of a message from the MED to the MAX environment including the relative coordinates X,Y of the clicked position. This event in Max/MSP could be used to start a large range of different actions (such as activation of play, or, etc.), while in this case the effective visualization of the rest with the sending of “Execute Command” to the MED is performed. This, in turn realizes in MED by the redraw of the visual representation on the screen.



**Sequence diagram User-MAX-MED**

### 13 MEX: Music Execution Service for Cooperative Work for MAX

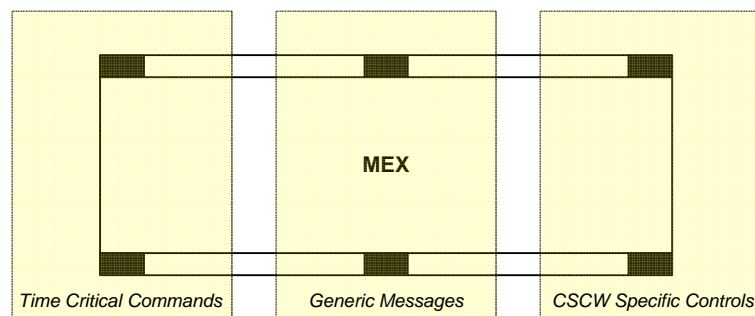
Music Execution Service provides all cooperative function available for tools working inside a cooperative lesson (e.g. metronome, gesture and posture tool, multimedia rendering tools, sensors...).

The service is implemented as Max External Object and only one copy of this object can be included inside a cooperative Lesson. To use this service is mandatory to create an object inside Max Designer Interface.

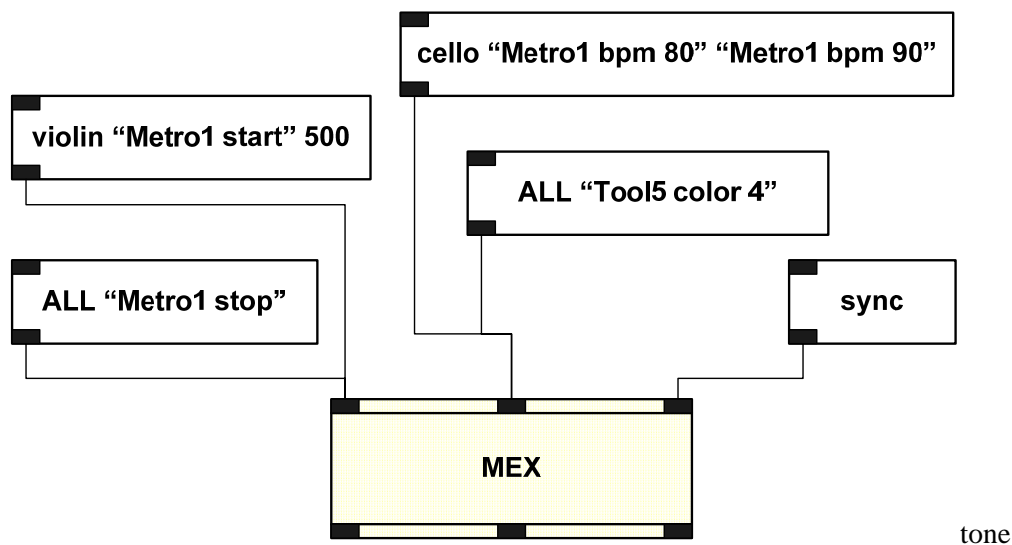
The Object has Inlets used to receive information from other Max Object and Outlets used to send information to other objects. Inlets are the way to exchange messages from Max to the P2P network; Outlets are the way to exchange messages from P2P Network to Max. MEX external has three different communications “channels” divided by the service kind they offer:

- Time Critical Commands (left inlet/outlet): commands that peers have to execute at the same time, such as starting a distributed tool.
- Generic Messages (middle inlet/outlet): every message that needs not to be strictly synchronised, like parameter setting.
- CSCW Specific Controls (right inlet/outlet): work group management information.

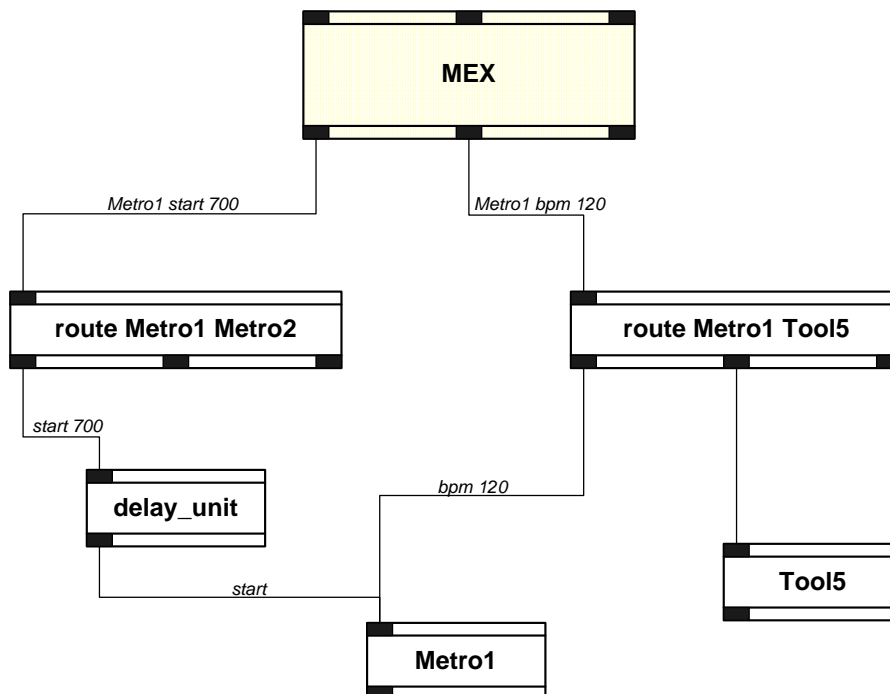
The following figure shows MEX External appearance in Max/MSP Environment.



An example of possible connection with the Music Execution Service Inlets. On the left Inlet two commands for distributed start/stop of a metronome (Metro1); on the middle one a message for Metro1 bpm change and a generic message prototype; on the right one a message of synchronisation request:



The following figure shows an example of the execution of a start command in a cooperative lesson. MEX receives a command from the P2P Network and pulls it out from the first outlet (in this case “Metro1 start 700”). The route object provides the correct metronome selection using the “Metro1” word. The value 700 is used by a delay unit that sends out a “start” command after 700 (milliseconds).



The same kind of logic (routing object) is used for the delivery of “Metro1 bpm 120”. The message “bpm 120” set the new metronome frequency. In these examples we have used the “route” object of Max/MSP to parse the command and forward the correct message to the correct tool. It is only an example; it is possible using some other objects to have the same behaviour.

### 13.1 Time Critical Commands

Inlet – Send Command	
Method	Inlet – Send Command
Description	Sends a command in a cooperative way specifying the delay time to wait before starting, the command will be executed at the same time. Max/MSP will provide logic for scheduling commands.
Input parameters	List(role command delay): Symbol role: One of the roles defined inside the lesson. If role=ALL the message is for all roles in the lesson; if role=SELF the message is for all users having role equal to the sender role. Symbol command: The name of the command to execute Int delay (optional): The delay to wait before start execution (milliseconds)
Sample Message	Cello “Metro start” 500

<b>Outlet - Send Command</b>	
Method	Outlet - Send Command
Description	It sends out the command coming from the P2P network. Max/MSP will provide a logic for scheduling and routing the command to the correct tool and/or role.
Output parameters	List(command delay): Symbol Command: The name of the command to execute Int delay(optional): The delay to wait before start execution (milliseconds)
Sample Message	“Metro start” 500

### 13.2 Generic Messages

<b>Inlet- Send Message</b>	
Method	Inlet- SendMessage
Description	Send a generic message in a cooperative way to a specified role
Input parameters	List(role command undo_cmd): Symbol role: One of the role defined inside the lesson. if role=ALL the message is for all roles in the lesson; if role=SELF the message is for all users having role equal to the sender role. Symbol command: It contains the message to deliver. The message is parsed by the specific tool. Symbol undo_cmd (optional): It contains the message to revoke the effect of command. It is used by CSCW Layer logic for keeping the stream of commands exact the same on each peer.
Sample Message	Bass “Metro bpm 120” “Metro bpm 90”

<b>Outlet – Receive Message</b>	
Method	Outlet - ReceiveMessage
Description	It sends out the message coming from the P2P network. Max/MSP will provide a logic for routing the command to the correct tool and/or role.
Output parameters	Symbols List command: It contains the message to deliver. The message is parsed by the specific tool.
Sample Message	Bass Metro bpm 100 (if receiver role is ALL) Metro bpm 100 (otherwise)



### 13.3 CSCW Specific Controls

Outlet - WorkGroupStatus	
Method	Outlet - WorkGroupStatus
Description	Monitors the cooperative session
Output parameters	List members: the list of members in the specified workgroup, empty if the user is not joined
Sample Message	

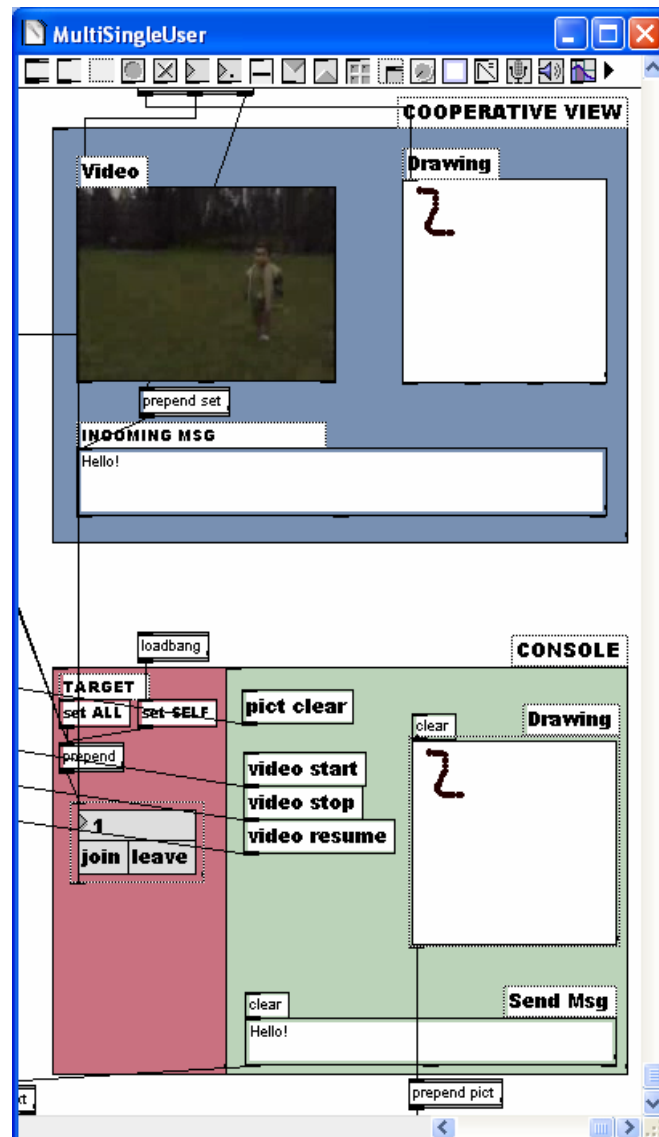
Inlet - Join	
Method	Inlet - Join
Description	Starts a cooperative session
Input parameters	List("join" user_name group_name role_name): -Symbol user_name: the name that the user choose on the peer network -Symbol group_name: the name of a workgroup to join in. -Symbol role_name: the name of the role chosen inside workgroup, ALL means every role
Sample Message	join jack lesson2 violin

Inlet - Leave	
Method	Inlet - Leave
Description	Ends a cooperative session
Input parameters	Symbol "leave"
Sample Message	leave

Inlet - Sync	
Method	Inlet - Sync
Description	Sets the time interval between two consecutive synchronisations
Input parameters	List("sync" interval): - Int Interval (optional): time interval in milliseconds, if absent one sync round is performed immediately
Sample Message	sync 1000

### 13.4 Examples of MEX for Cooperative Work among MAX Tools

The following image shows an example of cooperative interaction.



The image is divided in two boxes: the upper one represents the cooperative view of the lesson that each student connected can see. The second box shows the Console view of the lesson where the student can manage the lesson execution, interacting with this and sending messages to the other students connected.

In the Cooperative view the student can see for example a video played synchronously in all the peer connected. The “Drawing” small box shows in real time what all the students connected are drawing. Finally the “Incoming Message” box the students receive message from the other students.

In the Console view the user can send messages to the other peers and interact with the lesson. Pressing the Join/Leave button he can start or stop to work in the cooperative session joining a group or leaving it and selecting a role (for example violin or cello). The number in the little box represents the total number of peers connected in the same group.

Using the “Drawing” box the user can draw freely lines using the mouse as a pen. The drawing is sent in real time to the other peers in the group. Pressing the “clear” button all signs in his “Drawing” box in the Console view are deleted and the user can start to draw in a cleared area. Pressing the “Pict clear” button all the drawing in the “Drawing” box in the “cooperative View” of all the peers in the group are cleared.

As a teacher, the Console view has a number of commands reserved to him. For example using the “Video start”, “Video Stop”, “Video Resume” the teacher can simply interact with the video playing of all the peers connected in the group. Also he can write a text in the “Send Msg” box and send it to all the peers or only to one selected user.

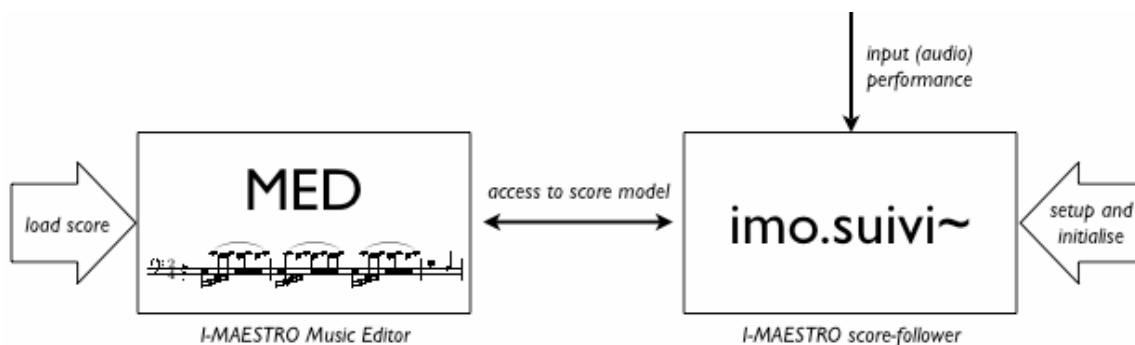
## 14 Score-following integration model (IRCAM)

The score model of the MED (constructed by loading a score file) is accessed in real-time by the score-follower in order to construct an internal representation of the score for following.

The score-follower constructs and updates in real-time an internal representation corresponding to a segment of the score around the current position of the performance input starting from an initial position. This internal representation corresponds to a Hidden Markov Model (HMM) used by the score-following algorithm.

The score-follower has to be provided with the following parameters before starting following:

- the voice to follow (in case of an SMR display of multiple voices)
- the initial position to start from
- a tempo factor (optional parameter that can lead to better performance of the follower just after the start in the case that the performance tempo derives very much from the tempo indicated in the score)

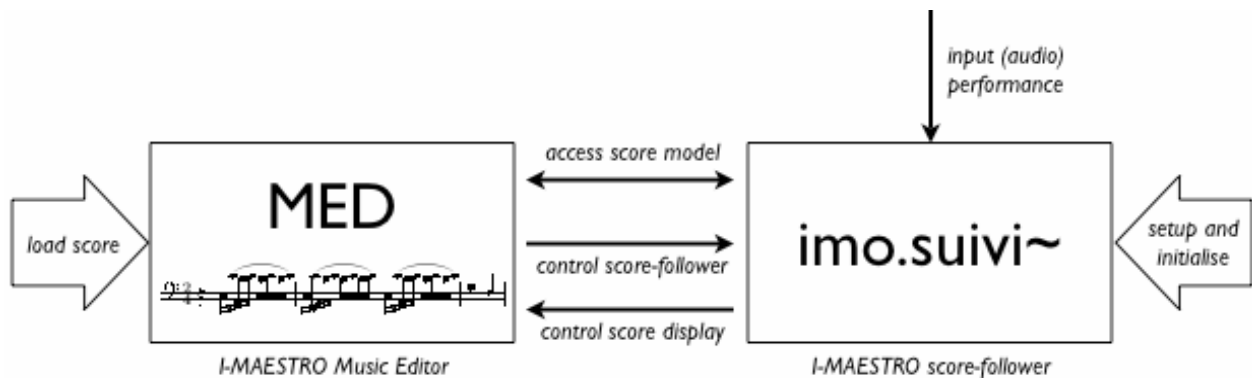


**Data and control flow between the MED and the score-follower (imo.suivi~)**

## 14.1 Mutual position control between the MED and the score-follower

An obvious user requirement is the control of the starting position and voice (in case of an SMR display of multiple voices) of the score-follower from the score display by a direct graphical interaction with the Music Editor.

Further more, its often desirable that the score-follower controls the score display in order to make the display of the score, following the performers interpretation. This feature is often referred to by “automatic page turning” or “automatic page scrolling”. (Note that two cases have to be distinguished here: 1. the rendering of the score for a third person – not requiring an anticipation of the displayed position, and 2. the rendering of the score for the performer himself – requiring a complex anticipation of the score position.)



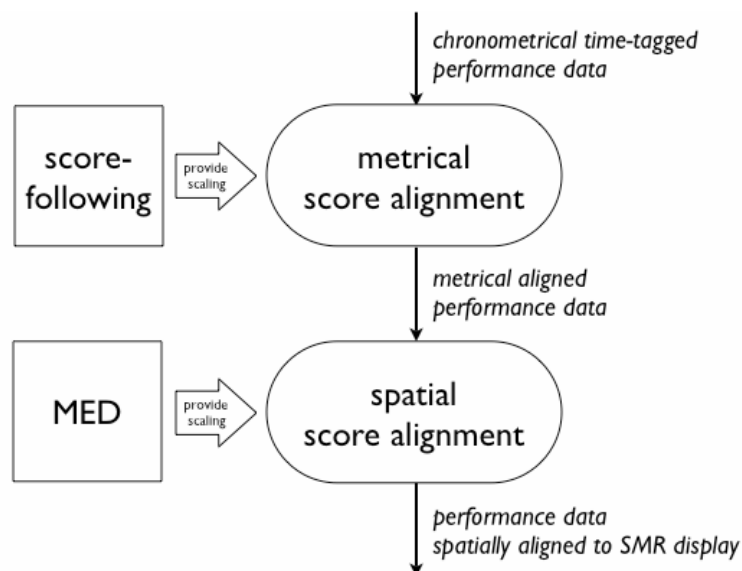
**Data and control flow including the mutual position control between MED score-follower**

## 14.2 Aligned rendering of performance data

Multiple user scenarios of the I-MAESTRO tools include the rendering of performance data sets from sensors (e.g. bow strokes) and audio extractors (e.g. loudness or pitch) spatially aligned to the display of the SMR corresponding to the given performance. The performance data sets are modelled as streams of events with time tags (see below).

In general, real-time and off-line rendering can be distinguished. Both cases require an alignment in two steps:

1. alignment of the performance chronometric data set to the metric of the score (provided by the score-following/score-alignment)
2. alignment of the metric of the score to the spatial score rendering (provided by the MED)



**Visual alignment of performance data to a given score in two steps**

## 15 Models for non-symbolic performance data (IRCAM)

As mentioned above, multiple user scenarios of the I-MAESTRO tools include the rendering of performance data sets from sensors (e.g. bow strokes) and audio extractors (e.g. loudness or pitch).

The I-MAESTRO developments for Sensor Support and Practice Training Paradigm Support are based on the SDIF, Sound Description Interchange Format, and extensions of FTM library developed in the framework of the I-MAESTRO project for the handling and representation of SDIF files. SDIF is an established standard for the well-defined and extensible interchange of a variety of sound descriptions including spectral, sinusoidal, time-domain, and higher-level models and applies also to sensor and motion capture data. SDIF consists of a basic data format framework and an extensible set of standard sound descriptions. The SDIF standard has been created in collaboration by IRCAM, CNMAT, and IUA-UPF. A complete specification of the format and documentation of the IRCAM SDIF library and related tools can be downloaded from the IRCAM SDIF home page<sup>1</sup>.

FTM is a shared library for Max/MSP providing a small and optimized real-time object system and a set of basic services to be used within Max/MSP externals. The latest FTM release including the documentation can be downloaded on the FTM home page<sup>2</sup>.

In the FTM library, SDIF files are handled by two specific classes:

- *track*, sequence of time-tagged values or objects
- *fmat*, matrix of floating point values

In general, an SDIF file is represented by one or multiple *track* objects containing *fmat* objects. A set of Max/MSP external modules allowing to access to SDIF data stored in FTM objects is provided by the FTM library.

The visualisation and editing of SDIF data sets within Max/MSP is handled by graphical Max/MSP modules accessing the FTM SDIF data representations. These modules handle also the alignment to the I-MAESTRO SMR display as described above.

<sup>1</sup> SDIF home page at IRCAM: <http://www.ircam.fr/sdif/>

<sup>2</sup> FTM home page at IRCAM: <http://www.ircam.fr/ftm/>

## 16 Models for collaborative interactive audio processing (IRCAM)

WP 5 contains several tools to be inserted into the framework of collaborative work throughout the project although most of the pedagogical paradigms are still to be defined.

The pedagogical field most advanced in terms of collaborative work with technology is that of innovative pedagogy working with Creative Interfaces and sound synthesis as well as sound transformations.

Here the challenge is to provide frameworks allowing collaborative work based on pertinent representations and open protocols.

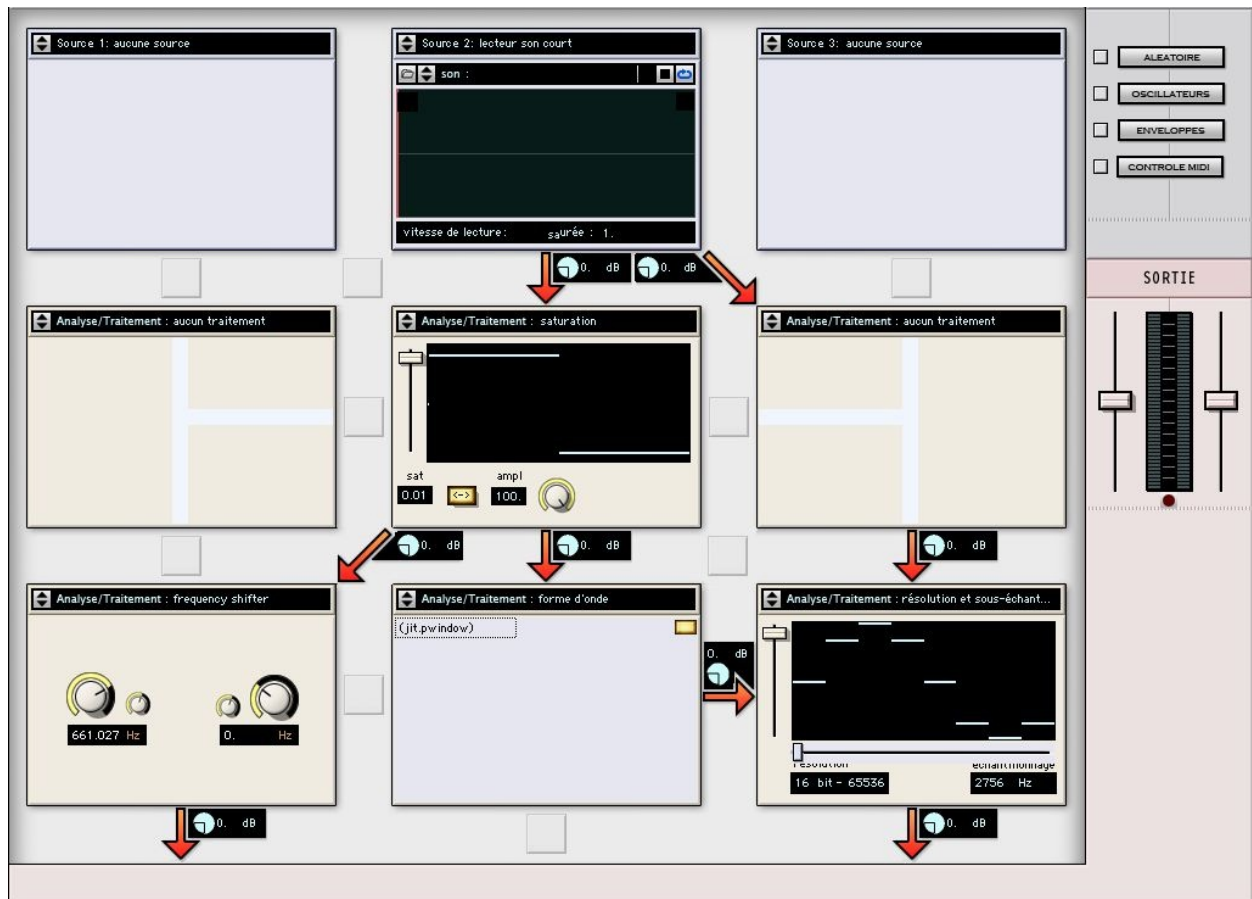
### 16.1 Simplified modular interactive audio processing framework

A first version of a simplified modular audio processing framework has been developed in the framework of the *MusicLabs II* project in collaboration between IRCAM and the French Ministry of Education. The basic design has been elaborated in vivid interaction with a group of selected music teachers teaching classes in secondary school (pupils' age from 12 to 18 years).

In the framework of the I-MAESTRO project the tool has been extended and will be further adapted to the tools in the I-MAESTRO system.

The tool supports pedagogical work with the following objectives;

- Comprehension of and experimentation with basic notions of audio transformation and synthesis techniques
- Musical control of audio processing by gestures (sensor and motion capture data)
- Sonification of gestures (based on sensors and motion capture data)
- Collaborative creation of audio processing instruments
- Collaborative performance of audio processing instruments



**Screenshot of the simplified modular interactive audio processing framework**

The finalised framework has the following basic features:

- Simplified selection and routing of audio processing modules in a 3 x 3 matrix including complete saving and recall (persistence) of a given selection, routing and parameterisation.
- Access to a wide range of audio processing modules including VST plug-ins (see below)
- Easy parameterisation of the modules via intuitive representations (display and visualisation)
- Possibility of modulation by inbuilt unit generators as well as external controllers such as gestural interfaces

For specifically supporting collaborative work the following features are envisaged:

- Synchronisation of the module selection, routing and parameterisation of two or more frameworks over the network (P2P)
- Synchronisation of modulation sources and rhythmic aspects over the network (P2P, while one client can be assigned as synchronisation master)
- Support of multiple control sources and interfaces also over network for collaborative playing

For the final version of the tool, the following inbuilt processing modules are planned.

Inbuilt sound synthesis modules:

Name/description	Parameters	Inbuilt interface/display
wave-form generator	choice of wave form frequency pulse width	single wave form display sliders/knobs
soundfile player	playing speed	wave form display sliders/knobs
granulator	choice of source sound duration of grains (min/max) reading speed (min/max)	wave form display with highlighted grain segments
additive synthesis	frequency levels of harmonics phases of harmonics (opt.)	spectral (harmonics) display wave-form display
simple FM synthesis	FM carrier/modulator routing carrier frequencies frequency ratios modulation factors	routing display sliders/knobs

Inbuilt sound transformation modules:

Name/description	Parameters	Inbuilt interface/display
Granulation	duration of grains (min/max) reading speed (min/max)	wave form display with highlighted grain segments
distortion by down-sampling	quantization (in bits)	quantized sine table
distortion by clipping	amplification factor	clipped wave-form
Harmoniser	transposition (in cent)	slider/knob
frequency Shifter	frequency shift (in Hz)	slider/knob
ring modulator	modulation freq. (in Hz)	slider/knob
source-filter cross-synthesis	choice of source	-
generalised cross-synthesis	choice of source mixing factors	2D mixing controller
spectral compressor/exciter	threshold ratio gain	level translation curve sliders/knobs
fixed recursive delay line	delay time feedback level	sliders/knobs
variable delay line (incl. chorus/flanger)	modulation depth modulation frequency	sliders/knobs
Filter	filter type (LP, HP, BP etc.) filter parameters	sliders/knobs frequency responds
Reverb	reverb time	schematised room
Doppler effect	max distance source speed	



Inbuilt sound analysis/visualisation modules:

Name/description	Display
wave form display	wave form
Spectroscope	frequency bands
Sonogram	scrolling sonogram
pitch detection	approx. note in piano notation

## 16.2 Synthesis control and synchronisation network protocol (IRCAM)

Max/MSP supports the Open Sound Control (OSC) protocol. Open Sound Control is a protocol for communication among computers, sound synthesizers, and other multimedia devices that is optimized for modern networking technology and has been used in many application areas. It is highly compatible with the Max/MSP programming paradigm.

OSC is an open standard that has been widely adapted by and computer music community and the audio software industry. It claims the following features:

- Open-ended, dynamic, URL-style symbolic naming scheme
- Numeric and symbolic arguments to messages
- Pattern matching language to specify multiple targets of a single message
- High resolution time tags
- "Bundles" of messages whose effects must occur simultaneously
- Query system to dynamically find out the capabilities of an OSC server and get documentation

A complete specification of the OSC protocol and the documentation of the IRCAM SDIF library and related tools can be downloaded from the OSC home page<sup>3</sup>.

<sup>3</sup> OSC home page: <http://www.cnmat.berkeley.edu/OpenSoundControl/>

## 17 Acronyms

The following are some abbreviations in common use:

ADL	Advanced Distributed Learning
CWS	Cooperative Work Service
GUI	Graphical User Interface is a method of interacting with a computer through a metaphor of direct manipulation of graphical images and widgets in addition to text.
IEC	The IEC is a similarly international organisation that "prepares and publishes international standards for all electrical, electronic and related technologies."
IEEE LTSC	Within the IEEE, the Learning Technology Standards Committee (LTSC) is chartered by the IEEE Computer Society Standards Activity Board to "develop accredited technical standards, recommended practices, and guides for learning technology"
IMS	"Instructional Management Systems (IMS) project", also sometimes referred to as "Global Learning Consortium, Inc.", IMS/GLC. The IMS Global Learning Consortium, Inc. (IMS) develops and promotes the adoption of open technical specifications for interoperable learning technology"
ISO	The International Standard Organisation is a standardization body that is recognised internationally, and was established under the auspices of the United Nations
LMS	Learning Management System
LOM	Learning Object Metadata" standard (IEEE 1484.12.1-2002)
OSC	Open Sound Control
RTE	Run Time Environment
SCO	Sharable Content Object
SCORM <sup>TM</sup>	Sharable Content Object Reference Model
URI	Uniform Resource Identifier, is a short string that identify resources in the web: documents, images, downloadable files, services, electronic mailboxes, and other resources.

## 18 Bibliography

- Advanced Distributed Learning (ADL), Sharable Content Object Reference Model (SCORM) Content Aggregation Model, Version 1.3.2, 2006
- <http://www.adlnet.gov/downloads/index.cfm?event=main.listing&categoryId=53>

### Overview:

- <http://www.adlnet.gov/technologies/SCORM/index.cfm>

### MPEG-SMR:

- B. W. Pennycook, “Computer-Music Interfaces: A Survey”, ACM Computing Surveys, June 1985, 17(2):267-289.
- CANTATE project, Deliverable 3.3: Report on SMDL evaluation, WP3, 1994. <http://projects.fnb.nl>
- Capella, 2005, CAPXML: <http://www.whc.de/capella.cfm>
- CAPXML: <http://www.whc.de/capella.cfm>
- CUIDADO: Processing of Music and Mpeg7: <http://www.ircam.fr/cuidad/>
- D. A. Byrd, Music Notation by Computer, Department of Computer Science, UMI, Dissertation Service, Indiana University, USA, <http://www.umi.com>, 1984
- D. Blostein, and H. S. Baird, “A Critical Survey of Music Image Analysis” in Structured Document Image Analysis, (H. S. Baird and H. Bunke and K. Yamamoto, eds.), Springer Verlag, NewYork, USA, 1992, pp. 405-434.
- D. Blostein, and L. Haken, “Justification of Printed Music”, Communications of the ACM, March 1991, 34(3):88-99.
- D. Crombie, D. Fuschi, N. Mitolo, P. Nesi, K. Ng, , and B. Ong, Bringing Music Industry into the Interactive Multimedia Age, 1st AXMEDIS International Conference, Florence, Italy, 2005.
- E. Selfridge-Field (Ed.), Beyond MIDI - The Handbook of Musical Codes, The MIT Press, London, UK, 1997.
- F. Pereira, and T. Ebrahimi (Eds.), The MPEG-4 Book, IMSC Press, 2002.
- Freehand: <http://www.freehandsystems.com/>
- G. M. Rader, “Creating Printed Music Automatically”, IEEE Computer, June 1996, pp.61-68.
- I-MAESTRO EC IST project, URL: <http://www.i-maestro.net>, <http://www.i-maestro.org>, <http://www.i-maestro.eu>
- IMUTUS: <http://www.exodus.gr/imutus/>
- J. S. Gourlay, “A Language for Music Printing”, Communications of the ACM, May 1986, 29(5):388-401.
- K.C. Ng, (ed), Journal of New Music Research (JNMR) special issue on Multimedia Music and the World Wide Web, 34(2), ISSN: 0929-8215, Routledge, 2005.
- M. Good, “MusicXML for Notation and Analysis”, In the Virtual Score Representation, Retrieval, Restoration, (W. B. Hewlett and E. Selfridge-Field, eds.) MT: The MIT Press, Cambridge, 2001, pp. 113-124. <http://www.recordare.com>
- MOODS project. <http://www.dsi.unifi.it/~moods>
- MOODS project. <http://www.dsi.unifi.it/~moods>
- MPEG SMR AHG web page: <http://www.interactivemusicnetwork.org/mpeg-ahg>
- MUSICALIS: <http://www.musicalis.fr/>
- N. Mitolo, P. Nesi, and K. C. Ng (eds.), Proceedings of the 5th MUSICNETWORK Open Workshop, Universität für Musik und darstellende Kunst Wien, Vienna, Austria, 2005.
- NIFF Consortium, “NIFF 6a: Notation Interchange File Format”, 1995.
- Notation, <http://www.notation.com>
- OPENDRAMA: <http://www.iua.upf.es/mtg/opendrama/>

- P. Bellini, J. Barthelemy, I. Bruno, P. Nesi, and M. B. Spinu, “Multimedia Music Sharing among Mediateques, Archives and Distribution to their attendees”, Journal on Applied Artificial Intelligence, Taylor and Francis, 2003, <http://www.wedelmusic.org>.
- P. Bellini, and P. Nesi, “WEDELMUSIC FORMAT: An XML Music Notation Format for Emerging Applications”, Proceedings of the 1st International Conference of Web Delivering of Music, IEEE press, 23-24 November 2001, Florence, Italy, pp. 79-86.
- P. Bellini, Della Santa, R., Nesi, P. (2001). Automatic Formatting of Music Sheet. Proc. of the First International Conference on WEB Delivering of Music, WEDELMUSIC-2001, IEEE Press, 23-24 November, Florence, Italy, pages 170-177.
- P. Bellini, F. Fioravanti, and P. Nesi, “Managing Music in Orchestras”, IEEE Computer, September 1999, pp. 26-34, <http://www.dsi.unifi.it/~moods/>.
- P. Bellini, I. Bruno, P. Nesi, “Automatic Formatting of Music Sheets through MILLA Rule-Based Language and Engine”, under publication on Journal of New Music Research.
- P. Bellini, Nesi, P., Spinu, M. B. (2002). Cooperative Visual Manipulation of Music Notation. ACM Transactions on Computer-Human Interaction, September, 9(3):194-237,
- P. Bellini, P. Nesi, G. Zoia, "Symbolic Music Representation in MPEG", IEEE Multimedia, IEEE Computer Society press, ISSN 1070-986X, Vol. 12, N. 4, pp. 42-29, October-December 2005.
- P. Bellini, P. Nesi, G. Zoia, “Symbolic Music Representation in MPEG for new Multimedia Applications”, IEEE Multimedia, 2005.
- R. B. Dannenberg, “A Brief Survey of Music Representation Issues, Techniques, and Systems”, Computer Music Journal, 1993, 17(3):20-30.
- R. B. Dannenberg, “A Structure for Efficient Update, Incremental Redisplay and Undo in Graphical Editors”, Software Practice and Experience, February 1990, 20(2):109-132.
- Sibelius Music Educational tools: <http://www.sibelius.com/>
- SMDL ISO/IEC, Standard Music Description Language. ISO/IEC DIS 10743, 1995.
- WEDELMUSIC: <http://www.wedelmusic.org>
- Yamaha tools: <http://www.digitalmusicnotebook.com/home/>

#### HP SCORM:

- <http://www.adlnet.gov>
- <http://www.adlnet.org>
- <http://www.ieee.org>
- <http://www.imsglobal.org>
- <http://www.ariadne-eu.org>
- <http://ltsc.ieee.org/wg12/>
- <http://www.reload.ac.uk/scormplayer.html>
- <http://koala.dls.au.com/scorm/>
- <http://www.scormplayer.com/>
- <http://www.e-learningconsulting.com/products/scorm-visualizer.html>
- <http://www.adlnet.gov/downloads/index.cfm?event=main.listing&categoryId=53>

**Overview:**

- <http://www.adlnet.gov/technologies/SCORM/index.cfm>
- <http://ltsc.ieee.org/wg12/>
- Alliance of Remote Instructional Authoring & Distribution Networks for Europe (ARIADNE) (<http://www.ariadne-eu.org/>)
- Aviation Industry CBT Committee (AICC) (<http://www.aicc.org/>)
- Institute of Electrical and Electronics Engineers (IEEE)
- Learning Technology Standards Committee (LTSC) (<http://ltsc.ieee.org/>)
- IMS Global Learning Consortium, Inc. (<http://www.imsglobal.org/>)
- [http://www.cancore.ca/docs/intro\\_e-learning\\_standardization.html](http://www.cancore.ca/docs/intro_e-learning_standardization.html)
- <http://exelearning.org/?q=about>
- [http://www.docebolms.org/doceboCms/page/23/E\\_Learning\\_scorm\\_tutorial\\_samples\\_open\\_source.html](http://www.docebolms.org/doceboCms/page/23/E_Learning_scorm_tutorial_samples_open_source.html)
- <http://www.lsal.cmu.edu/lsal/expertise/projects/scorm/scormevolution/reportv1p02/report-v1p02.html>
- <http://www.dotnetscorm.com/Home/tabid/36/Default.aspx>

**MAX Sources:**

<http://www.cycling74.com/download/> following Documents:

- AuthoringToolsApplicationGuidelines.pdf
- JavascriptInMax.pdf
- Jitter15Tutorial.pdf
- Max45ReferenceManual.pdf
- Max45TutorialsAndTopics.pdf
- MaxGettingStarted.pdf
- WhatsNewInMax/MSP455.pdf

**SDIF Sources:**

- Home page at IRCAM: <http://www.ircam.fr/sdif/>
- Project page at Berkeley: <http://www.cnmat.berkeley.edu/SDIF/>

**FTM Sources:**

- Home page: <http://www.ircam.fr/ftm/>

**OSC Sources:**

- Home page: <http://www.cnmat.berkeley.edu/OpenSoundControl/>
- Specification: <http://www.cnmat.berkeley.edu/OpenSoundControl/OSC-spec.html>
- Java library: <http://www.illposed.com/software/javaosc.html>
- OSCpack C++ library: <http://www.audiomulch.com/~rossb/code/oscpack/>