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

www.i-maestro.org, www.i-maestro.net

DE3.1.1b
I-MAESTRO Specification: Architecture Specification

Version: 2.3
Date: 28/04/2006
Responsible: DSI (Stefano Baldini, sbaldini@dsi.unifi.it, Nicola Mitolo, mitolo@dsi.unifi.it, Paolo Nesi, nesi@dsi.unifi.it)

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: DE3.1.1
Contractual Date of Delivery: M5
Actual Date of Delivery: 28/04/2006
Work-Package contributing to the Deliverable: WP3
Task contributing to the Deliverable: WP3
Nature of the Deliverable: Report
Author(s): ALL

Abstract:
This document reports the specification of the I-MAESTRO framework. The Deliverable DE3.1.1 is dichotomised into two parts: DE3.1.1a and DE3.1.1b. DE3.1.1a presents a general overview of the architecture. DE3.1.1b reports detail considerations of the architecture specification for the I-MAESTRO system.

Keyword List:
Requirements, use cases, test cases, content, interactive multimedia, education, music, creative, collaborative.

Table of Contents

1 Executive Summary and Report Scope

This document reports the specification for the first 6 months of the whole project activities. It takes information from the WP2 of user requirements, use cases and test cases and from the preliminary work performed by project partners and summarised in this document.

The Specification takes into account the general structure of the I-MAESWTRO framework and of the demonstrators planned in WP8. It is focused on mapping the research and development work on the real needs provided by the requirements, use cases and test cases:

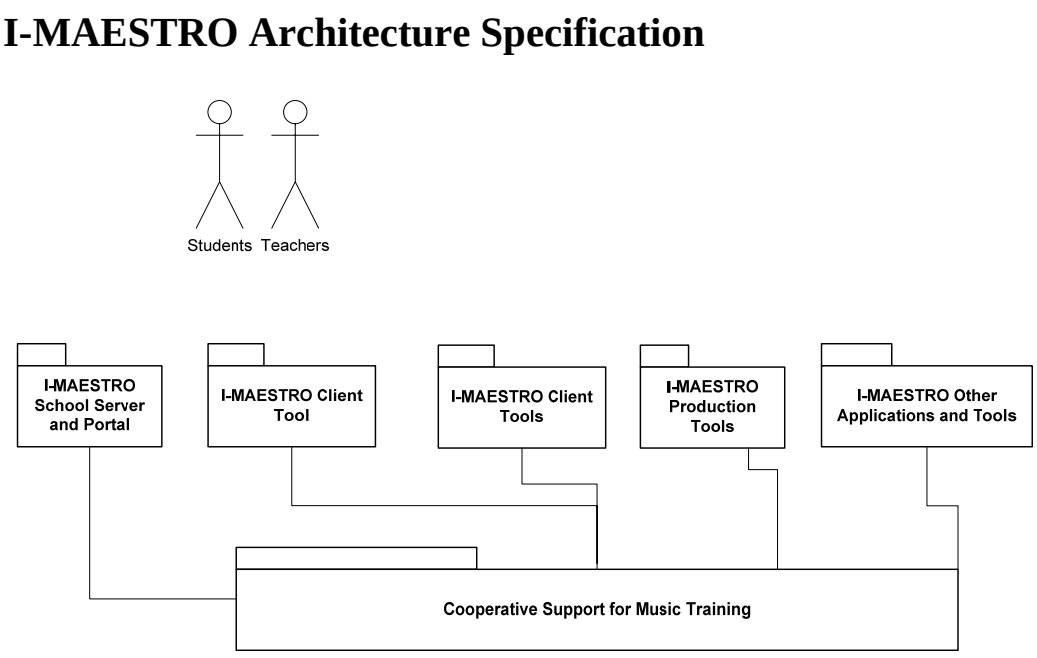
This is the second part of the document (part B). It reports detail aspects related to the architecture specification of the I-MAESTRO system.

2 General Overview of I-MAESTRO Architecture

In this section the I-MAESTRO architecture and main components are described.

2.1 The overview

The general architecture of the I-MAESTRO is showed in the following figure:



At a high level, I-MAESTRO Architecture is composed by:

- **I-MAESTRO Client Tools** support Teacher and Students in following Lesson and performing Exercises
- **I-MAESTRO Production Tools** are used by Teacher to create Exercises and package Lessons.
- **I-MAESTRO Other Application and Tools** include applications such as: a Metronome, a Tuner for instrument tuning, the Audio Recording Tool for audio analysis, the Video Recording Tool for gesture and posture analysis, etc.

- **I-MAESTRO School Server and Portal.** The School Server contains all available I-MAESTRO contents used by I-MAESTRO tools and all information about registered users (Students and Teacher profiles). The Portal provides also an external access to registered and anonymous users for downloading I-MAESTRO tools, manuals, user guides, information etc.
- **Cooperative Support for Music Training** allows I-MAESTRO Tools to work in a cooperative manner and to exchange information and contents.

2.2 The integration with the Cooperative support

The previous listed tools of I-MAESTRO can access to 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. The I-MAESTRO Other Application and Tools represents 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 authoring phase.

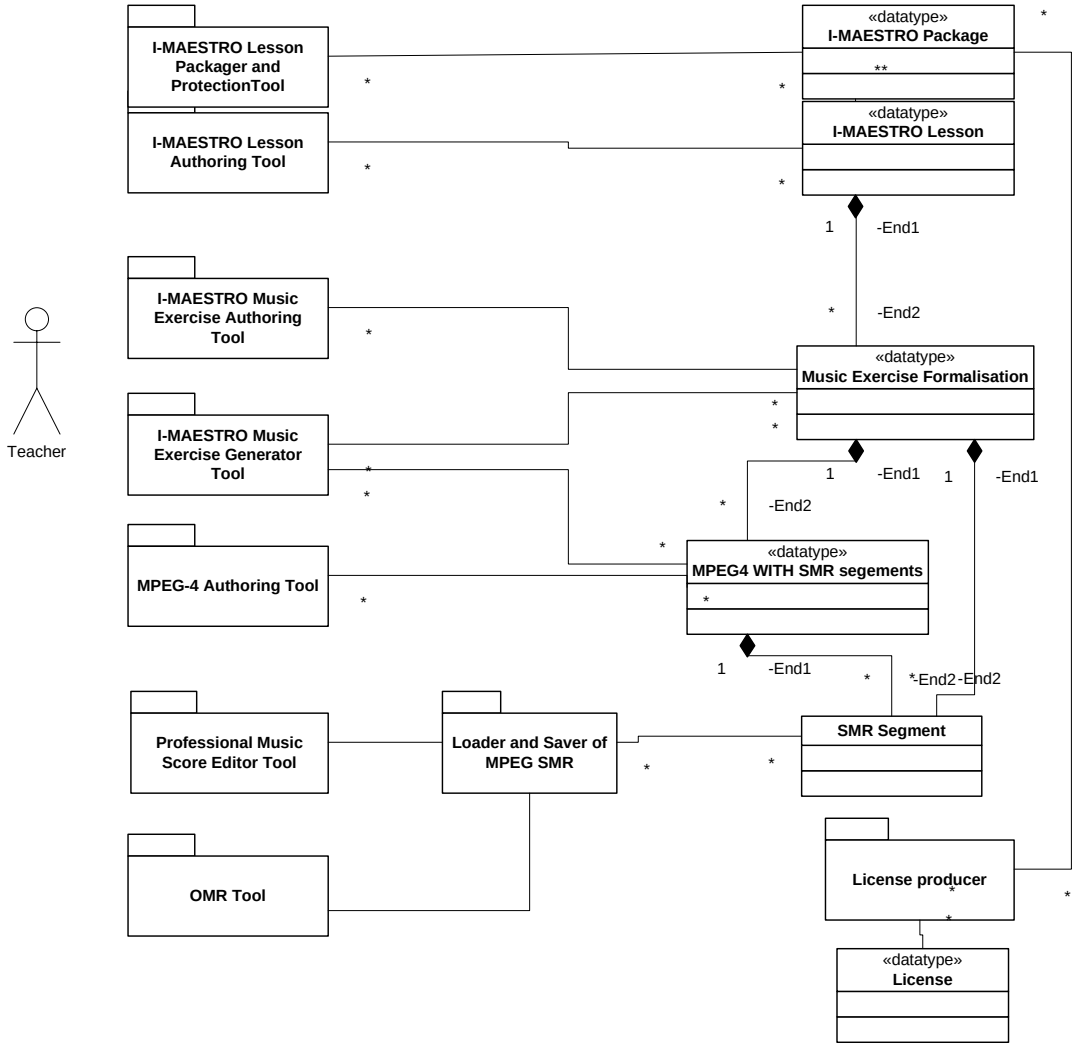
The I-MAESTRO School Server and the I-MAESTRO Production Tools don't use directly the peer to peer layer but they are available on the same network for the other tools to provide and store lessons and user profiles. (e.g. it is possible to search and download a lesson from the School Server using a Web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

3 I-MAESTRO Production Tools

I-MAESTRO Production Tools

- **I-MAESTRO Lesson Packager and Protection Tool**
- **I-MAESTRO Music Lesson Authoring (multimedia integration), as standard authoring, including the logic**
- **I-MAESTRO Music Exercise Authoring Tool, integrating: posture, gesture, etc.**
- **I-MAESTRO Music Exercise Generator: producing the MPEG SMR plus other information for the logic if needed**
- **MPEG-4 Authoring Tool**
- **Music Editing, producing MPEG SMR**
- **OMR**
- **other editors..**

I-MAESTRO Lesson Model

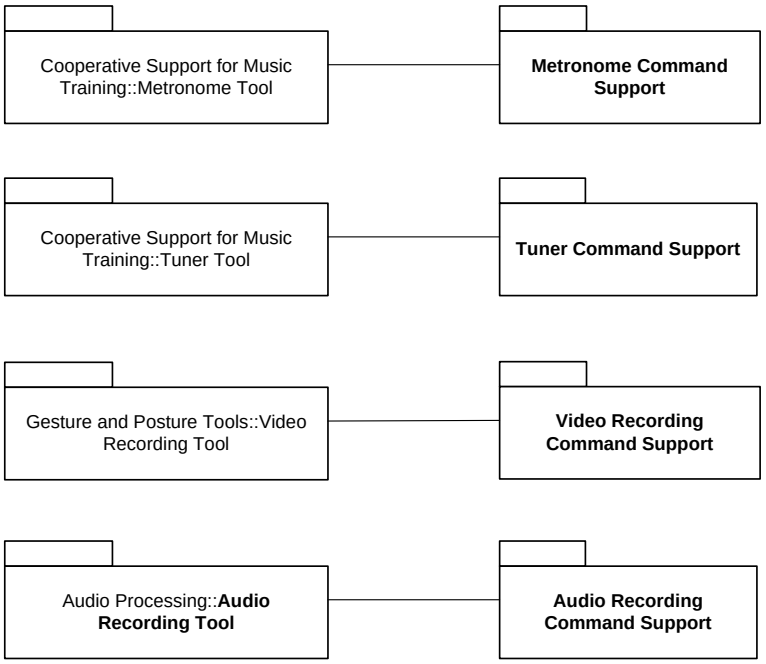


The I-MAESTRO Production Tools allows the teacher to create manually or automatically I-MAESTRO pedagogical contents for music education and collect them into structured music lessons.

The set of I-MAESTRO Production Tools consist in:

- The **I-MAESTRO Lesson Packager and Protection Tool,** that it is used to bundle lessons based on learning modalities and the pedagogic models.
- **I-MAESTRO Lesson Authoring Tool**, used by Teachers to create Lesson composed by Exercises or Units retrieved from the exercise repository. Using the Lesson Authoring Tool it is possible to specify also the contingent flow of exercises-execution with the help of a progress monitoring and control unit.
- **I-MAESTRO Music Exercise Authoring Tool**, used by Teacher to create Exercises composed by SMR Segments and/or MPEG-4 with SMR Segments,, images, audio, video, graphics etc. It also provides the teachers with options to select the assessment model and the input methods to be used in the exercise.
- **I-MAESTRO Music Exercise Generator Tool,** used to create music Exercises automatically or semi-automatically.
- **MPEG-4 Authoring Tool** , used to create MPEG-4 content with SMR Segments
- **Professional Music Score Editor**, used to create a scores in SMR format
- **Optical Music Recognition (OMR) Tool,** used to transcoding image of music sheet into SMR format.

3.1 I-MAESTRO Other Applications and Tools



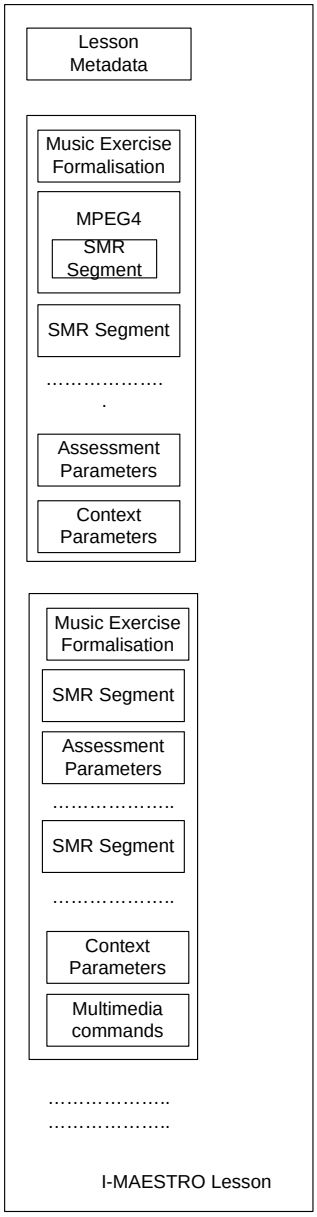
I-MAESTRO Other Application and Tools provide a set of different applications that can be used in different contexts:

- The Metronome tool, to mark the execution time
- The Tuner, to tuning the instrument
- The Audio Recording Tool, to acquire live music performance for online or off-line evaluation and storage
- The Video Recording Tool, to acquire the motion for the gesture and posture evaluation and storage.

Each tool has a correspondent module that provides commands (Command Support). These modules allow to control and monitoring the information that can be retrieve by each

4 Specification of I-MAESTRO Lesson Model

I-MAESTRO Lesson Model



Mainly a Lesson consists in (see User Requirements) one or more Exercises and logical information to implement the Lesson paradigm and relationship among contents. Lessons can be packaged before distribution. The model embeds concepts and structures coming from MPEG-4 and SMR formats.

Each Music Exercise is defined by:

- **Lesson Metadata**, containing the information describing the lesson
- **Music Exercise Formalization**, which defines the sequence of steps to perform the exercise
- **MPEG-4 and SMR Segment**, used to compose a multimedia files with symbolic music representation of music scores
- **SMR Segment**, it represents the symbolic music representation of music scores used as exercise
- **Assessment Parameters,** it defines the set of parameters used to asses the correctness of exercise's execution. They are used to compare the real performance of the pupil with the model and evaluate the level of performance, errors, and provide a score.
- **Context Parameters**, ...
- **Multimedia commands**, they represent commands that are executed to run automatically multimedia files (an audio or a video) when the pupil plays the exercise. For instance a hyperlink put on a note or a button could be associated with the execution of a video or a live recording.

4.1 Lesson Metadata

This metadata should be applied to Lessons, Exercises, Knowledge Units, assessment models/tools, etc., in order to classify them and to make possible links between Student profiles with different sets of Lessons or Exercises. Rigorous taxonomy classification in branches and sub-branches of the musical knowledge must be applied. For this reason this document should be considered as a first approach, not a definitive and closed model. On the contrary, it must be considered as opened to further modifications to attend the demands of each tool.

```
<xs:schema targetNamespace="http://purl.org/dc/elements/1.1/" elementFormDefault="qualified" attributeFormDefault="unqualified">

    <xs:annotation>
        <xs:documentation xml:lang="en">

Simple DC XML Schema, 2002-10-09 by Pete Johnston (p.johnston@ukoln.ac.uk), Carl Lagoze (lagoze@cs.cornell.edu), Andy Powell (a.powell@ukoln.ac.uk),Herbert Van de Sompel (hvdsomp@yahoo.com). This schema defines terms for Simple Dublin Core, i.e. the 15 elements from the http://purl.org/dc/elements/1.1/ namespace, with no use of encoding schemes or element refinements. Default content type for all elements is xs:string with xml:lang attribute available.
Supercedes version of 2002-03-12.  Amended to remove namespace declaration for http://www.w3.org/XML/1998/namespace namespace, and to reference lang attribute via built-in xml: namespace prefix. xs:appinfo also removed.
        </xs:documentation>
    </xs:annotation>
<xs:import namespace= "http://www.w3.org/XML/1998/namespace" schemaLocation="http://www.w3.org/2001/03/xml.xsd">
    </xs:import>

    <xs:element name="title" type="elementType"/>
    <xs:element name="creator" type="elementType"/>
    <xs:element name="subject" type="elementType"/>
    <xs:element name="description" type="elementType"/>
    <xs:element name="publisher" type="elementType"/>
    <xs:element name="contributor" type="elementType"/>
    <xs:element name="date" type="elementType"/>
    <xs:element name="type" type="elementType"/>
    <xs:element name="format" type="elementType"/>
    <xs:element name="identifier" type="elementType"/>
    <xs:element name="source" type="elementType"/>
    <xs:element name="language" type="elementType"/>
    <xs:element name="relation" type="elementType"/>
    <xs:element name="coverage" type="elementType"/>
    <xs:element name="rights" type="elementType"/>
```

```
    <xs:group name="elementsGroup">
        <xs:sequence>
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element ref="title"/>
                <xs:element ref="creator"/>
                <xs:element ref="subject"/>
                <xs:element ref="description"/>
                <xs:element ref="publisher"/>
                <xs:element ref="contributor"/>
                <xs:element ref="date"/>
                <xs:element ref="type"/>
                <xs:element ref="format"/>
                <xs:element ref="identifier"/>
                <xs:element ref="source"/>
                <xs:element ref="language"/>
                <xs:element ref="relation"/>
                <xs:element ref="coverage"/>
                <xs:element ref="rights"/>
            </xs:choice>
        </xs:sequence>
    </xs:group>
    <xs:complexType name="elementType">
        <xs:simpleContent>
            <xs:extension base="xs:string">
                <xs:attribute ref="xml:lang" use="optional"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:schema>
```

4.1.1 Categories of the whole spectrum of musical practice and teaching

Musical technique

Musical technique is divided into specific technique, when dealing with each instrument, and general technique, when it deals with general observations that are valid for any specialty.

a) General technique:

General technique for all instruments, fragmented according to following aspects:

- General Music Theory (i.e. harmony: intervals, triad structure, chords; formal design such as motives, phrases, periods; music history, etc.)
- Physiological and motor function aspects (i.e. gesture and posture, coordination, flexibility (i.e. body, joints, wrist, arms, etc.), finger extension, breathing, etc.)
- Sound quality aspects (i.e. weight, evenness, preparing the sound, bow pressure, bow speed, etc.)
- Aspects regarding the performance (i.e. fingerings, glissandi, bowings, virtuosity, etc.)
- General aspects of instruments' mechanics (i.e. care of the instruments, hair of the bow, thickness-tension-length of the strings, mute, resonant bodies, etc.)

b) Specific technique:

Specific technique deals with each specialty's exclusive technique. It will have as many subdivisions as instrumental families, for example, strings, winds, singing, etc. Each one of these specialties will in turn have three categories:

- Family (i.e. bowed string instruments. woodwinds, brass, etc.)
- Mechanics (i.e. aspects regarding materials, construction, etc.)
- Execution (i.e. tune, specific bow technique, left hand technique, etc.)
- Historical approach to early music technique (i.e. early music ornamentations/improvisation on the basis of musicological research, basso continuo according to the insights obtained by studying historical sources, early music's string technique, etc.)

Musical science

This section comprises, on one side, concepts regarding the six elemental aspects of music: Melody, Rhythm, Harmony, Form, Timbre and Texture. On the other side, it includes the categories dealing with musicology such as: Notation, History, Organology, Aesthetics, Ethnic musicology, Music psychology, Acoustics and Hearing.

Each one of these general sections is further divided into smaller categories, reaching in the end the isolated concept that ends each subdivision. For example, the "history" category ends with concepts such as: "the author", "the work" , "styles and periods", etc. Or within the "hearing" category we first find two more categories: "live sound" and "recorded sound". These end with derived concepts such as: the public, sound recording format types, etc.

Capabilities of the musician

Concepts regarding the musician's aptitude and attitude towards music.
Attitude: the musician's attitude towards the work, regarding his studies, towards the teacher or before the public, etc.

Aptitude: it comprises all of the musician's capabilities and acquired, as well as innate, abilities that allow the development and performance of music. For example, the musician's ability regarding flexibility, physical motor coordination, interpretation abilities, memory capacity, etc.

Musical expression aspects

We consider that musical expression is the result of combining scientific and technical abilities with the actions the musician carries out to achieve the goal of musical expression. Considering musical expression as the goal and only reality that justifies the existence of all previous actions and abilities.

The section dealing with musical expression is divided into:

- Elemental aspects of musical expression: dynamics, agogycs, accentuation, articulation, etc.
- Complex aspects of musical expression, from an extra-musical point of view, in other words, outside the boundaries of music (expressivity, character, description, etc.), as well as from a strictly musical point of view (phrasing, musical tension, etc.)

Metadata (a few tags examples and taxonomy)

```
<?xml version="1.0" encoding="UTF-8"?>

<!–
    Document   : iMaestroGeneric.xsd
    Author     : Fundación Albéniz
    Description:
        Common Generic Types
-->

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
            Generic Types for I-MAESTRO project.
            Fundación Albéniz
        </xsd:documentation>
    </xsd:annotation>
            <xsd:simpleType name="ValorationRange">
        <xsd:restriction base="xsd:integer">
            <xsd:minInclusive value="1"/>
            <xsd:maxInclusive value="5"/>
```

```
        </xsd:restriction>
    </xsd:simpleType>
    </xsd:schema>

=-=-=-=-=-=-=-=-=

<!--
    Document   : iMaestroMetadata.xsd
    Author     : Fundación Albéniz
    Description:
        Lesson Metadata.
-->

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
            Metadata Schema for I-MAESTRO project.
            Fundación Albéniz
        </xsd:documentation>
    </xsd:annotation>

    <xsd:include schemaLocation="iMaestroGeneric.xsd"/>

    <xsd:element name="lessonMetadata" type="LessonMetadataType"/>

    <xsd:simpleType name="stringList">
        <xsd:list itemType="xsd:string"/>
    </xsd:simpleType>

<?xml version="1.0" encoding="UTF-8"?>

    <xsd:complexType name="InstrumentNameType">
        <xsd:restriction base="xsd:string">
        <xsd:enumeration value="generic bow"/>
        <xsd:enumeration value="violin"/>
        <xsd:enumeration value="viola"/>
        <xsd:enumeration value="cello"/>
        <xsd:enumeration value="double bass"/>
        <xsd:enumeration value="bow"/>
        <xsd:enumeration value="generic plucked strings"/>
    </xsd:restriction>
    </xsd:complexType>

    <xsd:complexType name="InstrumentFamilyType">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="bow"/>
        </xsd:restriction>
    </xsd:complexType>

    <xsd:complexType name="InstrumentType">
        <xsd:sequence>
            <xsd:element name="name" type="InstrumentNameType"/>
            <xsd:element name="family" type="InstrumentFamilyType"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:simpleType name="instrumentList">
        <xsd:list itemType="InstrumentType"/>
    </xsd:simpleType>

    <xsd:complexType name="MechanicsKeywords">
        <xsd:restriction base="xsd:string">
        <xsd:enumeration value="bowHairs"/>
        <xsd:enumeration value="strings"/>
        <xsd:enumeration value="chinRest"/>
        <xsd:enumeration value="shoulderRest"/>
        <xsd:enumeration value="bridge"/>
        <xsd:enumeration value="pegs"/>
        <xsd:enumeration value="neck"/>
        <xsd:enumeration value="fingerboard"/>
        <xsd:enumeration value="soundpost"/>
        <xsd:enumeration value="bow"/>
        <xsd:enumeration value="frogBow"/>
        <xsd:enumeration value="middleBow"/>
        <xsd:enumeration value="tipBow"/>
        <xsd:enumeration value="tailPiece"/>
```

```
            <xsd:enumeration value="fret"/>
            <xsd:enumeration value="plectrum"/>
            <xsd:enumeration value="top"/>
            <xsd:enumeration value="instrumentBuilder"/>
           </xsd:restriction>
        </xsd:complexType>

        <xsd:simpleType name="mechanicsKeywordsList">
            <xsd:list itemType="MechanicsKeywords"/>
        </xsd:simpleType>

        <xsd:complexType name="ExecutionKeywords">
            <xsd:restriction base="xsd:string">
                <xsd:enumeration value="plucking"/>
                <xsd:enumeration value="strumming"/>
                <xsd:enumeration value="vibrato"/>
                <xsd:enumeration value="string crossing"/>
                <xsd:enumeration value="scordatura"/>
                <xsd:enumeration value="shifts"/>
                <xsd:enumeration value="pizzicato"/>
                <xsd:enumeration value="bowings"/>
                <xsd:enumeration value="bow direction"/>
                <xsd:enumeration value="bow inclination"/>
                <xsd:enumeration value="bow distribution"/>
                <xsd:enumeration value="bow stroke"/>
                <xsd:enumeration value="collé"/>
                <xsd:enumeration value="detaché"/>
                <xsd:enumeration value="martellé"/>
                <xsd:enumeration value="ricochet"/>
                <xsd:enumeration value="sautillé"/>
                <xsd:enumeration value="spicatto"/>
                <xsd:enumeration value="stacatto"/>
                <xsd:enumeration value="generic bow stroke"/>
                <xsd:enumeration value="alla tastiera"/>
                <xsd:enumeration value="sul ponticello"/>
                <xsd:enumeration value="col legno"/>
            </xsd:restriction>
        </xsd:complexType>

        <xsd:simpleType name="executionKeywordsList">
            <xsd:list itemType="ExecutionKeywords"/>
        </xsd:simpleType>

        <xsd:complexType name="PedagogicModelType">
            <xsd:restriction base="xsd:string">
                <xsd:enumeration value="self-study"/>
                <xsd:enumeration value="one-to-one"/>
                <xsd:enumeration value="classroom"/>
                <xsd:enumeration value="distance-learning"/>
                <xsd:enumeration value="cooperative-work"/>
            </xsd:restriction>
        </xsd:complexType>

        <xsd:complexType name="LessonMetadataType">
            <xsd:sequence>
                <xsd:element name="author" type="xsd:string" maxOcurrs="1"/><!--   why only one
    author?? -->
                <xsd:element name="date" type="xsd:date" maxOccurs="1"/>
                <xsd:element name="permissionRights" type="xsd:string"/>
                <xsd:element name="description" type="xsd:string" maxOccurs="1"/>
                <xsd:element name="targetAudience" type="xsd:string"/>
                <xsd:element name="compentencyLevel" type="xsd:string"/>
                <xsd:element name="keywords" type="stringList"/>
                <xsd:element name="executionKeywords" type="executionKeywordsList"/>
                <xsd:element name="mechanicsKeywordsList" type="mechanicsKeywordsList"/>
                <xsd:element name="targetedInstruments" type="instrumentList"/>
                <xsd:element name="difficulty" type="ValorationRange"/>
                <xsd:element name="pedagogicModel" type="PedagogicModelType"/>
                <!-- Reference to the right IMaestroExerciseUnit type should be
                changed, and the referenced schema should be included at the
                beggining -->
                <xsd:element name="exerciseUnit" type="IMaestroExerciseUnit"/>
            </xsd:sequence>
        </xsd:complexType>

    </xsd:schema>
```

4.2 Music Exercise Formalisation

4.2.1 Explanation of XML scheme for Exercise Unit Formalisation

This definition shows the structure of Exercise Units. Every possible progress, test, assessment should be possible. So the Author can design the Exercise depending on his own preferences, the dependences to the earlier and following units, the Level, the kind of user, the pedagogical Method and the textual classification.

1 Category
1.1 theory (Yes or No)
1.2 practice (Yes or No)
1.3 Author (Name of the Author)
1.4 Date (of doing the Exercise)
1.5 Keywords (important words, in order to easily find and identify the Exercise Unit especially for search mechanisms)
1.5 pedagogic Model (notice, if Exercise Unit is based on a certain pedagogic Model)
1.6 Target group (Students, amateur, pupil)
1.7 Content description (a short description in plain words, what the exercise is about)

2 Systematical content coherence
Each array described in 1 to 5 words or/and predefined categories for data base search machines:

2.1 Subject (general, e.g. notes and clefs or Strings)
2.2 Themes (e. g. notes or bowing)
2.3 Details (e. g. middle C or matele)

3. Level
3.1 Level 1 (professional)
3.2 Level 2
3.3 Level 3
3.4 Level 4
3.5 Level 5 (beginner)

4. Relation and context to next exercise
The teacher could suggest more or less relevant Knowledge or Exercise Units, so the way of going on is predefined. Another option is, that the pupil him/herself can decide which Unit he wants to do next.

4.1 Teacher suggest next exercise units
4.1.1 Subject (general, e.g. notes and clefs or Strings)
4.1.2 Themes (e. g. notes or bowing)
4.1.3 Details (e. g. middle C or matele)

4.2.1 Higher
4.2.2 Lower level
4.2.3 Same level

4.3 Pupil can decide what to do next or skip a Unit
4.3.1 Higher
4.3.2 Lower level
4.3.3 Same level (Variation)
4.3.3 Doing the same Unit

5. Iterations
How many iterations the user has to do, 0 to n= 10, showed to the user.

5.1 Number of Recursive Iteration 1- 10
5.2 Number of repetitions Iteration 1- 10

6. Time slice
Time allowed to do the test or exercise (e. g. 20 sec)

6.1 Time to pass test in seconds: n sec.
6.1.2 Test-passing:
6.1.2.1 Higher Level
6.1.2.2 Variation of the same Level

6.2.2 Test-failure:
6.2.2.1 Lower Level
6.2.2.2 Variation of the same Level
6.2.2.3 Repeating the test

7. Type of test or exercise input:
7.1 Multiple Choice
7.1.1 Weighting of the boxes (e.g. 1 to 5)
7.1.2 Failure limit (e.g. min 8 of max10 scores)

7.1.2.1 Test-failure:
7.1.2.1.1 Lower Level
7.1.2.1.2 Variation of the same Level
7.1.2.1.3 Repeating the test

7.1.2.2 Test-passing:
7.1.2.2.1 Higher Level
7.1.2.2.2 Variation of the same

7.2 Written text, notes, etc.
7.2.1 Failure limit
7.2.2 Input 1- 5 (analyzed by exercise Generator)

7.2.2.1 Test-failure:
7.2.2.1.1 Lower Level
7.2.2.1.2 Variation of the same Level
7.2.2.1.3 Repeating the test

7.2.2.2 Test-passing:
7.2.2.2.1 Higher Level
7.2.2.2.2 Variation of the same

7.3 Audio, Video etc.
7.3.1 Failure limit
7.3.2 Input 1- 5 (analyzed by pitch recognition tool, gesture recognition tool, etc)

7.3.2.1 Test-failure:

7.3.2.1.1 Lower Level
7.3.2.1.2 Variation of the same Level
7.3.2.1.3 Repeating the test

7.3.2.2 Test-passing:
7.3.2.2.1 Higher Level
7.3.2.2.2 Variation of the same

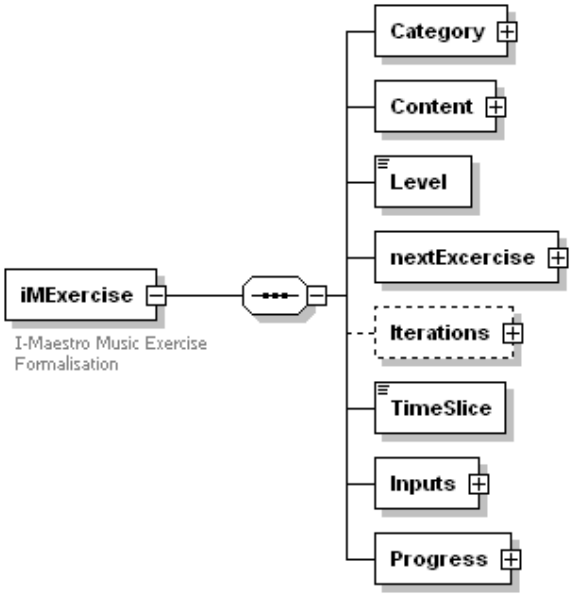
8. Status and progress
8.1 Number of unit
8.2 Number of all units in the lesson
8.3 Maximum score

4.2.2 XML scheme for Exercise Unit Formalisation

Elements	Complex types
iMExercise	**Assessment**
	Suggestion
	UnitDetails

element **iMExercise**

diagram



properties content complex

children **Category Content Level nextExcercise Iterations TimeSlice Inputs Progress**

annotation documentation
I-Maestro Music Exercise Formalisation

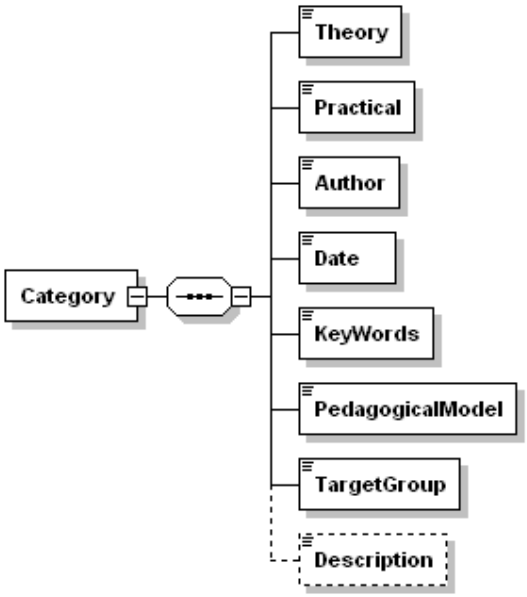
source
```
<xs:element name="iMExercise">
 <xs:annotation>
  <xs:documentation>I-Maestro Music Exercise Formalisation</xs:documentation>
 </xs:annotation>
 <xs:complexType>
  <xs:sequence>
   <xs:element name="Category">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="Theory" type="xs:boolean"/>
      <xs:element name="Practical" type="xs:boolean"/>
      <xs:element name="Author" type="xs:string"/>
      <xs:element name="Date" type="xs:date"/>
```

```
      <xs:element name="KeyWords" type="xs:string"/>
      <xs:element name="PedagogicalModel" type="xs:string"/>
      <xs:element name="TargetGroup" type="xs:string"/>
      <xs:element name="Description" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="Content">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Subject" type="xs:string"/>
      <xs:element name="Themes" type="xs:string"/>
      <xs:element name="Details" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="Level" type="xs:int"/>
<xs:element name="nextExcercise">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="TeacherSuggestion" type="Suggestion" minOccurs="0"/>
      <xs:element name="PupilSuggestion" type="Suggestion" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="Iterations" minOccurs="0">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Recursive" minOccurs="0">
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:int">
              <xs:attribute name="Variable" type="xs:string"/>
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
      <xs:element name="Repitation" type="xs:int" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="TimeSlice" type="xs:unsignedInt"/>
<xs:element name="Inputs">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="MultipleChoice" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="BoxWeights" type="xs:int"/>
            <xs:element name="FailureLimit" type="Assessment"/>
          </xs:sequence>
          <xs:attribute name="ToolName" type="xs:string"/>
        </xs:complexType>
      </xs:element>
      <xs:element name="WrittenNotes" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Weights" type="xs:int"/>
            <xs:element name="FailureLimit" type="Assessment"/>
          </xs:sequence>
          <xs:attribute name="ToolName" type="xs:string"/>
        </xs:complexType>
      </xs:element>
      <xs:element name="Audio" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Weight"/>
            <xs:element name="FailureLimit" type="Assessment"/>
          </xs:sequence>
          <xs:attribute name="ToolName" type="xs:string"/>
        </xs:complexType>
      </xs:element>
      <xs:element name="Video" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
```

```
            <xs:element name="Weight"/>
            <xs:element name="FailureLimit" type="Assessment"/>
           </xs:sequence>
           <xs:attribute name="ToolName" type="xs:string"/>
          </xs:complexType>
         </xs:element>
         <xs:element name="Other" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
           <xs:sequence>
            <xs:element name="Weight"/>
            <xs:element name="FailureLimit" type="Assessment"/>
           </xs:sequence>
           <xs:attribute name="ToolName" type="xs:int"/>
          </xs:complexType>
         </xs:element>
        </xs:sequence>
       </xs:complexType>
      </xs:element>
      <xs:element name="Progress">
       <xs:complexType>
        <xs:sequence>
         <xs:element name="UnitNo" type="xs:int"/>
         <xs:element name="TotalUnits" type="xs:int"/>
         <xs:element name="Score" type="xs:int"/>
        </xs:sequence>
       </xs:complexType>
      </xs:element>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
```

element **iMExercise/Category**

diagram



properties

isRef	0
content	complex

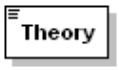
children **Theory Practical Author Date KeyWords PedagogicalModel TargetGroup Description**

source

```
<xs:element name="Category">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="Theory" type="xs:boolean"/>
   <xs:element name="Practical" type="xs:boolean"/>
   <xs:element name="Author" type="xs:string"/>
   <xs:element name="Date" type="xs:date"/>
   <xs:element name="KeyWords" type="xs:string"/>
   <xs:element name="PedagogicalModel" type="xs:string"/>
```

```
      <xs:element name="TargetGroup" type="xs:string"/>
      <xs:element name="Description" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

element **iMExercise/Category/Theory**

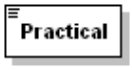
diagram 

type **xs:boolean**

properties isRef 0
content simple

source <xs:element name="Theory" type="xs:boolean"/>

element **iMExercise/Category/Practical**

diagram 

type **xs:boolean**

properties isRef 0
content simple

source <xs:element name="Practical" type="xs:boolean"/>

element **iMExercise/Category/Author**

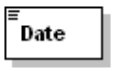
diagram 

type **xs:string**

properties isRef 0
content simple

source <xs:element name="Author" type="xs:string"/>

element **iMExercise/Category/Date**

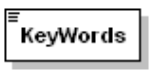
diagram 

type **xs:date**

properties isRef 0
content simple

source <xs:element name="Date" type="xs:date"/>

element **iMExercise/Category/KeyWords**

diagram 

type **xs:string**

properties isRef 0
content simple

source <xs:element name="KeyWords" type="xs:string"/>

element **iMExercise/Category/PedagogicalModel**

diagram

type **xs:string**

properties
isRef 0
content simple

source
```
<xs:element name="PedagogicalModel" type="xs:string"/>
```

element **iMExercise/Category/TargetGroup**

diagram

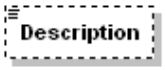
type **xs:string**

properties
isRef 0
content simple

source
```
<xs:element name="TargetGroup" type="xs:string"/>
```

element **iMExercise/Category/Description**

diagram



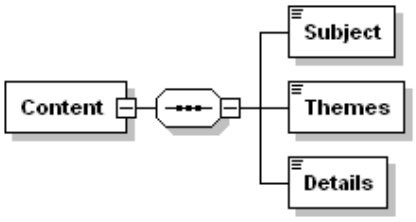
type **xs:string**

properties
isRef 0
minOcc 0
maxOcc 1
content simple

source
```
<xs:element name="Description" type="xs:string" minOccurs="0"/>
```

element **iMExercise/Content**

diagram

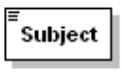


properties
isRef 0
content complex

children **Subject Themes Details**

source
```
<xs:element name="Content">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Subject" type="xs:string"/>
      <xs:element name="Themes" type="xs:string"/>
      <xs:element name="Details" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

element **iMExercise/Content/Subject**

diagram 

type **xs:string**

properties isRef 0
content simple

source
```
<xs:element name="Subject" type="xs:string"/>
```

element **iMExercise/Content/Themes**

diagram 

type **xs:string**

properties isRef 0
content simple

source
```
<xs:element name="Themes" type="xs:string"/>
```

element **iMExercise/Content/Details**

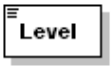
diagram 

type **xs:string**

properties isRef 0
content simple

source
```
<xs:element name="Details" type="xs:string"/>
```

element **iMExercise/Level**

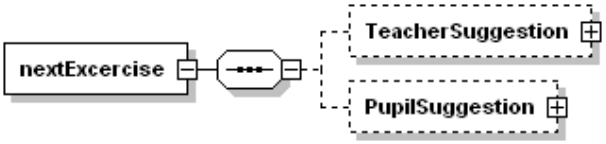
diagram 

type **xs:int**

properties isRef 0
content simple

source
```
<xs:element name="Level" type="xs:int"/>
```

element **iMExercise/nextExcercise**

diagram 

properties isRef 0
content complex

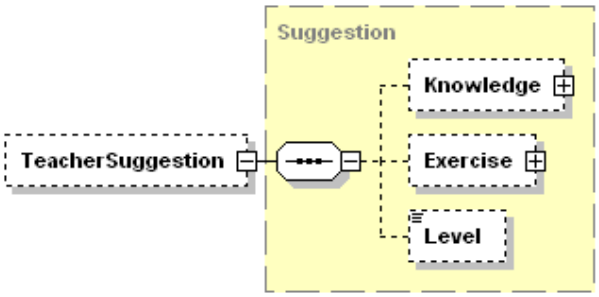
children **TeacherSuggestion PupilSuggestion**

source
```
<xs:element name="nextExcercise">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="TeacherSuggestion" type="Suggestion" minOccurs="0"/>
      <xs:element name="PupilSuggestion" type="Suggestion" minOccurs="0"/>
```

```
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

element **iMExercise/nextExcercise/TeacherSuggestion**

diagram



type **Suggestion**

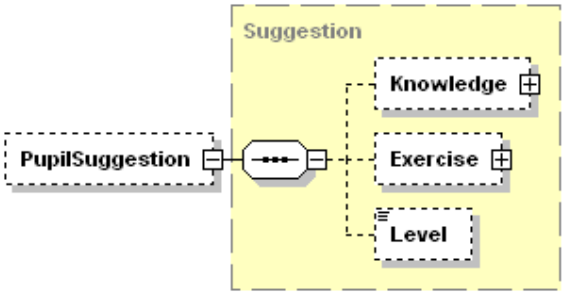
properties
- isRef 0
- minOcc 0
- maxOcc 1
- content complex

children **Knowledge Exercise Level**

source
```
<xs:element name="TeacherSuggestion" type="Suggestion" minOccurs="0"/>
```

element **iMExercise/nextExcercise/PupilSuggestion**

diagram



type **Suggestion**

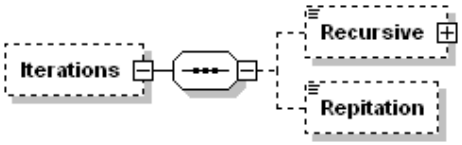
properties
- isRef 0
- minOcc 0
- maxOcc 1
- content complex

children **Knowledge Exercise Level**

source
```
<xs:element name="PupilSuggestion" type="Suggestion" minOccurs="0"/>
```

element **iMExercise/Iterations**

diagram



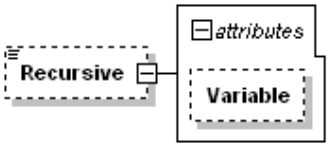
properties
- isRef 0
- minOcc 0
- maxOcc 1
- content complex

children **Recursive Repitation**

source
```
<xs:element name="Iterations" minOccurs="0">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="Recursive" minOccurs="0">
    <xs:complexType>
     <xs:simpleContent>
      <xs:extension base="xs:int">
       <xs:attribute name="Variable" type="xs:string"/>
      </xs:extension>
     </xs:simpleContent>
    </xs:complexType>
   </xs:element>
   <xs:element name="Repitation" type="xs:int" minOccurs="0"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

element **iMExercise/Iterations/Recursive**

diagram



type extension of **xs:int**

properties

isRef	0
minOcc	0
maxOcc	1
content	complex

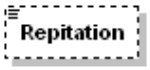
attributes

Name	Type	Use	Default	Fixed	Annotation
Variable	**xs:string**				

source
```
<xs:element name="Recursive" minOccurs="0">
 <xs:complexType>
  <xs:simpleContent>
   <xs:extension base="xs:int">
    <xs:attribute name="Variable" type="xs:string"/>
   </xs:extension>
  </xs:simpleContent>
 </xs:complexType>
</xs:element>
```

element **iMExercise/Iterations/Repitation**

diagram



type **xs:int**

properties

isRef	0
minOcc	0
maxOcc	1
content	simple

source
```
<xs:element name="Repitation" type="xs:int" minOccurs="0"/>
```

element **iMExercise/TimeSlice**

diagram

type **xs:unsignedInt**

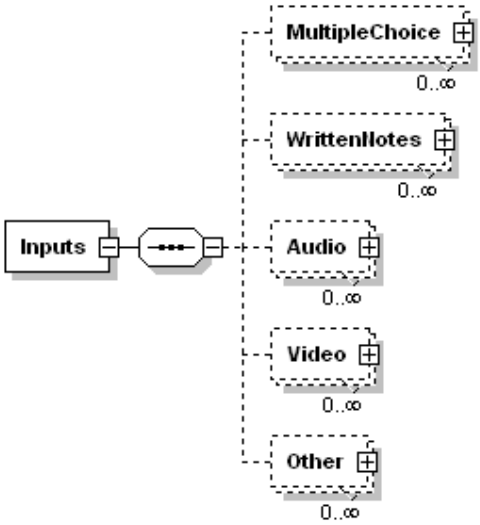
properties

isRef	0
content	simple

source
```
<xs:element name="TimeSlice" type="xs:unsignedInt"/>
```

element **iMExercise/Inputs**

diagram



properties
isRef 0
content complex

children **MultipleChoice WrittenNotes Audio Video Other**

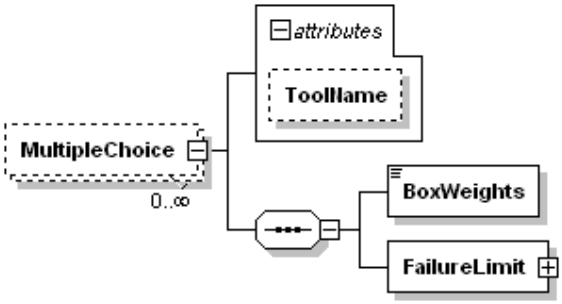
source

```
<xs:element name="Inputs">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="MultipleChoice" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="BoxWeights" type="xs:int"/>
      <xs:element name="FailureLimit" type="Assessment"/>
     </xs:sequence>
     <xs:attribute name="ToolName" type="xs:string"/>
    </xs:complexType>
   </xs:element>
   <xs:element name="WrittenNotes" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="Weights" type="xs:int"/>
      <xs:element name="FailureLimit" type="Assessment"/>
     </xs:sequence>
     <xs:attribute name="ToolName" type="xs:string"/>
    </xs:complexType>
   </xs:element>
   <xs:element name="Audio" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="Weight"/>
      <xs:element name="FailureLimit" type="Assessment"/>
     </xs:sequence>
     <xs:attribute name="ToolName" type="xs:string"/>
    </xs:complexType>
   </xs:element>
   <xs:element name="Video" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="Weight"/>
      <xs:element name="FailureLimit" type="Assessment"/>
     </xs:sequence>
     <xs:attribute name="ToolName" type="xs:string"/>
    </xs:complexType>
   </xs:element>
   <xs:element name="Other" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="Weight"/>
```

```
        <xs:element name="FailureLimit" type="Assessment"/>
       </xs:sequence>
       <xs:attribute name="ToolName" type="xs:int"/>
      </xs:complexType>
     </xs:element>
    </xs:sequence>
   </xs:complexType>
  </xs:element>
```

element **iMExercise/Inputs/MultipleChoice**

diagram



properties	isRef 0 minOcc 0 maxOcc unbounded content complex
children	**BoxWeights FailureLimit**

attributes

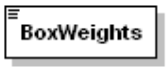
Name	Type	Use	Default	Fixed	Annotation
ToolName	**xs:string**				

source

```
<xs:element name="MultipleChoice" minOccurs="0" maxOccurs="unbounded">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="BoxWeights" type="xs:int"/>
   <xs:element name="FailureLimit" type="Assessment"/>
  </xs:sequence>
  <xs:attribute name="ToolName" type="xs:string"/>
 </xs:complexType>
</xs:element>
```

element **iMExercise/Inputs/MultipleChoice/BoxWeights**

diagram



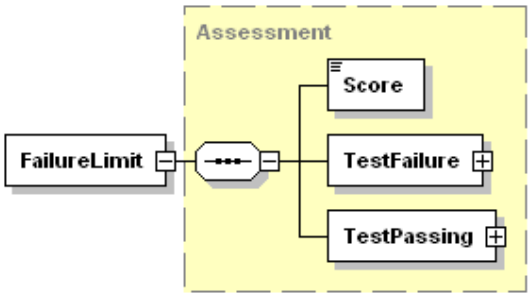
type	**xs:int**
properties	isRef 0 content simple

source

```
<xs:element name="BoxWeights" type="xs:int"/>
```

element **iMExercise/Inputs/MultipleChoice/FailureLimit**

diagram



type **Assessment**

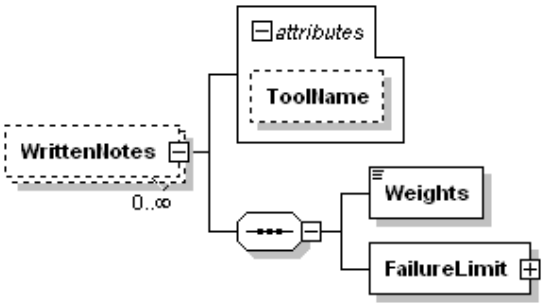
properties
isRef	0
content	complex

children **Score TestFailure TestPassing**

source
```
<xs:element name="FailureLimit" type="Assessment"/>
```

element **iMExercise/Inputs/WrittenNotes**

diagram



properties
isRef	0
minOcc	0
maxOcc	unbounded
content	complex

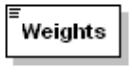
children **Weights FailureLimit**

attributes
Name	Type	Use	Default	Fixed	Annotation
ToolName	**xs:string**				

source
```
<xs:element name="WrittenNotes" minOccurs="0" maxOccurs="unbounded">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="Weights" type="xs:int"/>
   <xs:element name="FailureLimit" type="Assessment"/>
  </xs:sequence>
  <xs:attribute name="ToolName" type="xs:string"/>
 </xs:complexType>
</xs:element>
```

element **iMExercise/Inputs/WrittenNotes/Weights**

diagram



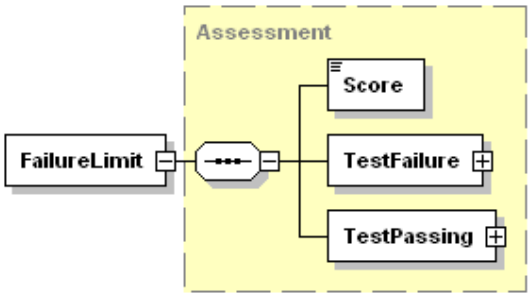
type **xs:int**

properties
isRef	0
content	simple

source
```
<xs:element name="Weights" type="xs:int"/>
```

element **iMExercise/Inputs/WrittenNotes/FailureLimit**

diagram



type **Assessment**

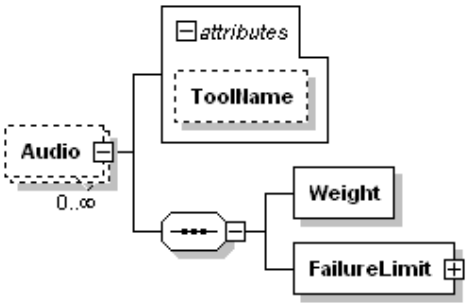
properties
isRef 0
content complex

children **Score TestFailure TestPassing**

source
```
<xs:element name="FailureLimit" type="Assessment"/>
```

element **iMExercise/Inputs/Audio**

diagram



properties
isRef 0
minOcc 0
maxOcc unbounded
content complex

children **Weight FailureLimit**

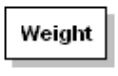
attributes

Name	Type	Use	Default	Fixed	Annotation
ToolName	**xs:string**				

source
```
<xs:element name="Audio" minOccurs="0" maxOccurs="unbounded">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="Weight"/>
   <xs:element name="FailureLimit" type="Assessment"/>
  </xs:sequence>
  <xs:attribute name="ToolName" type="xs:string"/>
 </xs:complexType>
</xs:element>
```

element **iMExercise/Inputs/Audio/Weight**

diagram

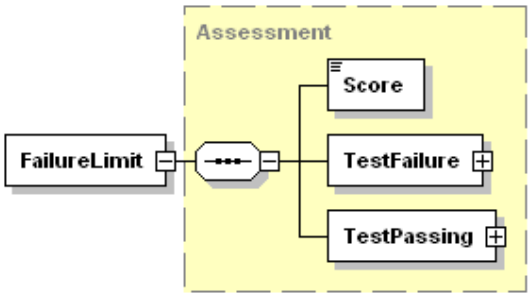


properties
isRef 0

source
```
<xs:element name="Weight"/>
```

element **iMExercise/Inputs/Audio/FailureLimit**

diagram



type **Assessment**

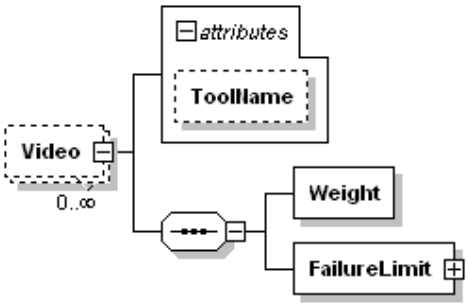
properties
isRef	0
content	complex

children **Score TestFailure TestPassing**

source
```
<xs:element name="FailureLimit" type="Assessment"/>
```

element **iMExercise/Inputs/Video**

diagram



properties
isRef	0
minOcc	0
maxOcc	unbounded
content	complex

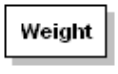
children **Weight FailureLimit**

attributes
Name	Type	Use	Default	Fixed	Annotation
ToolName	**xs:string**				

source
```
<xs:element name="Video" minOccurs="0" maxOccurs="unbounded">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="Weight"/>
   <xs:element name="FailureLimit" type="Assessment"/>
  </xs:sequence>
  <xs:attribute name="ToolName" type="xs:string"/>
 </xs:complexType>
</xs:element>
```

element **iMExercise/Inputs/Video/Weight**

diagram

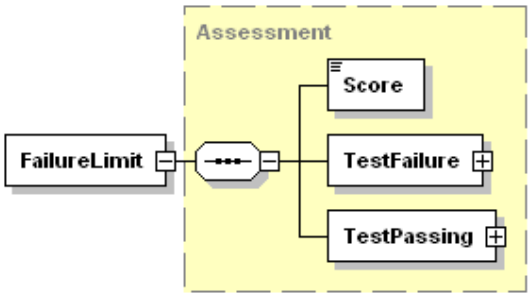


properties
isRef	0

source
```
<xs:element name="Weight"/>
```

element **iMExercise/Inputs/Video/FailureLimit**

diagram



type **Assessment**

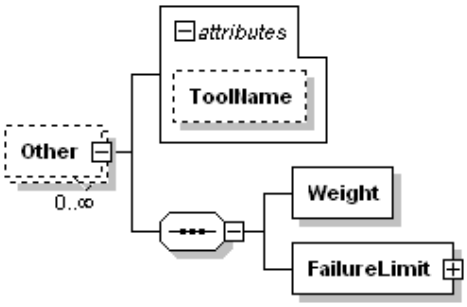
properties
isRef 0
content complex

children **Score TestFailure TestPassing**

source
```
<xs:element name="FailureLimit" type="Assessment"/>
```

element **iMExercise/Inputs/Other**

diagram



properties
isRef 0
minOcc 0
maxOcc unbounded
content complex

children **Weight FailureLimit**

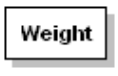
attributes

Name	Type	Use	Default	Fixed	Annotation
ToolName	**xs:int**				

source
```
<xs:element name="Other" minOccurs="0" maxOccurs="unbounded">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="Weight"/>
   <xs:element name="FailureLimit" type="Assessment"/>
  </xs:sequence>
  <xs:attribute name="ToolName" type="xs:int"/>
 </xs:complexType>
</xs:element>
```

element **iMExercise/Inputs/Other/Weight**

diagram

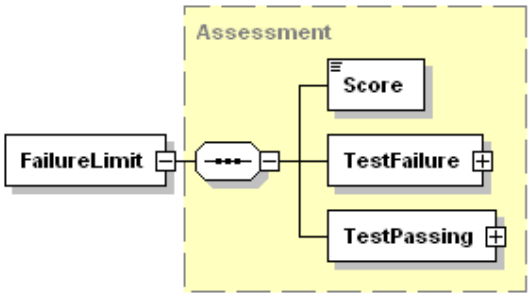


properties
isRef 0

source
```
<xs:element name="Weight"/>
```

element **iMExercise/Inputs/Other/FailureLimit**

diagram



type **Assessment**

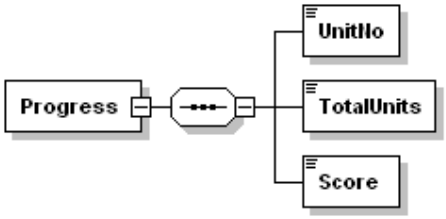
properties isRef 0
content complex

children **Score TestFailure TestPassing**

source
```
<xs:element name="FailureLimit" type="Assessment"/>
```

element **iMExercise/Progress**

diagram



properties isRef 0
content complex

children **UnitNo TotalUnits Score**

source
```
<xs:element name="Progress">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="UnitNo" type="xs:int"/>
      <xs:element name="TotalUnits" type="xs:int"/>
      <xs:element name="Score" type="xs:int"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

element **iMExercise/Progress/UnitNo**

diagram

type **xs:int**

properties isRef 0
content simple

source
```
<xs:element name="UnitNo" type="xs:int"/>
```

element **iMExercise/Progress/TotalUnits**

diagram

type **xs:int**

properties isRef 0
content simple

source
```
<xs:element name="TotalUnits" type="xs:int"/>
```

element **iMExercise/Progress/Score**

diagram

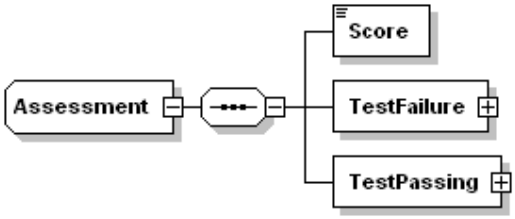
type **xs:int**

properties isRef 0
content simple

source
```
<xs:element name="Score" type="xs:int"/>
```

complexType **Assessment**

diagram



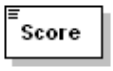
children **Score TestFailure TestPassing**

used by elements **iMExercise/Inputs/MultipleChoice/FailureLimit iMExercise/Inputs/WrittenNotes/FailureLimit iMExercise/Inputs/Audio/FailureLimit iMExercise/Inputs/Video/FailureLimit iMExercise/Inputs/Other/FailureLimit**

source
```
<xs:complexType name="Assessment">
 <xs:sequence>
  <xs:element name="Score" type="xs:int"/>
  <xs:element name="TestFailure">
   <xs:complexType>
    <xs:sequence>
     <xs:element name="LowerLevel" type="xs:boolean"/>
     <xs:element name="Variation" type="xs:boolean"/>
     <xs:element name="Repeat" type="xs:boolean"/>
    </xs:sequence>
   </xs:complexType>
  </xs:element>
  <xs:element name="TestPassing">
   <xs:complexType>
    <xs:sequence>
     <xs:element name="HigherLevel" type="xs:boolean"/>
     <xs:element name="Variation" type="xs:boolean"/>
    </xs:sequence>
   </xs:complexType>
  </xs:element>
 </xs:sequence>
</xs:complexType>
```

element **Assessment/Score**

diagram



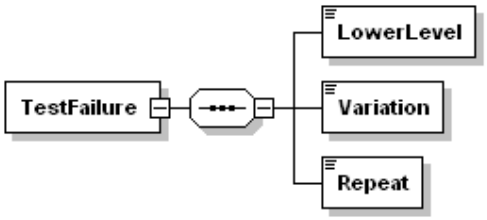
type **xs:int**

properties isRef 0
content simple

source
```
<xs:element name="Score" type="xs:int"/>
```

element **Assessment/TestFailure**

diagram



properties
isRef 0
content complex

children **LowerLevel Variation Repeat**

source
```
<xs:element name="TestFailure">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="LowerLevel" type="xs:boolean"/>
   <xs:element name="Variation" type="xs:boolean"/>
   <xs:element name="Repeat" type="xs:boolean"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

element **Assessment/TestFailure/LowerLevel**

diagram

type **xs:boolean**

properties
isRef 0
content simple

source
```
<xs:element name="LowerLevel" type="xs:boolean"/>
```

element **Assessment/TestFailure/Variation**

diagram

type **xs:boolean**

properties
isRef 0
content simple

source
```
<xs:element name="Variation" type="xs:boolean"/>
```

element **Assessment/TestFailure/Repeat**

diagram

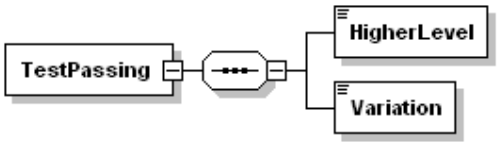
type **xs:boolean**

properties
isRef 0
content simple

source
```
<xs:element name="Repeat" type="xs:boolean"/>
```

element **Assessment/TestPassing**

diagram



properties
isRef 0
content complex

children **HigherLevel Variation**

source
```
<xs:element name="TestPassing">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="HigherLevel" type="xs:boolean"/>
   <xs:element name="Variation" type="xs:boolean"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

element **Assessment/TestPassing/HigherLevel**

diagram

type **xs:boolean**

properties
isRef 0
content simple

source
```
<xs:element name="HigherLevel" type="xs:boolean"/>
```

element **Assessment/TestPassing/Variation**

diagram

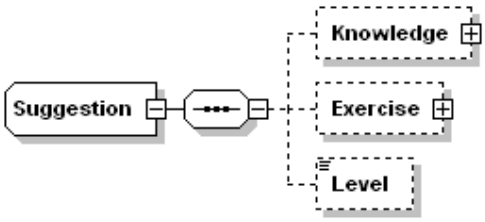
type **xs:boolean**

properties
isRef 0
content simple

source
```
<xs:element name="Variation" type="xs:boolean"/>
```

complexType **Suggestion**

diagram



children **Knowledge Exercise Level**

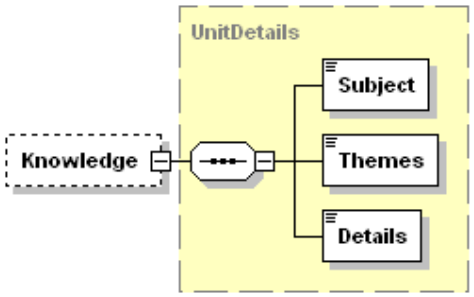
used by
elements **iMExercise/nextExcercise/PupilSuggestion iMExercise/nextExcercise/TeacherSuggestion**

source
```
<xs:complexType name="Suggestion">
 <xs:sequence>
  <xs:element name="Knowledge" type="UnitDetails" minOccurs="0"/>
  <xs:element name="Exercise" type="UnitDetails" minOccurs="0"/>
  <xs:element name="Level" type="xs:int" minOccurs="0"/>
```

```
    </xs:sequence>
</xs:complexType>
```

element **Suggestion/Knowledge**

diagram



type **UnitDetails**

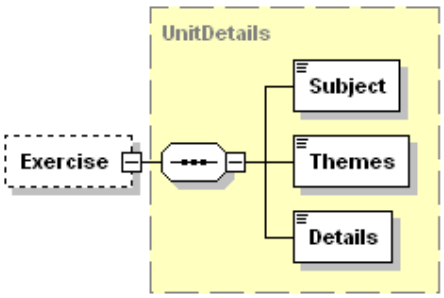
properties
- isRef 0
- minOcc 0
- maxOcc 1
- content complex

children **Subject Themes Details**

source
```
<xs:element name="Knowledge" type="UnitDetails" minOccurs="0"/>
```

element **Suggestion/Exercise**

diagram



type **UnitDetails**

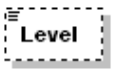
properties
- isRef 0
- minOcc 0
- maxOcc 1
- content complex

children **Subject Themes Details**

source
```
<xs:element name="Exercise" type="UnitDetails" minOccurs="0"/>
```

element **Suggestion/Level**

diagram



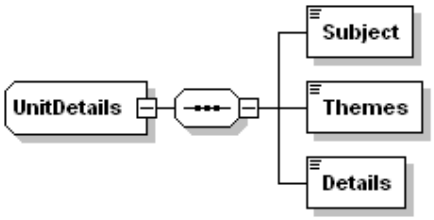
type **xs:int**

properties
- isRef 0
- minOcc 0
- maxOcc 1
- content simple

source
```
<xs:element name="Level" type="xs:int" minOccurs="0"/>
```

complexType **UnitDetails**

diagram



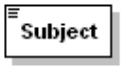
children **Subject Themes Details**

used by
elements **Suggestion/Exercise Suggestion/Knowledge**

source
```
<xs:complexType name="UnitDetails">
 <xs:sequence>
  <xs:element name="Subject" type="xs:string"/>
  <xs:element name="Themes" type="xs:string"/>
  <xs:element name="Details" type="xs:string"/>
 </xs:sequence>
</xs:complexType>
```

element **UnitDetails/Subject**

diagram



type **xs:string**

properties
isRef 0
content simple

source
```
<xs:element name="Subject" type="xs:string"/>
```

element **UnitDetails/Themes**

diagram

type **xs:string**

properties
isRef 0
content simple

source
```
<xs:element name="Themes" type="xs:string"/>
```

element **UnitDetails/Details**

diagram

type **xs:string**

properties
isRef 0
content simple

source
```
<xs:element name="Details" type="xs:string"/>
```

4.3 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 bitstreams 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.

4.4 SMR Segments for exercises

These are in MPEG-SMR.

4.5 SMR and Multimedia

MPEG-4 permits the encoding of multimedia content, including many different kinds of object types and a scene description allowing precise synchronization 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 20th 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 synchronization),
- Software for entertainment (mainly synchronization 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 synchronization 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.

4.6 Assessment Parameters

Assessment is a integral component of the teaching and learning process: is the application of information about the student performances or learning effectiveness to make educational decisions, and in all this process is fundamental a clear alignment between what is taught and what is learn.

EXAMPLES OF ASSESSMENT PARAMETERS:

An example of a possible type of rating scales:

- Excellent (5 points): highest category of level-work
- Adequate (4-3 points): acceptable level work
- Needs improvement (3-1 points): weak level work
- No adequate (no answer or fault) (0 points)

An example of possible level of achievement criteria:

- Level of achievement:
    1. Excellent/very good
    2. Adequate
    3. Needs improvement

An example of the standards criteria for a particular task:

- Conceptual understanding level:
    1. Excellent (demonstrate clear understanding)
    2. Adequate (demonstrate a partial understanding)
    3. Needs improvement (does not demonstrate understanding)

An example of skills checklist

- Pitch accuracy
    Focus on concepts attributed to the development of good intonation, including test for:
        Dictation (also with chords)
        Intervals recognition
- Tone quality
    Focus on concepts attributed to the development of good tone production, including test for:

Good playing position
Appropriate bowing
Fingering
Bow techniques
Posture
Vibrato

- Rhythm

  Focus on concepts attributed to the development of rhythmical skills, including test for:

  Changing meter, irregular beat divisions
  Polyrhythm, polymeter
  Rhythms dictation
  Pulse, rhythmic precision
  Simple meter, subdivide
  Meter identificación
  Tempo markings: a*ccelerando, meno mosso, piu mosso, retardando,* etc.

- Dynamics

  Focus on concepts attributed to the development of the knowledge of different dynamics, including test for:

  Bow technique
  forte piano, forte, fortissimo, mezzo forte, mezzo piano, pianissimo,piano, sforzando, sostenuto, subito, etc.

- Posture

  Focus on concepts attributed to the development of good posture, including test for:

  Good stand position: i.e. control of the feed position, balance of the body weight, etc.
  Good sit position: i.e. keep the back straight, sit at the front of the chair, etc.
  To play in a correct position i.e. check the left elbow position, correct setting of the fingers in the fingerboard, thumb placed, etc.

- Articulation (and diction for singers)

  Focus on concepts attributed to the development of articulation, including test for:

  Bow technique articulation
  Accent
  Legato, marcato, pizzicato, spiccato, staccato, tenuto

- Expression/ Musical Interpretation

  Focus on concepts attributed to the development of expression and interpretation skills, including test for:

  Musical form
  Music history, style, tempo
  Melodic direction, phrase structure
  Tempo and stylistic
  Making tension – release
  Dolce, maestoso, etc.

- Harmony

  Focus on concepts attributed to the development of harmony, including test for:

  Functional harmony (recognition of simple cadence patterns, all diatonic triads and inv., etc.)
  Spelling of major and minor triads
  Structural and formal analysis, roman numeral realization, figured bass realization, soprano harmonization, etc.
  Simple/advanced modulations

- Score Reading

  Focus on concepts attributed to the development of score reading, including test for:

  Sight reading
  Full score reading
  Transposition

- Playing/singing together/cooperative work
  Focus on concepts attributed to the development of cooperative work, including test for:
  Professional attitude and behaviour during rehearsals and performances
  Learn leadership (by sectional rehearsals)
- Concentration and focus
  Focus on concepts attributed to the development of concentration skills, including test for:
  Sound projection

Schema for the assessment parameters.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
          Metadata Schema for I-MAESTRO project.
          Assessment parameters
          iMaestroAssessmentParams.xsd
       </xsd:documentation>
    </xsd:annotation>
    <xsd:include schemaLocation="iMaestroGeneric.xsd"/>

<xsd:simpleType name="AssessmentMethod">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="Human"/>
    <xsd:enumeration value="Automatic"/>
    <xsd:enumeration value="Reflective"/>
    <xsd:enumeration value="Adaptive"/>
    <xsd:enumeration value="Static"/>
  </xsd:restriction>
</xsd:simpleType>
    <!-- The above AssessmentMethod type possibly to be moved in the overall Lesson metadata -->

    <xsd:simpleType name="AssessmentLevel">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="Lenient"/>
            <xsd:enumeration value="Medium"/>
            <xsd:enumeration value="Strict"/>
        </xsd:restriction>
    </xsd:simpleType>
    <!-- The above AssessmentLevel type possibly to be moved in the overall Lesson metadata -->

    <xsd:element name="assessment">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="ID"/>
                <xsd:element name="method" type="AssessmentMethod"/>
                <xsd:element name="level" type="AssessmentLevel"/>
                <xsd:element name="date" type="xsd:date"/>    <xsd:element name="assessment">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="ID"/>
                <xsd:element name="method" type="AssessmentMethod"/>
                <xsd:element name="level" type="AssessmentLevel"/>
                <xsd:element name="date" type="xsd:date"/>
                <xsd:element name="topic" maxOccurs="unbounded"/>
                <!-- could be captured from lesson/exercise. They could be limited but > 1 -->
                <xsd:element name="author"/>
                <!-- author of the assessment. -->
                <xsd:element name="student"/>
                <!-- deriving/linking with profile(s) could “catch” other information from the student’s metadata-->
                <xsd:element name="mark">
                    <xsd:complexType>
                        <xsd:attribute name="criteria" use="required"/>
                        <xsd:attribute name="metrics"  use="required"/>
                    </xsd:complexType>
                </xsd:element>
                <!-- depending on criteria and metrics used the types for ‘criteria’ and ‘name’ to be defined -->
                <xsd:element name="passed" type="xsd:boolean"/>
                <xsd:element name="definitive" type="xsd:boolean"/>
                <!-- for possible partial assessments (e.g. part of a lesson/exercise) -->
                <xsd:element name="comment" minOccurs="0"/>
```

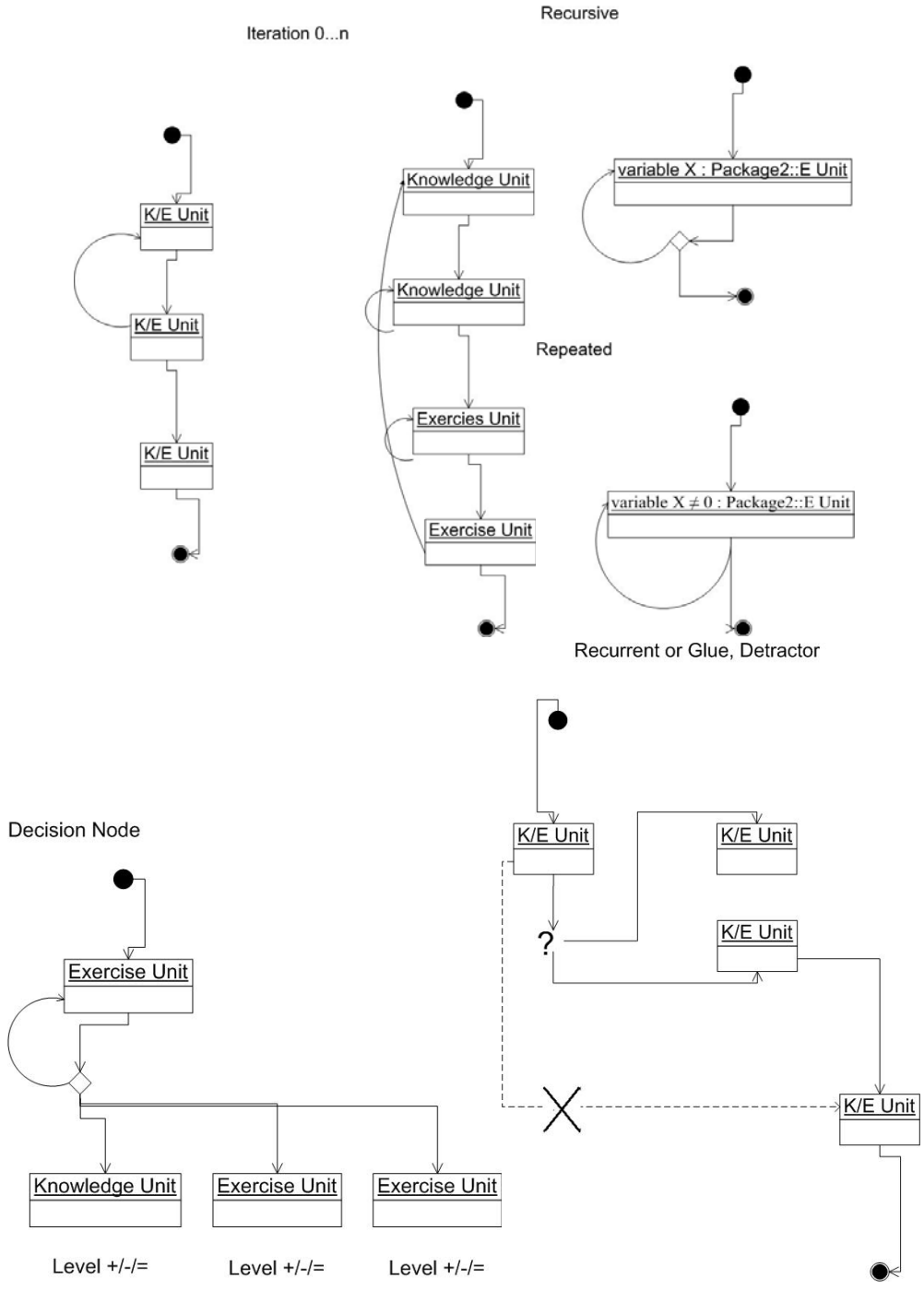
```
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
```

4.7 Context Parameters

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
        Metadata Schema for I-MAESTRO project.
        Context parameters(broad)
        iMaestroContextParams.xsd
      </xsd:documentation>
    </xsd:annotation>
    <xsd:include schemaLocation="iMaestroGeneric.xsd"/>
    <xsd:simpleType name="PedagogicModelType">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="self-study"/>
            <xsd:enumeration value="one-to-one"/>
            <xsd:enumeration value="classroom"/>
            <xsd:enumeration value="distance-learning"/>
            <xsd:enumeration value="cooperative"/>
            <!-- The above type to be possibly moved up in overall lesson metadata -->
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:element name="context">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="Teacher"/>
                <xsd:element name="NumberOfStudents" type="xsd:nonNegativeInteger"/>
                <xsd:element name="PedagogicModel" type="PedagogicModelType"/>
                <xsd:element name="StudentsAverageAge"/> <!-- Type to be specified -->
                <xsd:element name="StudentsAverageLevel"/> <!-- Type to be specified -->
                <xsd:element name="ImpairedStudents" type="xsd:boolean" minOccurs="0"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

4.8 Logic of the lesson, the glue

The following pictures show the basic structure of the possibilities of progress from Unit to Unit within a lesson:



4.9 Packaging the Lesson

Module/Tool Profile		
Packaging the Lesson		
Responsible Name	Marius Spinu	
Responsible Partner	Exitech	
Status (proposed/approved)	Proposed	
Implemented/not implemented	Not implemented	
Status of the implementation	0%	
Executable or Library/module (Support)	library	
Single Thread or Multithread		
Language of Development		
Platforms supported		
Reference to the location of the source code demonstrator		
Reference to the location of the demonstrator executable tool for internal download		
Reference to the location of the demonstrator executable tool for public download		
Address for accessing to WebServices if any, add accession information (user and Passwd) if any		
Test cases (present/absent)		
Test cases location		
Major Problems not solved		
Major pending requirements		
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
Formats Used	Shared with	format name or reference to a section
Protocol Used	Shared with	Protocol name or reference to a section
Used Database name		
IMDB		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not

4.9.1 General Description of the Module

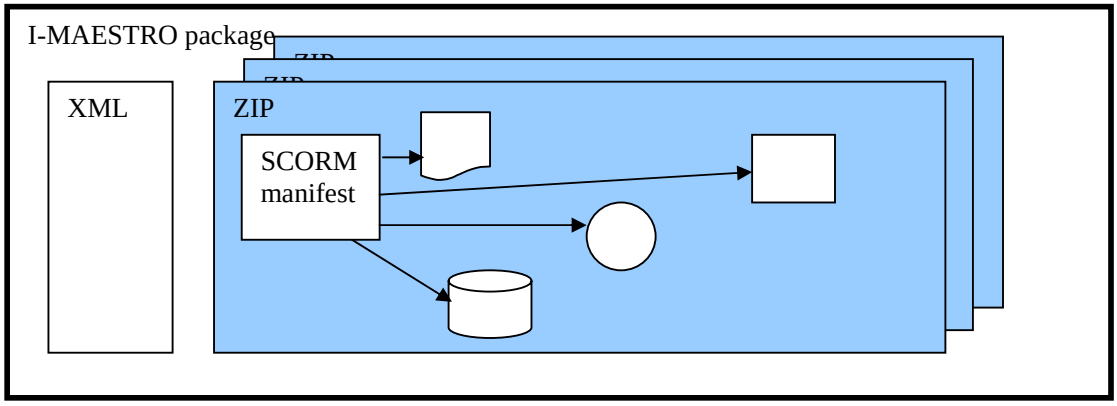
The Teacher as Course creator has to create Lessons by adding Exercises Units and Knowledge Units. More Lessons create a course which can be considered a first level of packaging.

A package will be composed by an XML header and by one or more zipped files.

The XML header will contain general information about the Lessons include into the package as:

- Author
- Content description
- Lesson identification
- Assessment Used
- Genre
- Level of difficulty
- Targeted Audience (Age, Level of Competency)
- Order of Lessons (The possible paths of the Lesson and the decisions to follow it)
- Permission and sharing rights

Each zip file will contain several types of files including Lesson Metadata, Music Exercise Formalization, MPEG-4 and SMR Segment, representation of music scores, SMR Segment, Assessment Parameters, Context Parameters, Multimedia commands, material (PDF, text, flash, video, audio, etc) needed for the Knowledge Units, etc. It is expected that the result of the authoring tool is accompanied with a SCORM manifest file in order to link all the available resources.

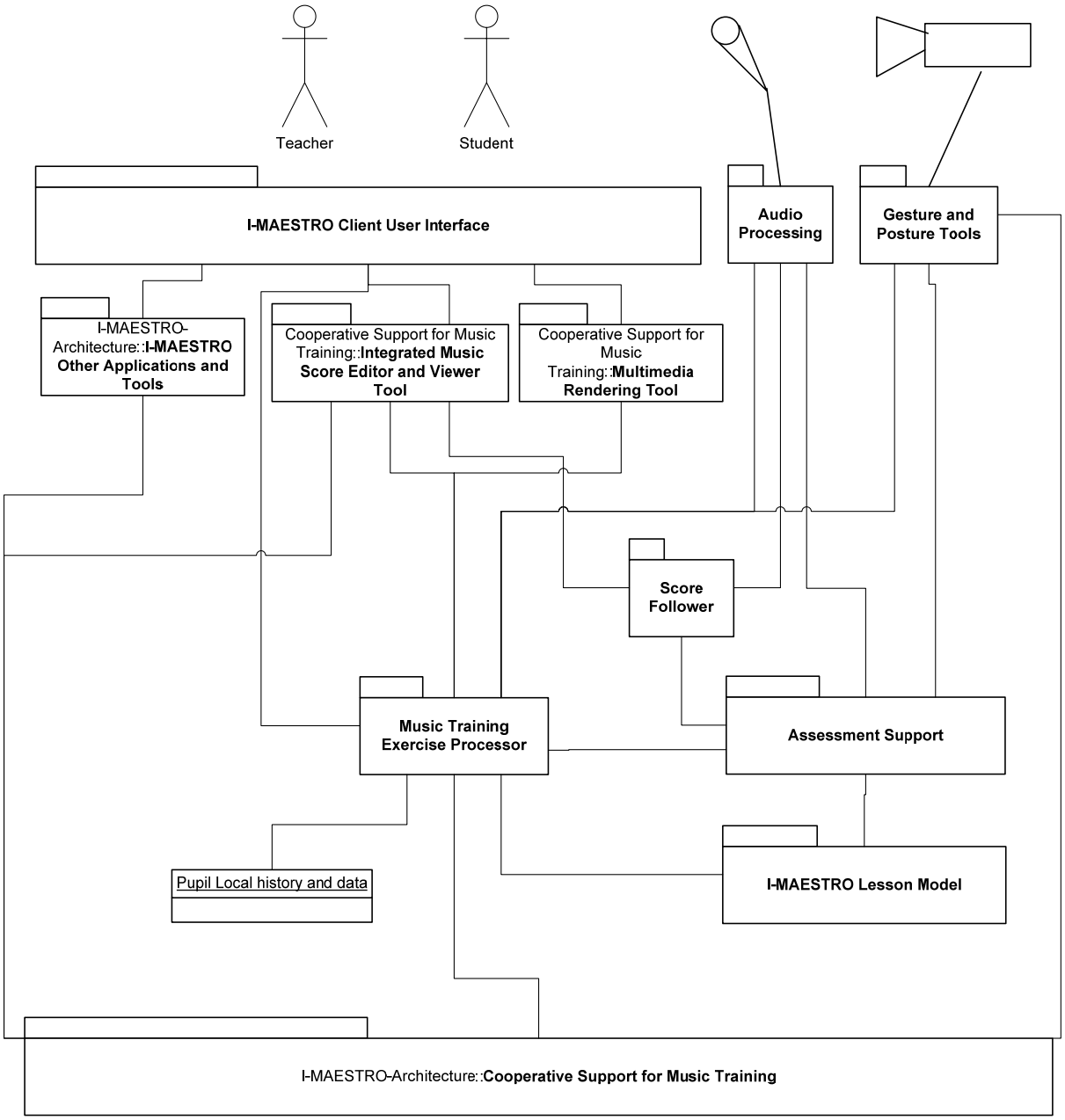


5 Specification of I-MAESTRO Client Architecture

5.1 I-MAESTRO Client General Overview

I-MAESTRO Client:

- **A tool in the hands of the Pupils for executing I-MAESTRO exercises. It can be based on:**
  - **MPEG-4 player,**
  - **MaxMSP player, also integrating SCORM information**
- **The client is capable of processing a lesson in I-MAESTRO format**



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 Other Applications (Tuner, Metronome, etc.), the Integrated Music Score Editor and Viewer Tool, the Multimedia Rendering Tool and Music Training Exercise Processor.
- **Music Training Exercise Processor.** It is the core of I-MAESTRO Client and it is able to manage (sending and receiving commands) to 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 about performances depending on the evaluations of the Assessment Support.
- **I-MAESTRO Lesson Model**. It 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 assess it.
- **Gesture and Posture Tools.** It uses a set of tools to assess the Students executions capturing data from sensors and 3D Motion Data capturing system.
- **Score Follower.** It is used in some type of exercise 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 pass 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 about performance's quality.

5.2 I-MAESTRO Client User Interface

Client user interface of the lesson can be created in the Max/MSP environment.

5.3 I-MAESTRO Client Accessible User Interface

User interface options

In order to launch the accessible user interface from the I-Maestro client tool the user will select one of the converting options which results in and Accessible Music notation:

Figure: Accessible Music Conversion options

The menu bar will be accessible either by keyboard shortcuts or by screen reader assistance. Selecting one of these options will launch one of the accessible music decoder screens. From a system architecture perspective, these are very much the same module (See section 6.8 of this document), but from a user perspective, they are using distinct modules. As a result there are a set of generic functionalities of these system options and some more output specific options. These are shown below.

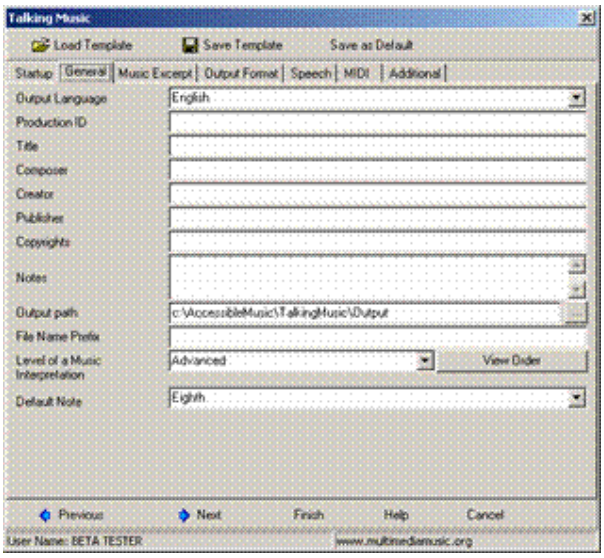


Figure: Example user preferences screen

Generic functionalities

Welcome screen: This window provides the user with options as to where they will go next. If the user wishes to create a rendering of the I-MAESTRO lesson using generic preferences or preferences fed by the I-MAESTRO client tool, he can click "convert". Other wise, there are several options which are described below. The user can enter data in None, Some or all of the fields.

General Screen: General Window you can enter information about the accessible I-MAESTRO lesson. These relate to, output languages, save path. Many of these options will call other dialog boxes from the I-MAESTRO client tool.

Musical Excerpt: This allows the user to choose to convert specific parts or instrument which can be picked from a full I-MAESTRO lesson. This is important for teachers creating a lesson for a class which contains both impaired and non-impaired students. The user can also select a section of the music, dictated by measure/bar numbers. With this function you can make specific movements or parts of a score into separate Accessible music lessons.

Format specific options

Talking Music Output options: Depending on the system setup, there will be a choice of several synthetic speech voices. Once a voice has been selected, users can select bit rate for the output audio. This along with midi playback options makes up the user preference setting for Talking Music.

Braille Printing: This dialog box allows you to choose a printer for Braille output. This could be a standard printer or a special Braille Embosser. The printer dialog box connects to the standard printing functionality of the I-MAESTRO client tool. This window also allows you to set the lines per page and characters per line relevant to the Braille Standard being used.

Large Print Options: Most of the functionality for creating large print output is available elsewhere in the I-MAESTRO client tool. This set of options brings them together and allows settings to be specified for level of magnification, colours, fonts etc. and the printer which will be used.

Configuration files accessible as proxy objects

The General overview for the specifications (DE3.1.1a) highlights the need for very adaptable and scalable user requirements, as recommended in IMS white papers.[1] In order to achieve this, the Accessible User Interface of the I-MAESTRO client tool makes extensive use of XML technologies to ensure that these parameter controls are well structured and can be reused to provide for future user needs.

For example, the Talking Music template file containing the overall settings of the Talking Music interpretation and production process reads as follows:

```
<SPOKEN_MUSIC_TEMPLATE>
        <GENERAL
                lang="English"
                interpret_level="Custom"
                default_note="Eighth"
                GUI-lang="NLD">

                <PROD_ID></PROD_ID>
                <TITLE></TITLE>
                <COMPOSER></COMPOSER>
                <CREATOR></CREATOR>
                <PUBLISHER></PUBLISHER>
                <COPYRIGHTS></COPYRIGHTS>
                <NOTES/>
                <OUT_PATH>c:\AccessibleMusic\TalkingMusic\Output</OUT_PATH>
                <OUT_FILE_NAME></OUT_FILE_NAME>
        </GENERAL>
        <MUSIC_EXCERPT
                ignore_last_measures="true"
                fragment_length="4"
                use_transposition="false"/>
        <LYRIC
                display_lyric="false"/>
        <OUTPUT_FORMAT
                format="HTML 4.0"/>
        <AUDIO_FILES_FORMAT/>
        <SPEECH
                voice=""
                rate="-2"
                pitch="0"/>
        <MIDI
                qtempo_whole="120"
                qtempo_section="120"
                instr="0"/>
        <ADDITIONAL
                skin="NoSkin">
                <WORK_PATH>c:\AccessibleMusic\TalkingMusic\Work</WORK_PATH>
                <WORK_PARAMS>TS_Additional_WorkParams_Combo</WORK_PARAMS>
        </ADDITIONAL>
</SPOKEN_MUSIC_TEMPLATE>
```

This process is then repeated for all user requirements which are fed to the Accessible Music decoder by the I-MAESTRO Client tool.

Accessible User Interfaces

In order that the I-MAESTRO client tool is accessible software, several design considerations have to be taken into account in the system architecture to ensure that standard accessibility controls and components are included within at least the modules used for Accessibility. Guidelines for doing this are specified in DE3.1.1a, but the system should take into account the use of:

[1] http://www.imsglobal.org/accessibility/accwpv0p6/imsacc_wpv0p6.html

- Textual alternative descriptions of any icons, images, access buttons, text field labels, menu item and other components used for accessing functionality within the I-MAESTRO client tool
- Alternative means to access functionality other than with a mouse i.e. ensure that all actions can be completed from the keyboard.
- The standard components available for the operating system or development platform, or follow accessibility guidelines on how to create custom controls.
- Provide help files, including an orientation to the interface and its functionality.

The following should be avoided:

- Indexing or navigation systems using complex frames that do not have the title or name attribute.
- Tables of contents with expand/collapse features that are indicated with images with no text labels (i.e., blue triangles or plus and minus signs).
- Menu bars built in non standard scripting languages.

In order to meet these requirements fully, it may be required to create several interface specific modules within Max/MSP which can provide some of these functionalities:

Text 2 speech

If some or all of the screen reading functionalities need to be handled by the Max/MSP environment, the re is an external available called “Flite” which may be useful for adapting as a text2speech functionality:

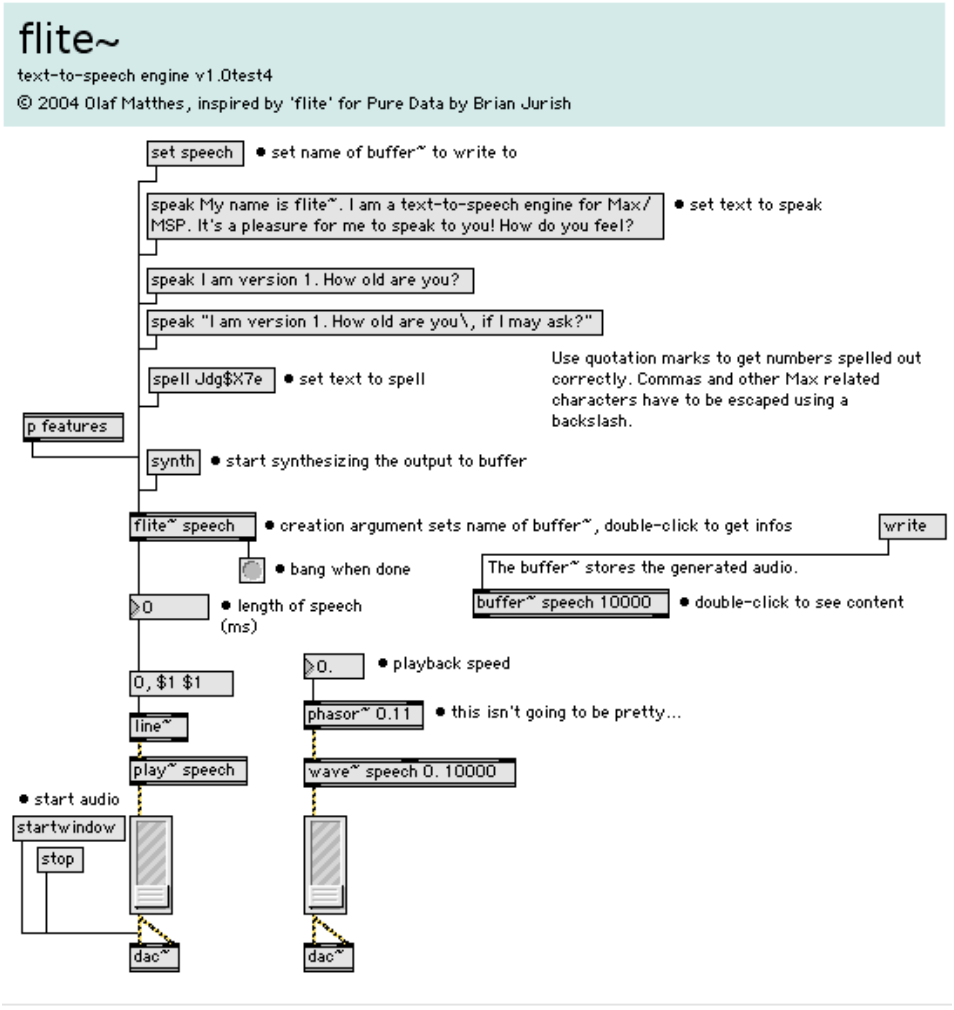


Figure: An example of functionality which could be used to create screen reading functionality within Max/MSP

Braille Bars

In order to incorporate Braille Bars within Max/MSP environments, the functionalities for controlling midi devices can be used. This may require different set ups depending on the manufacturer of the Braille Bar. If this is the case, several open source drivers for Braille bars are available and functionality can be provided for the main manufacturers. An interfacing dialog can be provided in the I-MAESTRO client tool to choose specific setups for such devices. (Number of characters per line, Serial port, USB connecting, etc.)

Keyboard Shortcuts

Keyboard shortcuts are integral component of accessible interface design. If the system specific keyboard shortcuts are not used by the Native OS in question, decisions will have to be made as to which keyboard shortcuts to use. There are standards for this.[2]

[2] http://accessibility.freestandards.org/

6 Professional Music Editing

Sibelius manages notation content using a proprietary and encrypted file format optimized for efficient storage and rendering of musical notation. It features built-in import facilities from a variety of alternative formats, namely

- MusicXML
- Midi
- NIFF
- Score
- Finale ETF

Native file writing is confined to Sibelius formats (including formats readable by earlier versions of Sibelius) and to MIDI output.

Sibelius implements a plug-in mechanism for scripting additions to core Sibelius functionality. This is done by means of functions written in a scripting language – Manuscript – developed specifically for Sibelius. The elements of the notated content are made available to Manuscript via built-in objects (in the same way as, for example, the contents of an HTML or XML document are made available to web browser scripting via a Document Object Model or DOM). The Manuscript language is fully documented in the Sibelius distribution.

It is not planned to modify Sibelius to read and write SMR files directly. A Sibelius plug-in will be developed to export MPEG SMR from Sibelius score data. In addition, plug-ins will be developed to insert and edit aspects of MPEG SMR which are not directly supported by the Sibelius format.

A plug-in will be developed to export MPEG SMR from the structure of a Sibelius file.

6.1 Using SIBELIUS for MPEG SMR production

The workflow for MPEG SMR production from Sibelius necessitates a number of steps:

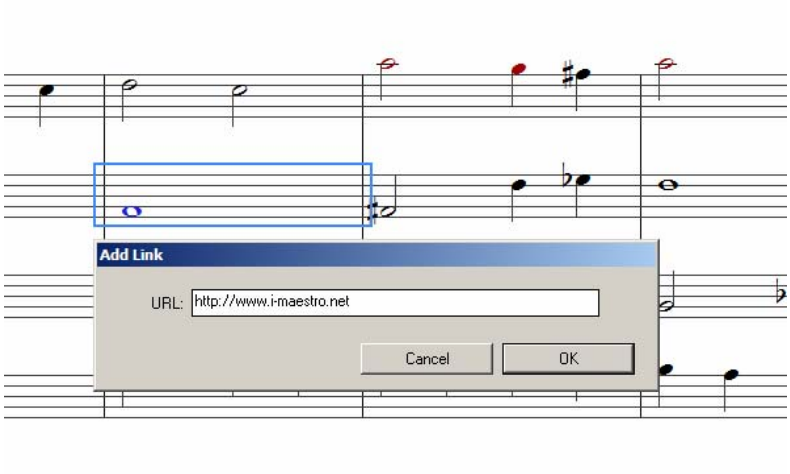
- Author or edit content in Sibelius
  - Use special text tags and/or styles to embed content for specific SMR processing
- Export MPEG SMR using Sibelius-SMR plug-in
- Perform further editing and assembly of content using the generated SMR in the other I-MAESTRO authoring and packaging tools

SMR data and annotations not directly mapped to notation features supported in Sibelius will be implemented using a special text style and/or tags identifying SMR content, which will be preserved in export and processed by the conversion utility to generate the relevant SMR content.

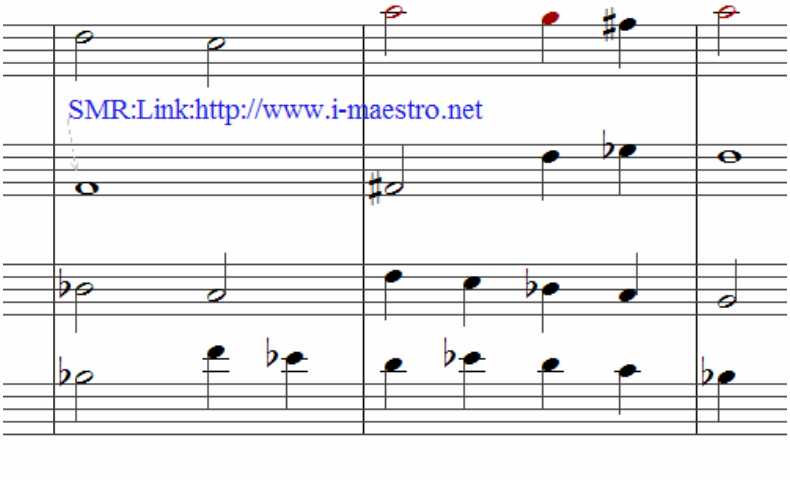
6.2 Plug-ins for MPEG SMR annotation in Sibelius

A set of plug-ins will be developed to manage SMR content not directly represented in a Sibelius score. The content will be represented as text, with a specific prefix scheme identifying the tag as an aspect of SMR content, and further identifying the type of content.

For example, Sibelius does not currently support URL links in a score. While these could be added as plain text, to support the level of annotation required to perform a mapping to an MPEG SMR link a simple plug-in to insert and edit HTML links might be look like this:



With an annotation added to the score as follows:

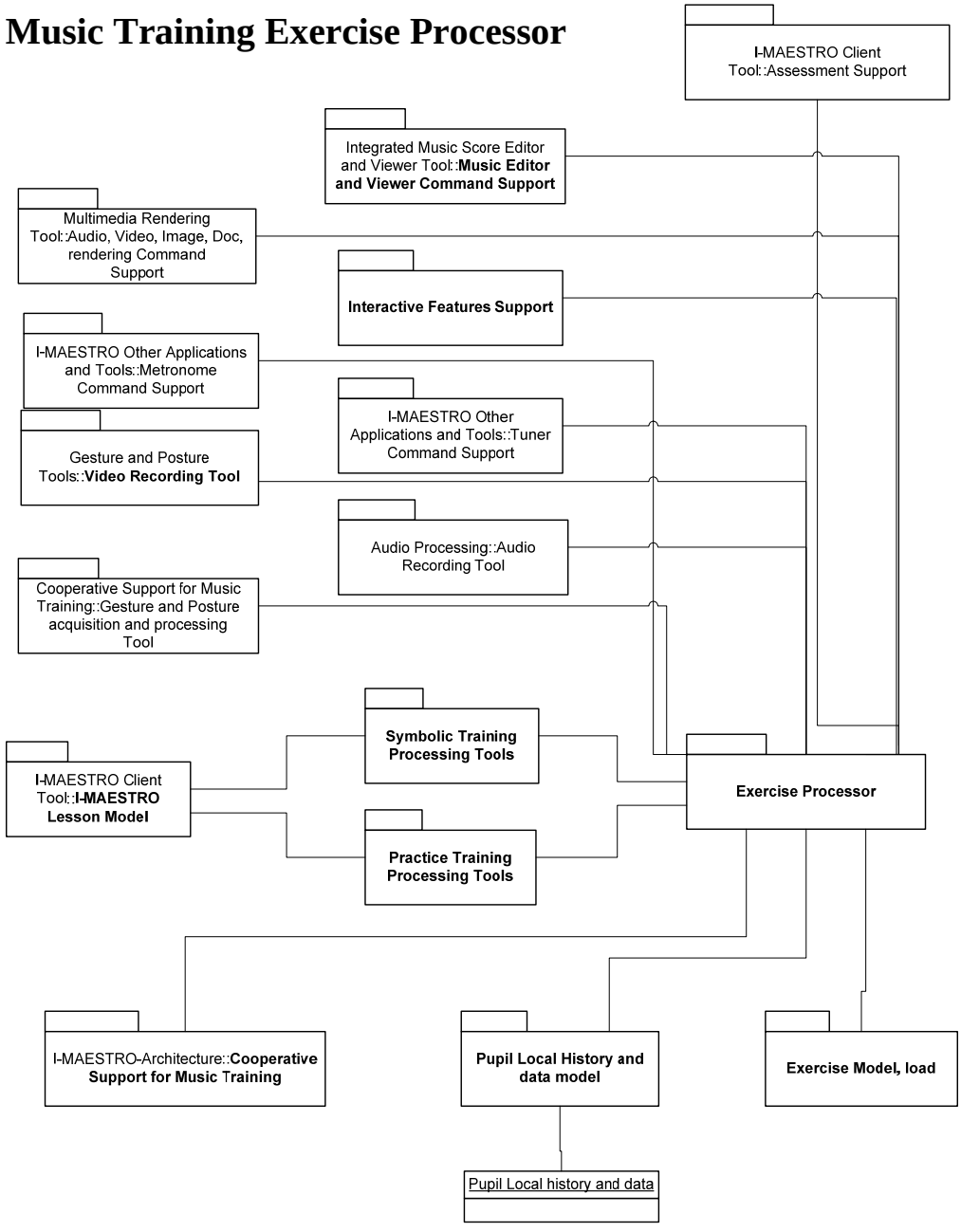


For all such plug-ins

- The text schema for the annotation is MPEG SMR:<type>:<content>
- The plug-in will operate on a selected passage or single score element
- A single instance of the annotation will be placed at the same position and associated with the same bar as the selected element or at the start of the selected passage
- If any of the plug-ins is invoked when an existing SMR annotation is selected, the appropriate plug-in for the type of annotation is loaded and the content of the annotation is made available for editing in the plug-in
- A new text style (MPEG SMR-Annotation, that is implemented as a SMR Selection) is created to allow MPEG SMR annotations to be distinguished graphically in the score

7 Specification of I-MAESTRO Music Exercise Authoring Tool

7.1 Music Training Exercise Processor General Overview



The Music Training Exercise Processor processes/executes the Music Exercise Formalisation taking into account the profile, history, and pupil's results step by step. It has to be capable of synchronising and to be synchronised by external events for the rendering and visualization of results and eventually with other tools via Cooperative Support for Music Training.

Music Training Exercise Processor communicates with:

- Cooperative Support for Music Training to communicate with other Music Training Exercise Processors. Most of the commands for the other tools of I-MAESTRO Client Tool can be issued to them from another Music Training Exercise Processor via the middleware module of Cooperative Support for Music Training.
- Assessment support: applying assessment models taking into account profile, exercise, context, audio processing results, symbolic processing results, sensors, gesture and posture, etc.
- Command Support of others tool (i.e. Music and Editor and Viewer Command Support, Metronome Command Support, Audio Video Image Doc rendering Command Support).
- Gesture and Posture acquisition and processing tool.
- Audio Recording Tool.

Also the Music Training Exercise Processor comprehends:

- Symbolic Training Processing Tools: supporting symbolic score analysis, comparison, searching, navigation, annotation, etc.
- Practice Training Processing Tools: supporting symbolic score analysis versus audio processing results, comparison, searching, navigation, etc.
- Exercise Processor that executes commands present in the Exercises.
- Interactive Features Support, which implement chat and file sharing.
- The Exercise Model, which provides the structure of Exercises.

7.2 Music Training Exercise Processor processing the logic of the lesson Formal Model

Specified in Max/MSP formalism.

7.3 Integrated or inherited Multimedia Rendering Tools

There are several Multimedia rendering tools available in Max/MSP. Here we provide a list of the principal:

For video and audio rendering Max/MSP has two objects:

1. movie objects
2. imovie objects.

The *movie* object plays a QuickTime movie in its own window, and the *imovie* object plays a QuickTime movie in a box inside a patcherwindow. They can receives from their Inlets commands such as start, stop, pause, read, open, next movie and many others. This objects can open at least 26 different types of files that can be managed by QuickTime, these include movie files such as MPEG, audio files including AIFF and MP3, and graphics files including GIF and JPEG. To know all available commands it is possible to reference to the Max/MSP manual.

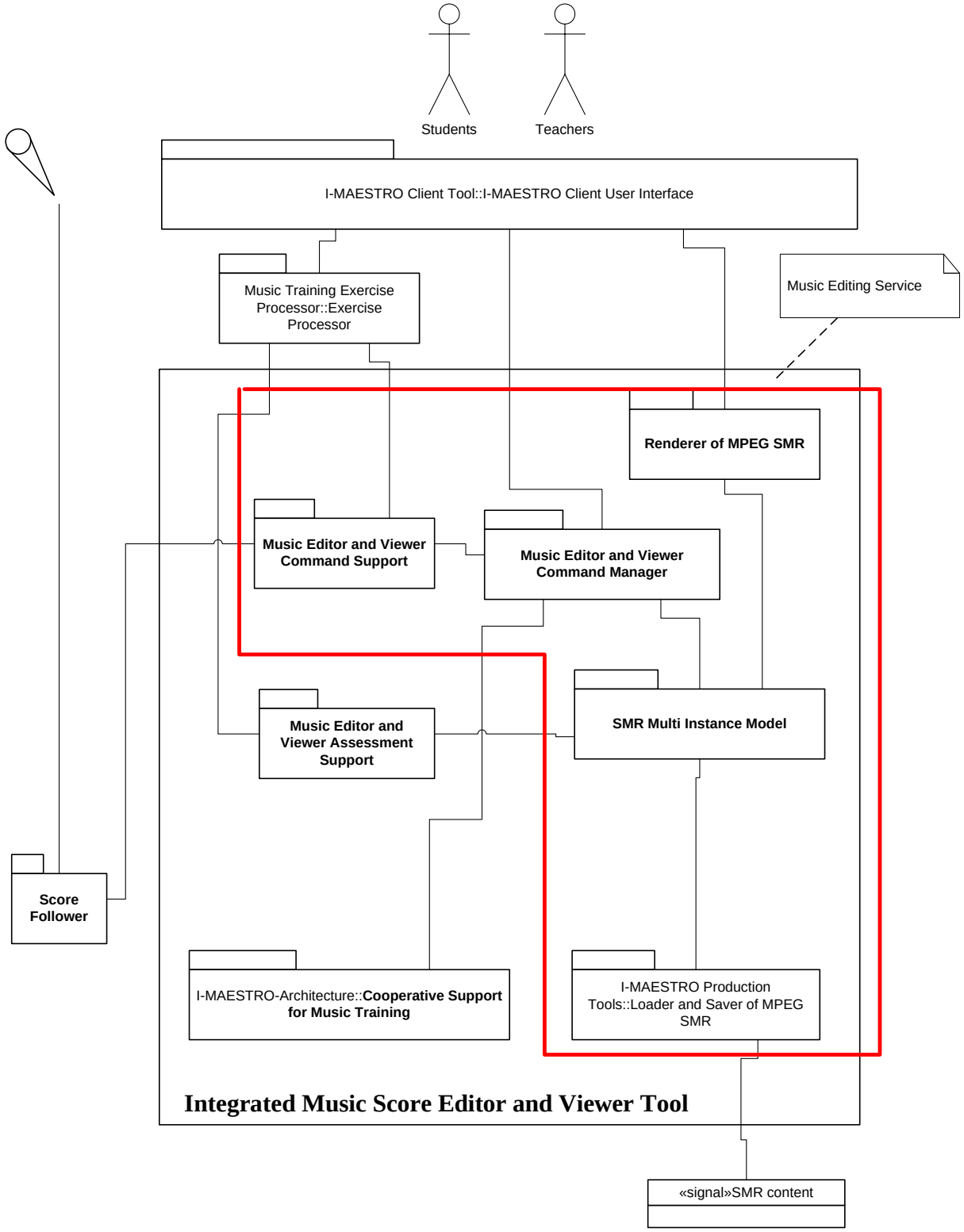
As regard pictures and images, in addition to the previously objects, there are *fpic*, *pic*, *pictr*,*l graphic* that can draw an image. The available image format are pict and pic (default format fro Max/MSP) and also formats provided by QuickTime when it is installed.

Regarding the audio, Max/MSP provides much support and many objects to manage MIDI audio, such as *midin*, *midiout*, *midiinfo*, *midiparse*, *midiformat*, *midiflush*…). There is also an object to play and record midi data called *seq*.

There are also two type of metronome objects (*metro* and *qmetro*), but they don't have a GUI. It is possible to use some MIDI objects to realize the structure of a tuner albeit without GUI.

8 Specification of Integrated Music Score Editor and Viewer Tool

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 Teacher 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 comprehends 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 represent the score loaded from the Loader and Saver of MPEG SMR.

8.1 Music Editing Service

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.

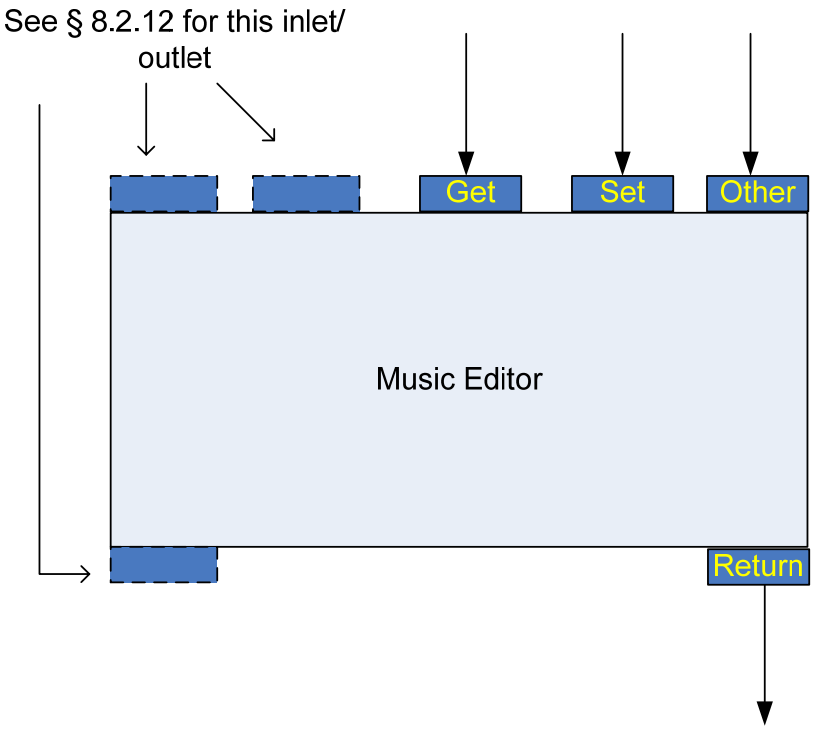
Teacher or Coordinator Student can choose to save cooperative work at any time; in this case all group members receive new document version and the change log is reset.

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:



8.1.1 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 then 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

setArgumentOnExecute	
Method	setArgumentOnExecute
Description	It indicates arguments for the commandOnExecute command.
Input parameters	String array of arguments (depending on the type of command)
Output parameters	None

setCommandOnExecute	
Method	setCommandOnExecute
Description	It indicates the command to be executed by the user
Input parameters	string command_id – command to execute
Output parameters	None
List of possible	• "ADD_TEXT_ANNOTATION" the first value in *argumentsOnExecute* contains the text to be added to the score in the

commands	position where the user will click via the *mouseOnExecute* • ”ADD_LABEL" the first value in *argumentsOnExecute* contains the label text to be added to the measure where the user will click via the *mouseOnExecute* if the measure already has a label the label is substituted • "ADD_NOTE" the first value in *argumentsOnExecute* 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. • "ADD_REST" the first value in *argumentsOnExecute* 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 *mouseExecute*. • "SET_ALTERATION" the first value in *argumentsOnExecute* 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 *mouseExecute*. • "SET_DOTS" the first value in *argumentsOnExecute* 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 *mouseExecute*. • "ADD_SYMBOL" the first value in *argumentsOnExecute* 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 *mouseExecute*. • "ADD_MEASURE" adds a measure to the score, the first value in *argumentsOnExecute* can be: "BEFORE", "AFTER" or "APPEND", the second value in *argumentsOnExecute* indicates the measure with respect to the new measure is added. The second value is necessary only for *execute* eventIn and it is not necessary for *mouseClickExecute* in fact in this case the measure where the user clicks indicates the measure with respect to the new measure is added. • "DEL_MEASURE" removes a measure of the score; the first value in *argumentsOnExecute* indicates the measure number to be removed. The value is necessary only for *execute* eventIn and it is not necessary for *mouseClickExecute* in fact in this case the measure where the user clicks indicates the measure to be removed. • "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 *argumentsOnExecute* contains the clef type, it can be: "TREBLE", "SOPRANO", "BASS", "TENOR"…, The clef change applies to the measure where the user clicks via the *mouseExecute* • "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 *argumentsOnExecute* 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 *mouseExecute* • "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 *argumentsOnExecute* 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 *mouseClickExecute* • "SET_METRONOME" sets the metronome for the whole piece. The first value in *argumentsOnExecute* 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 *execute* eventIn. • "DELETE" allows deleting any symbol, note, rest, alteration, label and annotation added by the user in the position where the user clicks via the *mouseExecute*

setFirstVisibleMeasure	
Method	setFirstVisibleMeasure
Description	It sets the first measure currently visible
Input parameters	Int number_of_measure
Output parameters	None

setHyperlinkEnable	
Method	setHyperlinkEnable
Description	When it is set to 1 hyperlinks are shown; when the user clicks on a link an **activatedLink** is generated
Input parameters	Int 1 to activate Hyperlink 0 otherwise
Output parameters	None

setPartLyrics	
Method	setPartLyrics
Description	It is an array of strings indicating for which part to view the lyrics and in which language (e.g. ["it", "en", ""] to view lyrics for part 1 in Italian and for part 2 in English)
Input parameters	String array of language for part lyrics (the position represents the number of part, the string represents the language)
Output parameters	None

setPartShown	
Method	setPartShown
Description	It is an array of integers indicating which parts have to be shown; the number is the position in the array of parts names; if **partShown** is empty all parts will be visible (e.g. [ ] to view main score with all parts, [2] to view single part number 2, [1,3] view main score with parts 1 and 3, etc.).
Input parameters	Int array (number of the parts to show)
Output parameters	None

setScoreOffset	
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. **scoreOffset** is indicated in seconds from the beginning of the score. **scoreOffset** can be used only if synchronization information is provided or a metronome indication is present in the score.
Input parameters	Float time
Output parameters	None

setChronometricPosition	
Method	setChronometricPosition
Description	It sets the chronometric position (in millisecond) in the SMR object
Input parameters	Float position
Output parameters	None

setComputerViewParams	
Method	setComputerViewParams
Description	It sets params for ComputerView
Input parameters	Short Top Short bottom Short left Short right Short staff
Output parameters	None

setMetricPosition	
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	Unsigned long int
Output parameters	None

SetPrintViewParams	
Method	SetPrintViewParams
Description	It sets params for PrintView
Input parameters	Short nPage Short top Short bottom Short left Short right Float magnify Short linelenght Short nStaffs Short nSystems Short Distance
Output	None

parameters	

setSize	
Method	setSize
Description	It expresses the width and height of the music score in the units of the local coordinate system. A size of -1 in either coordinate means that the **MusicScore** node is not specified in size in that dimension, and that the size is adjusted to the size of the parent node.
Input parameters	Vec2f (array of two floats)
Output parameters	None

setSpeed	
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 speed
Output parameters	None

setViewType	
Method	setViewType
Description	It indicates the kind of view to be used (one of the *availableViewTypes*).
Input parameters	String view_type
Output parameters	None

8.1.2 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 Musice Edtior 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 parameters (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**

getArgumentOnExecute	
Method	ArgumentOnExecute
Description	It returns arguments for the current commandOnExecute command.
Input parameters	None
Output parameters	String array of arguments

getAuthor	
Method	getAuthor

Description	It returns the author of the SMR loaded
Output parameters	String author

getAvailableCommands	
Method	getAvailableCommands
Description	It gives an array 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	String array of available commands

getAvailableLabels	
Method	getAvailableLabels
Description	It gives an array of strings with labels (e.g. ["A", "B", "SEGNO", "CODA"]).
Input parameters	None
Output parameters	String array of labels

getAvailableLyricLanguages	
Method	getAvailableLyricLanguages
Description	It gives an array of strings where for each part there is the list of languages (using the ISO 639-2 standard), separated with ";", for which the lyric is available (e.g. ["en;it", "en;it", ""]) (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	String array of LyricLanguages

getAvailableViewTypes	
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	String array of ViewTypes

getChronometricPosition	
Method	getChronometricPosition
Description	It provides the present chronometric position (in millisecond) in the SMR object
Input parameters	None
Output parameters	Float position

getCommandOnExecute	
Method	getCommandOnExecute
Description	It returns the current command set with the setCommandOnExecute method
Input	None

parameters	
Output parameters	string command_id – current command set to execute
List of possible commands	• "ADD_TEXT_ANNOTATION" • ”ADD_LABEL" • "ADD_NOTE" • "ADD_REST" • "SET_ALTERATION" • "SET_DOTS" • "ADD_SYMBOL" • "ADD_MEASURE" • "DEL_MEASURE" • "CHANGE_CLEF" • "CHANGE_KEYSIGNATURE" • "CHANGE_TIME" • "SET_METRONOME" • "DELETE"

getFirstVisibleMeasure	
Method	getFirstVisibleMeasure
Description	It returns the first measure currently visible
Input parameters	None
Output parameters	Int number_of_measure

getHighlightPosition	
Method	getHighlightPosition
Description	It outputs the highlight position in local coordinates.
Input parameters	None
Output parameters	Vecf3 (0,0,0) (array of three floats)

getHyperlinkEnable	
Method	getHyperlinkEnable
Description	When it is set to 1 hyperlinks are shown; when the user clicks on a link an **activatedLink** is generated
Input parameters	None
Output parameters	Int 1 if Hyperlink is activated 0 otherwise

getLastVisibleMeasure	
Method	getLastVisibleMeasure
Description	It is the last measure currently visible
Input parameters	None
Output	Int number_of_measure

parameters	

getMetricPosition	
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	Unsigned long int

getMousePositionOnExecute	
Method	getMousePostionOnExecute
Description	It is used to indicate the point where the user has clicked, the position will be considered when the **executeCommand** will be issued.
Input parameters	None
Output parameters	Vecf3 (0,0,0) (array of three floats)

getNumMeasure	
Method	getNumMeasure
Description	It gives the number of measures in the score.
Input parameters	None
Output parameters	Int number_of_measure

getPartLyrics	
Method	getPartLyrics
Description	It is an array of strings indicating for which part to view the lyrics and in which language (e.g. ["it", "en", ""] to view lyrics for part 1 in Italian and for part 2 in English)
Input parameters	None
Output parameters	String array of part lyrics

getPartNames	
Method	getPartNames
Description	It gives an array of strings with part names (instruments, e.g. ["soprano", "baritone", "piano"])
Input parameters	None
Output parameters	String array of part names.

getPartShown	
Method	getPartShown
Description	It is an array of integers indicating which parts are shown; the number is the position in the array of parts names; if **partShown** is empty all parts will be visible (e.g. [ ] to view main score with all parts, [2] to view single part number 2, [1,3] view main score with parts 1 and 3, etc.).
Input parameters	None

Output parameters	Int array (number of the parts shown)

getScoreOffset	
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. **scoreOffset** is indicated in seconds from the beginning of the score. **scoreOffset** can be used only if synchronization information is provided or a metronome indication is present in the score.
Input parameters	None
Output parameters	Float time

getSize	
Method	getSize
Description	It returns the width and height of the music score in the units of the local coordinate system. A size of -1 in either coordinate means that the **MusicScore** node is not specified in size in that dimension, and that the size is adjusted to the size of the parent node.
Input parameters	None
Output parameters	Vec2f (array of two floats)

getSpeed	
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	Float speed

getSymPartName	
Method	getSymPartName
Description	It returns a description of a Symbolic Score
Input parameters	None
Output parameters	string desc – description of the score

getTitle	
Method	getTitle
Description	It returns the title of the SMR loaded
Output parameters	String title

getViewType	
Method	getViewType
Description	It returns the view used (one of the *availableViewTypes*).
Input	None

parameters	
Output parameters	String view_type

8.1.3 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	Int 1 if the command is correctly performed 0 otherwise

GoBackward	
Method	GoBackward
Description	It shows previous score page
Input	None

parameters	
Output parameters	None

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

GoBottom	
Method	GoBottom
Description	It shows the bottom of the score
Input parameters	None
Output parameters	None

GoTop	
Method	GoTop
Description	It shows the top of the score
Input parameters	None
Output parameters	None

GotoLabel	
Method	GotoLabel
Description	It positions the score on the page containing the specified label (one of the *availableLabels*).
Input parameters	String label
Output parameters	None

GotoMeasure	
Method	GotoMeasure
Description	positions the score on the page containing the specified measure
Input parameters	Int32 number of measure
Output parameters	None

HighlightTimePosition	
Method	HighlightTimePosition
Description	It highlights the time position indicated relative to the scoreOffset field
Input parameters	None
Output parameters	None

Justify

Method	Justify
Description	It justifies (logarithmic or linear) the current score depending on entry parameters
Input parameters	Short fromMeasure Short toMeasure int logarithmic (1 is logarithmic, 0 is linear)
Output parameters	None

Load	
Method	Load
Description	Load a file from disk or from a URL
Input parameters	string file – name or path of the file to load
Output parameters	Int 1 if the file is correctly loaded, 0 otherwise

Pause	
Method	Pause
Description	It pauses the current file execution
Input parameters	None
Output parameters	None

Play	
Method	Play
Description	It plays the file previously loaded
Input parameters	None
Output parameters	None

PlaySync	
Method	PlaySync
Description	It plays a synchronous file
Input parameters	None
Output parameters	None

PlaySyncFromTo	
Method	PlaySyncFromTo
Description	It executes an synchronous file from the measure x to the measure y
Input parameters	long startMeasure long endMeasure
Output parameters	None

Print

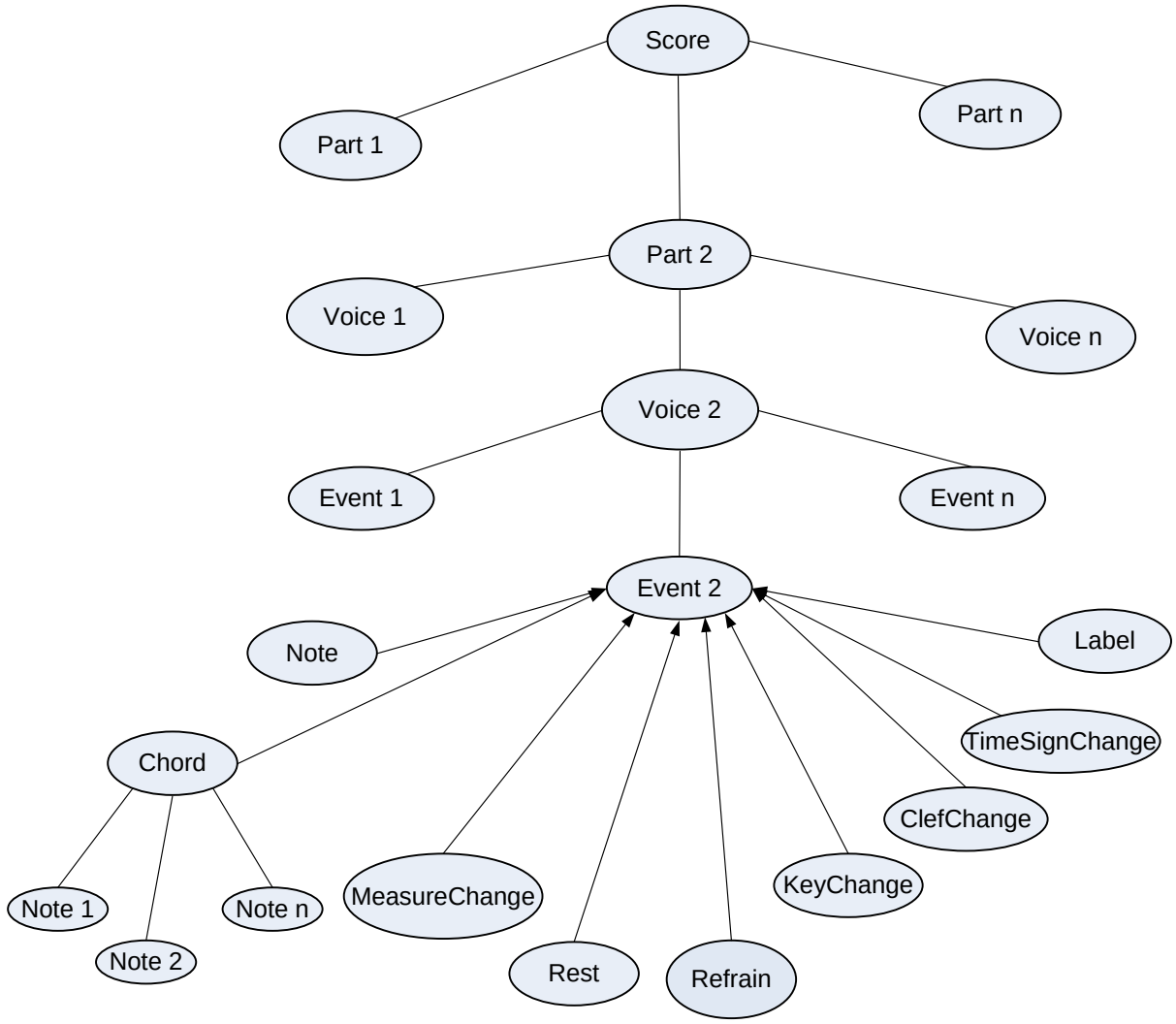
Method	Print
Description	Print the current score
Input parameters	None
Output parameters	None

Stop	
Method	Stop
Description	It stops the execution of a file loaded
Input parameters	None
Output parameters	None

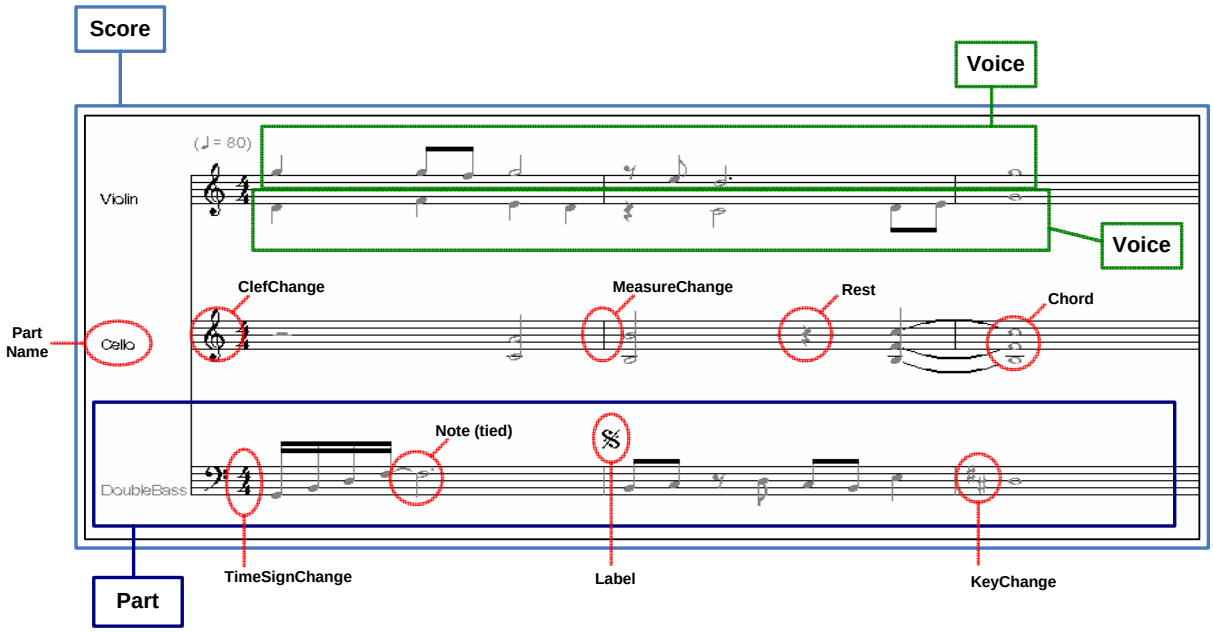
Transpose	
Method	Transpose
Description	It transposes notes in the score.
Input parameters	Short daBat – start measure to execute transposition Short aBat – final measure to execute transposition Short partnumber – number of the part to transpose Short clef – number of the clef to change Short translation – it point how much the note have to be tranlated Short interval – type of interval Short up – it points if the translation is up or down Short numofstaff – it gives the capability to transpose using more then one staff Short sharps Short adjust
Output parameters	None

8.2 Music editor and Score Follower Support

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 Musisc 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)
SetVoice(1)
For i=1,i<getNumEvents(),i++
{
        setCurrentEvent(i)
        event_type=getEventType()
        if (event_type==1) //the event is a note
                {
                        currenteNoteDuration=getDuration()
                }
}
```

8.2.1 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	Int total_parts_number

setPart	
Method	setPart
Description	It selects the part inside the score
Input	Int part_number

parameters	
Output parameters	Int 0 if no error -1 otherwise

getScoreName	
Method	getScoreName
Description	It returns the name of the score (if present)
Input parameters	None
Output parameters	String score_name

getNumVoices	
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	Int total_voices_number if no error -2 otherwise

setVoice	
Method	setVoice
Description	It selects the voice inside the part
Input parameters	Int voice_number
Output parameters	Int 0 if no error -1 otherwise

getPartName	
Method	getPartName
Description	It returns the name of the part (if present)
Input parameters	None
Output parameters	String part_name

getNumEvents	
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	Int total_events_number if no error -3 otherwise

setCurrentEvent	
Method	setCurrentEvent
Description	It selects an event inside the voice
Input parameters	Int event_number
Output parameters	Int 0 if no error -1 otherwise

getEventType	
Method	getEventType
Description	It returns the type of the event selected
Input parameters	None
Output parameters	int event_type if no error -4 otherwise

eventType	Int
Label	0
Note	1
Rest	2
MeasureChange	3
Chord	4
Refrain	5
KeyChange	6
ClefChange	7
TimeSignatureChange	8

8.2.2 Label Info

getLabel	
Method	getLabel
Description	It returns the label code (ASCII code if alfanumeric)
Input parameters	None
Output parameters	int label_type if no error -5 otherwise

refrainType	Int	Example
CODA	0	𝄌
SEGNO	1	𝄋
FINE	2	**Fine**
Letter label	ASCII(…)	**A…Z, 0...9**

8.2.3 Note Info

getPitch	
Method	getPitch
Description	It returns the pitch of note
Input parameters	None

Output parameters	String pitch_value (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, E1, C3 = middle C) if no error Empty string otherwise

getDuration	
Method	getDuration
Description	It retuns the symbolic positive note duration calculated depending on the duration of a whole note. (whole note duration = 4096)
Input parameters	None
Output parameters	Unsigned Int duration if no error -5 otherwise

Unsigned int	**Example**
8192	
4096	
2048	
1024	
512	
256	
128	
64	
32	

getIsTied

Method	getIsTied
Description	It returns 1 if the note is tied with the previous one (with the same pitch).
Input parameters	None
Output parameters	int 1 if the note is tied 0 if not and -5 otherwise

8.2.4 Rest info

getDuration	
Method	getDuration
Description	It retuns the symbolic positive rest duration calculated depending on the duration of a whole rest. (whole rest duration = 4096)
Input parameters	None
Output parameters	Unsigned Int duration if no error -5 otherwise

Unsigned int	Example
16384	
8192	
4096	
2048	
1024	
512	
256	
128	
64	
32	

8.2.5 MeasureChange info

getBPM	
Method	getBPM
Description	It define the metronome speed of the current measure
Input parameters	None
Output parameters	Unsigned int beat_per_minute if no error -5 otherwise

getDurationRef	
Method	getDurationRef
Description	It retuns the symbolic positive note duration (whole note duration = 4096) used as unit of measurement for the metronome speed set using getBPM method
Input parameters	None
Output parameters	Unsigned Int duration if no error -5 otherwise

getBarType	
Method	getBarType
Description	It returns the type of the barline of the measure
Input parameters	None
Output parameters	int bar_type if no error -5 otherwise

barType	Int	Example
SINGLE	0	
DOUBLE	1	
START_REFRAIN	2	
END_REFRAIN	3	
START_END_REFRAIN	4	

		END	5	
		INVISIBLE	6	

8.2.6 Chord info

getNumNotes	
Method	getNumNotes
Description	It returns the total number of note inside the chord ordered from the lower to the higher pitch.
Input parameters	None
Output parameters	Int total_note_number. if no error -5 otherwise

setNote	
Method	setNote
Description	It selects a note inside the chord.
Input parameters	Int note_number
Output parameters	Int 0 if no error -1 otherwise

8.2.7 Refrain info

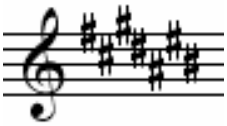
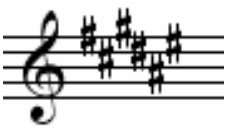
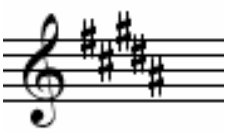
getRefrain	
Method	getRefrain
Description	It returns the type of refrain.
Input parameters	None
Output parameters	int refrain_type if no error -5 otherwise

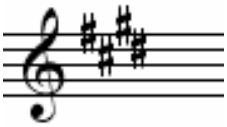
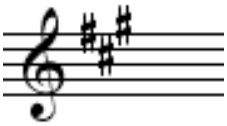
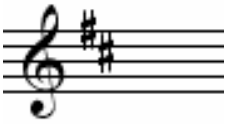
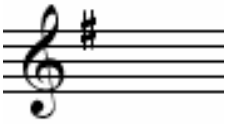
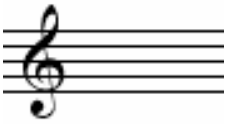
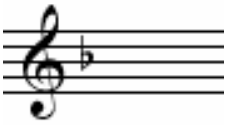
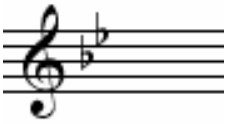
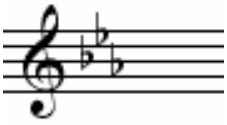
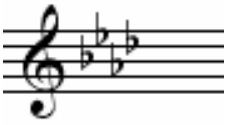
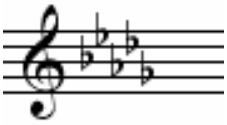
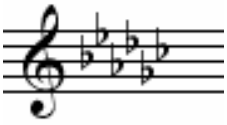
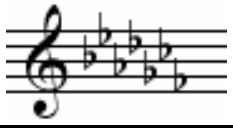
refrainType	**Int**	**Example**
DC	0	**D.C. - Da Capo**
DCAF	1	**D.C. al Fine** **Da Capo al Fine**
DS	2	**D.S. - Dal Segno**

DSAF	3	**D.S. al Fine** **Dal Segno al Fine**
DCAS	4	**D.C. al Segno** **Da Capo al Segno**
DSAC	5	**D.S. al Coda** **Dal Segno al Coda**
FIRSTTIME	6	
SECONDTIME	7	
THIRDTIME	8	
ENDTIME	9	

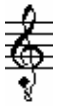
8.2.8 KeyChange info

getKey	
Method	getKey
Description	It returns the key signature type.
Input parameters	None
Output parameters	int keychange_type if no error -5 otherwise

KeyChangeType				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		

8.2.9 ClefChange info

<table>
<tr><th colspan="2">getClef</th></tr>
<tr><td>Method</td><td>getClef</td></tr>
<tr><td>Description</td><td>It returns the clef signature type.</td></tr>
<tr><td>Input parameters</td><td>None</td></tr>
<tr><td>Output parameters</td><td>int clef_type if no error
-5 otherwise
<table>
<tr><th>clefType</th><th>Int</th><th>Example</th></tr>
<tr><td>BARITONE</td><td>0</td><td>  </td></tr>
<tr><td>BASS</td><td>1</td><td>  </td></tr>
<tr><td>BASSOLD</td><td>2</td><td>  </td></tr>
<tr><td>ALTO</td><td>3</td><td>  </td></tr>
<tr><td>MEZZOSOPRANO</td><td>4</td><td>  </td></tr>
<tr><td>SOPRANO</td><td>5</td><td>  </td></tr>
<tr><td>TENOR</td><td>6</td><td>  </td></tr>
<tr><td>TENOR8</td><td>7</td><td>  </td></tr>
<tr><td>TREBLE</td><td>8</td><td>  </td></tr>
<tr><td>TREBLE8</td><td>9</td><td>  </td></tr>
<tr><td>8TREBLE</td><td>10</td><td>  </td></tr>
<tr><td>BASS8</td><td>11</td><td>  </td></tr>
<tr><td>8BASS</td><td>12</td><td>  </td></tr>
<tr><td>EMPTY(questo non è un cambio chiave)</td><td>13</td><td>  </td></tr>
<tr><td>PERCUSBOX</td><td>14</td><td>  </td></tr>
<tr><td>PERCUS2LINES</td><td>15</td><td>  </td></tr>
</table></td></tr>
</table>

		TAB	16	

8.2.10 TimeSignChange info

getTimeNum	
Method	getTimeNum
Description	It returns the numerator part of the time signature
Input parameters	None
Output parameters	Int numerator if no error -5 otherwise

getTimeDen	
Method	getTimeDen
Description	It returns the denominator part of the time signature
Input parameters	None
Output parameters	Int denominator if no error -5 otherwise

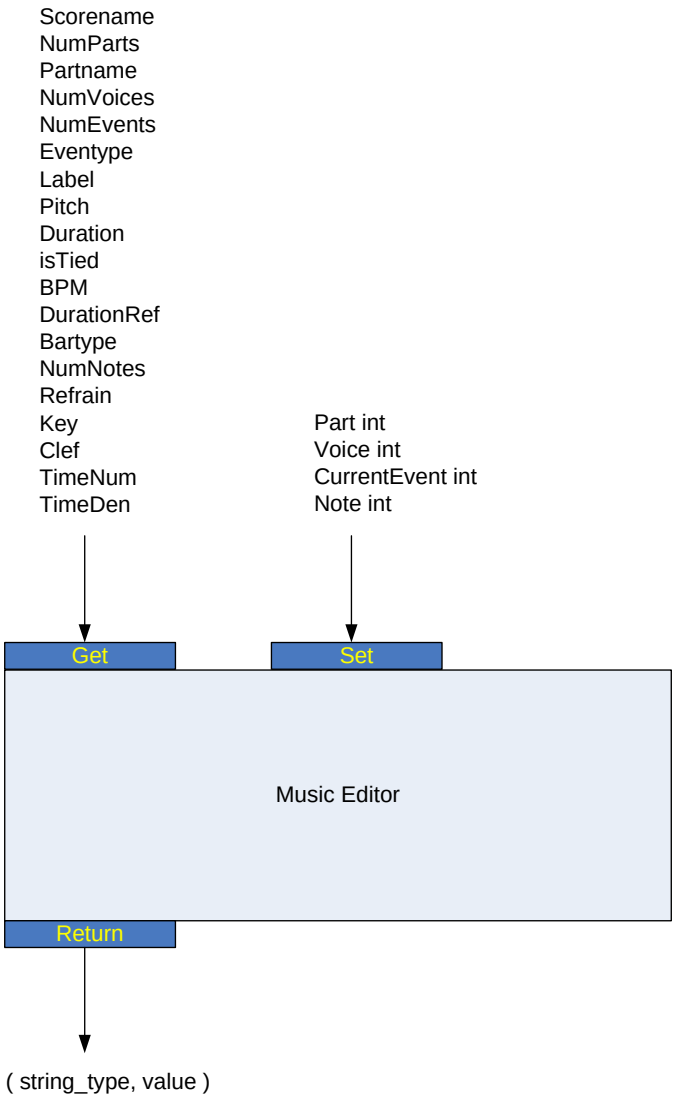
8.2.11 Error codes description

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

Description	**Code (Int)**
Out of bounds	-1
Part not set	-2
Voice not set	-3
Current event not set	-4
Wrong type event	-5

8.2.12 Interaction with Music Editor in Max/MSP

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 desribed. 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			

8.3 Music editor and Viewer Assessment Support

The main goal of assessment for theory exercises is to check/test the student’s understanding of musical structure and compositional procedures thought recorded and notated examples, with emphasis on the listening abilities (recognition and understanding of melodic and rhythmic patterns, compositional techniques, harmonic functions, etc).

Assessment support for theory exercises will be done through:

- written exercises (multiple choice questions, essays, etc.)
- written musical assignments (dictation, harmony etc.)
- listening exercises
- performance exercises
-

Key points

- Number of exercises classified in different topics
  - Music history, styles, etc.
  - Notation (rhythms and meters, scales, modes, intervals, chords, clefs, key signatures, transposition, etc.)
  - Elementary theory: identify intervals, scales, chords, cadences, rhythms, etc
  - Harmony/composition: composition of a bass line for a given melody, realization of a figured bass, four voices realization, write down chord progressions, etc.

  - Dictation: pitch training, rhythms,
  - Score analysis (identification of types of cadences, roman-numeral and figured-bass analysis, modulations, etc.
  - Formal procedures analysis (phrase structure, small forms etc.)
- Suitable for different ages and knowledge/levels
- Assessment structure organized in levels of increasing difficulty
- Monitors student progress: Possibility to get feedback/reply to the actions identifying strengths and weaknesses
- Exercises to test the student's ability to develop or organise different material, ideas, etc.
- Possibility of interactive testing.
- Different designed test for classical, electronic, jazz, specific twentieth-century techniques, etc.
- Exercises under timed conditions
- Suitable in different scenarios: classroom setting, self training, cooperative work, etc.
- Possibility to the teachers to create the assessment tests.

8.3.1 Practice training

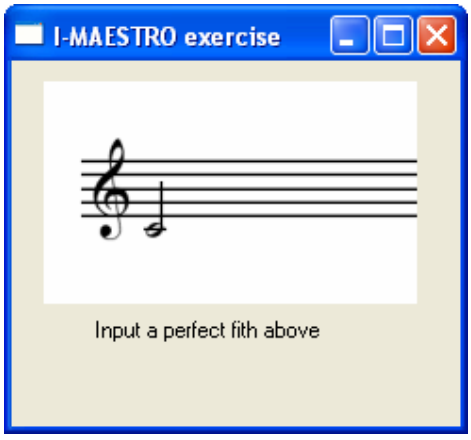
Evaluation and assessment of practice training is supported by other I-MAESTRO features. In the music editor particular annotations could be used. Nevertheless some characteristics of symbolic training assessment could apply to practice training as well.

8.3.2 Symbolic training

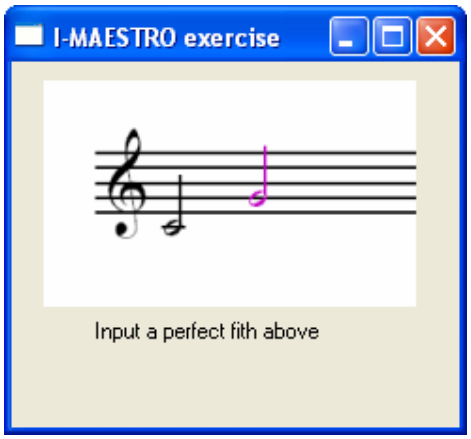
In the assessment of a symbolic training two levels should be distinguished.

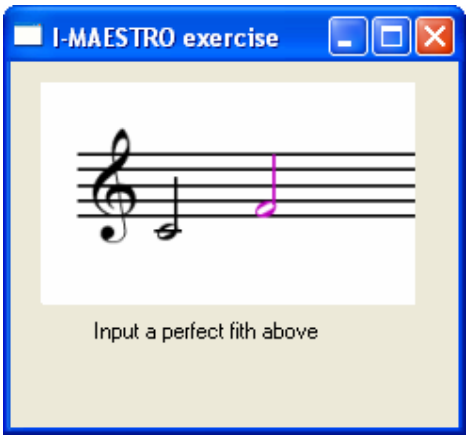
1. Error detection
2. Evaluation and assessment

1. Error detection is strongly connected to the kind of exercise and the model used. In the case of binary (e.g. yes/no) exercise or simple unique possibility answers error detection is quite easy. E.g. ear training, intervals. The exercise consists in writing the note at a given interval:



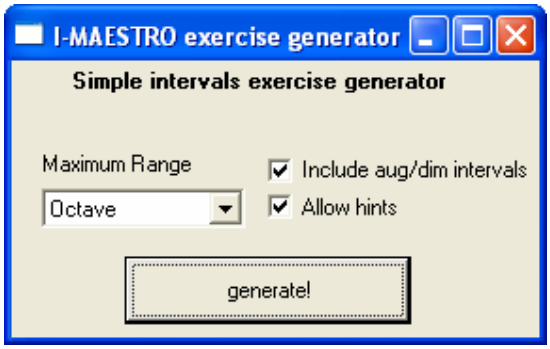
In this case there is only one possible answer. An algorithm which "knows" intervals will check if the answer is correct and detect a possible mistake:



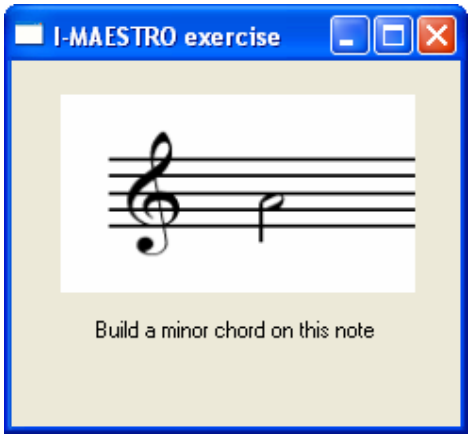


+7 semitones = perfect fith above: correct! 1+5 semitones= perfect fourth: wrong!

Furthermore this kind of exercise could be generated automatically by the exercise generator, once the model has been defined, relieving the teacher from a tedious and repetitive work.



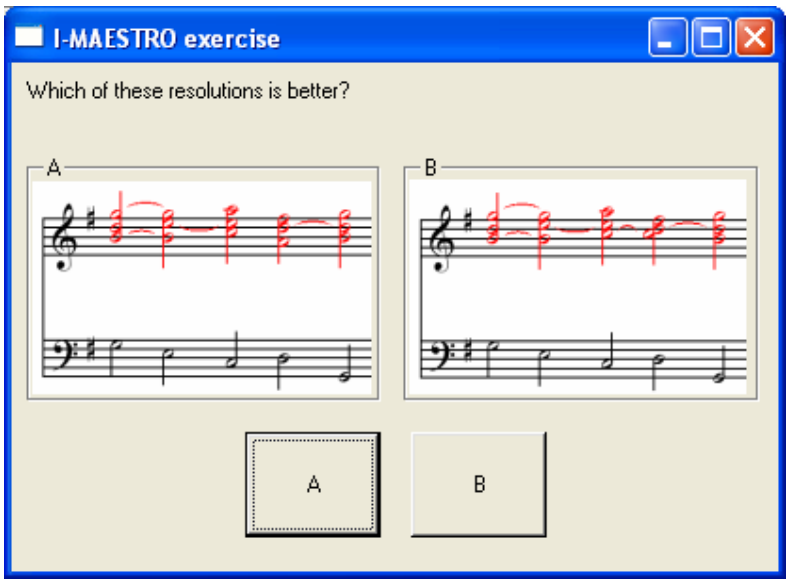
Adding just a slight complexity to the exercise adds complexity to the detection of possible error, nevertheless an algorithm can still be used in many cases:



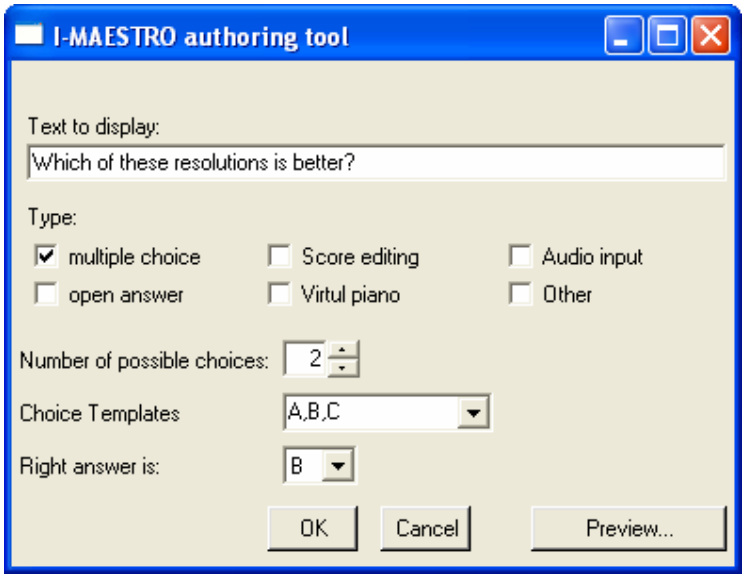
For tonal harmony exercises, although some algorithms have been implemented[3] the final and main supervision by the teacher is necessary; furthermore tonal harmony rules have changed and change continuously.

[3] A good example is the harmony training software 'Harmony Practice': http://membres.lycos.fr/mbaron/hp.htm.

In such cases, especially where issues which are not strictly technical are involved (like aesthetics), the teacher will want to decide the exact answer according to her/his pedagogical model and conventions: so in the authoring process s/he will have the possibility to set the answer(s).



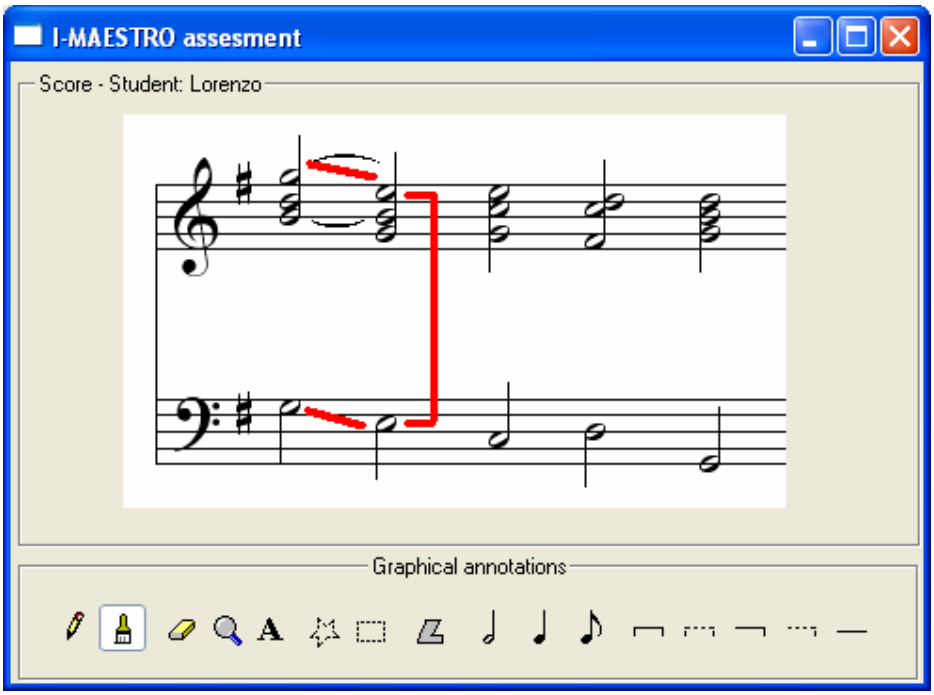
Student's view



Teacher's view in the authoring tool

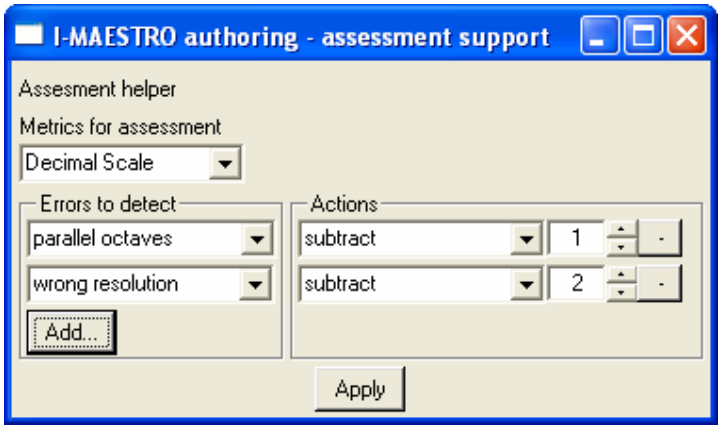
Of course if the right answers have been set by the teacher, the error detection will be quite simple.

For some kind of exercise the teacher may also decide to detect manually the mistakes, for example through annotations.

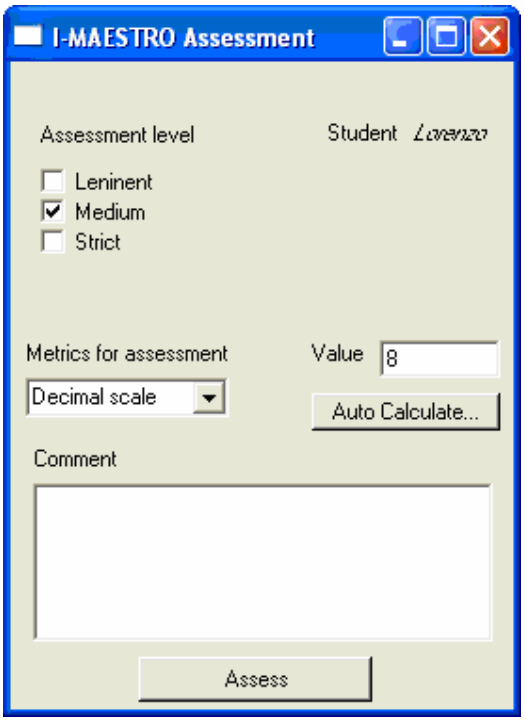


2. Evaluation and assessment

Once errors have somehow been detected, the teacher has to decide what will be the consequences especially in terms of assessment. Some parameters could be entered during the authoring process to assist in the assessment where error detection algorithms are present.



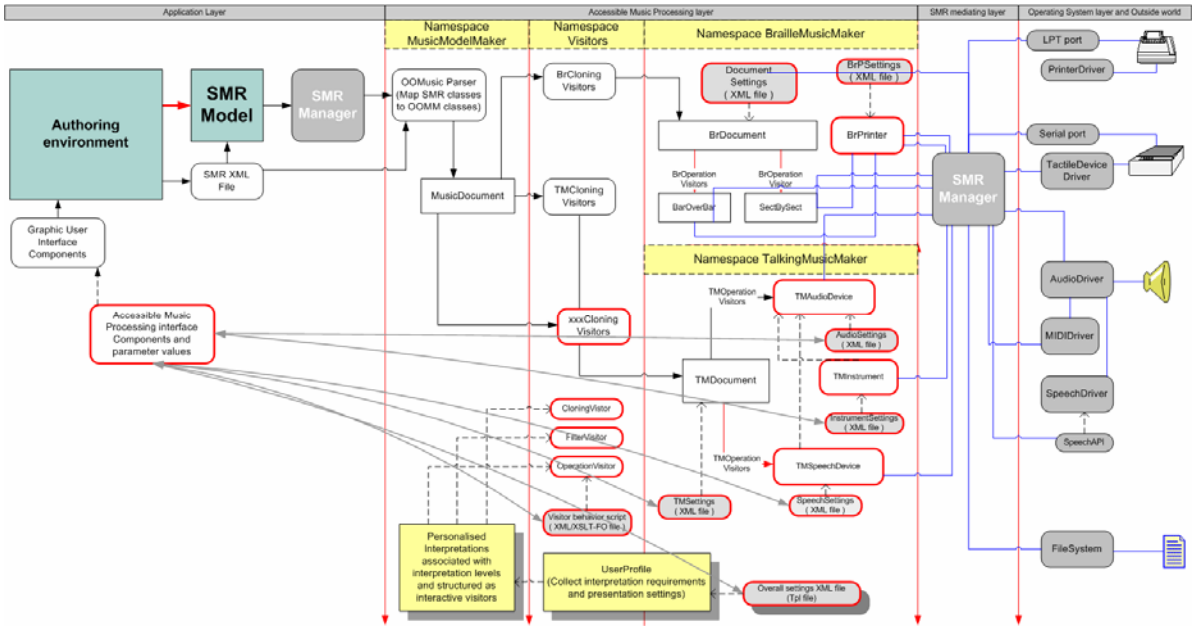
The final decision for theory training assessment should be left to the teacher, although in some cases (such as the simple interval exercise) some form of automatic calculation would be provided.



8.4 Music editor and Viewer Tool Accessible User Interface

The Accessible Music Notation Module within the I-MAESTRO client tool

Document DE3.1.1a provides an overview of the general design requirements behind designing a tools and systems for including Accessible Notation decoders within the I-MAESTRO framework. The overall structure of such an Accessible Music decoder is shown below:



The I-MAESTRO framework sends an I-MAESTRO lesson (in the SMR format) to the Accessible Music Processing module. This along with parameter values and user preferences sent by the I-MAESTRO client

tool are used to create a user personalised accessible notation in the required format (Talking Music, SVG, Braille).

Accessible presentation of Structured music representations

The accessibility component of the I-MAESTRO client tool essentially provides an accessible presentation of the SMR based content within I-MAESTRO lessons. In order to provide this accessible presentation of the musical information, the main adaptations take place in a presentation layer.

In order to keep the possibility of flexibility in formatting and cooperation with other components, a Object Oriented Document Model is used. The purpose of such a model is the possibility of *domain independent* presentation modelling. In other words: The content of the Music Model does not effect its presentation directly, but it can be changed by the presentation layer. Also the behaviour of the separate document classes can be specialised by means of inheritance, to provide the more specialised features required by the different output formats. In the figure below, a class tree is shown:

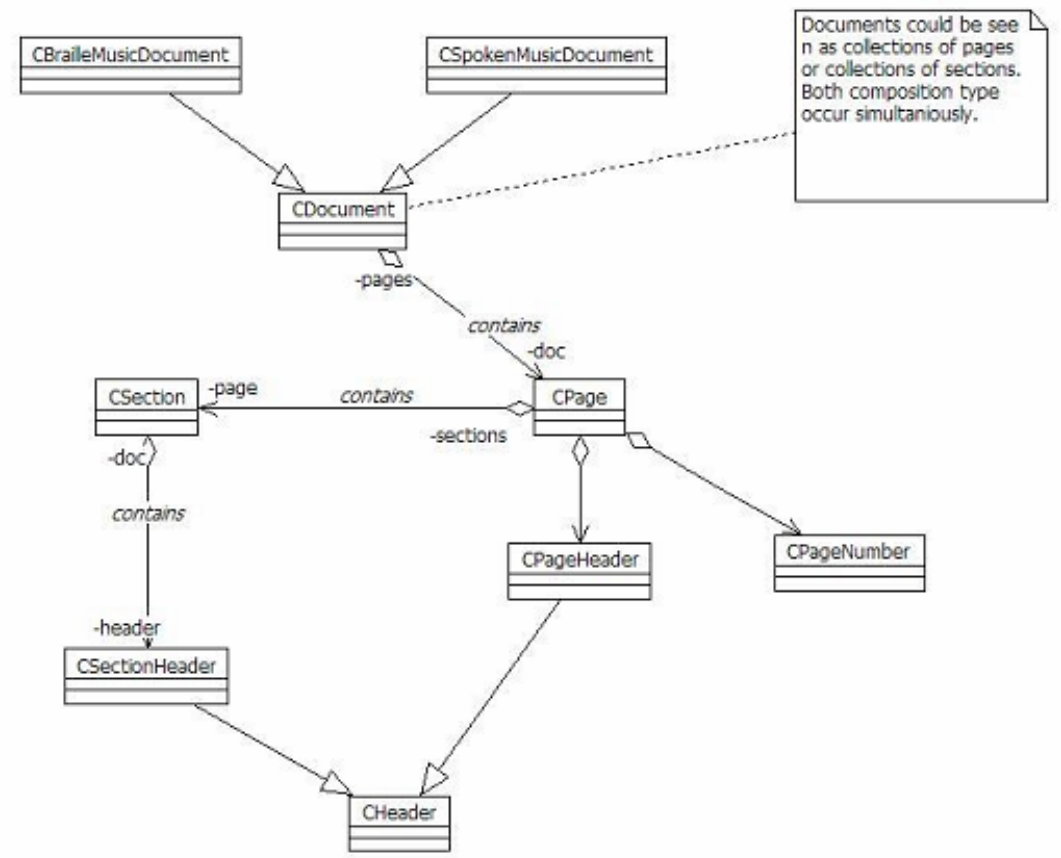


Figure: A class diagram representing a document model

Again, a well-balanced class tree is preferred, since it enables us to reuse general behaviour in document management in a more intuitive way. More specialised formatting tasks can be catered for by means of inheritance.

FileOutput

To be able to model the different requirements for *outputting* the transcribed music, the relevant entities involved in output modelling should be represented as classes. In this stage all additionally required procedures that are specific for the possible output formats can be designed.

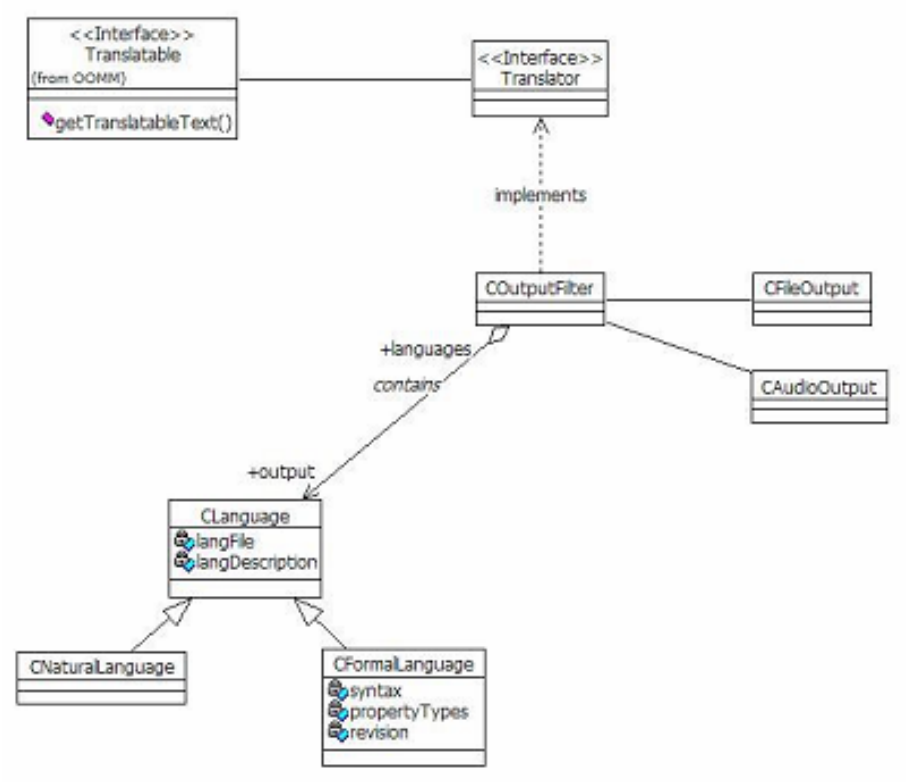


Figure: FileOutput Classes

Cooperation between components/packages

Note that not all components for interpreting Braille Music and Talking Music have to be present in one plugin. Both output formats merely share the same codebase, making it possible to *combine* both modules in one decoder.

All resources from the I-MAESTRO client tool can be used, such as: font dialogs, ornament dialogs, printing dialogs, MIDI playback and file manipulation. Specific procedures for calling such resources must be followed. The protocols and procedures for interfacing with these elements are made clear else where in this specification document (DE3.1.1)

9 Specification of Gesture and Posture Acquisition and Processing Tool

9.1 Gesture and Posture Overview

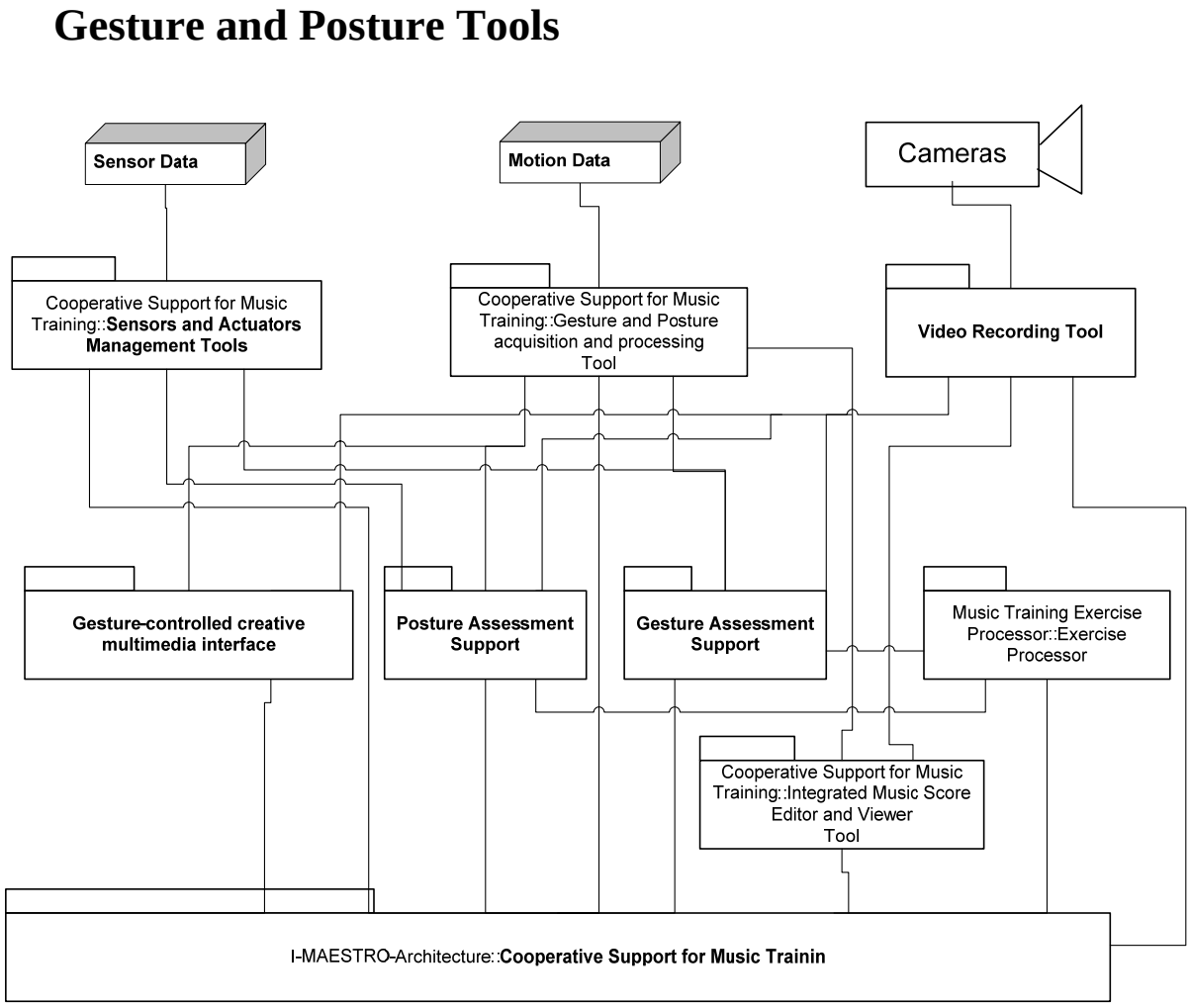


Figure: Gesture and Posture Tools

Gesture and Posture Tools use a set of modules to assess the Students performance. They include Cameras with Video Recording Tool, Motion Capture Systems with Gesture and Posture Acquisition and Processing Tool, Sensors with Sensor and Actuators Management Tools.

The Sensor and Actuators Management Tools, Gesture and Posture Acquisition and Processing Tool and Video Recording Tool are linked to the Posture Assessment Support and to the Gesture Assessment Support to send Students' evaluations for assessment through the Cooperative Support for Music Training.

The Exercise Processor is connected to the Gesture Assessment Support and the Posture Assessment Support for provision and execution of exercises, the performance for which is to be assessed.

The Integrated Music Score Editor and Viewer Tool is connected to the Gesture and Posture Acquisition and Processing Tool and the Video Recording Tool in order to perform Annotations using motion or video data.

Gesture and Posture Acquisition and Processing Tool and Video Recording Tool are also connected to the Gesture-controlled creative multimedia interface to allow mapping of motion (2D or 3D) into sound (sonification).

Each tool is connected directly to Cooperative support for Music Training to send and receive commands and information through the P2P Network.

9.2 Gesture and Posture Acquisition

Gesture and Posture Acquisition and Processing Tool

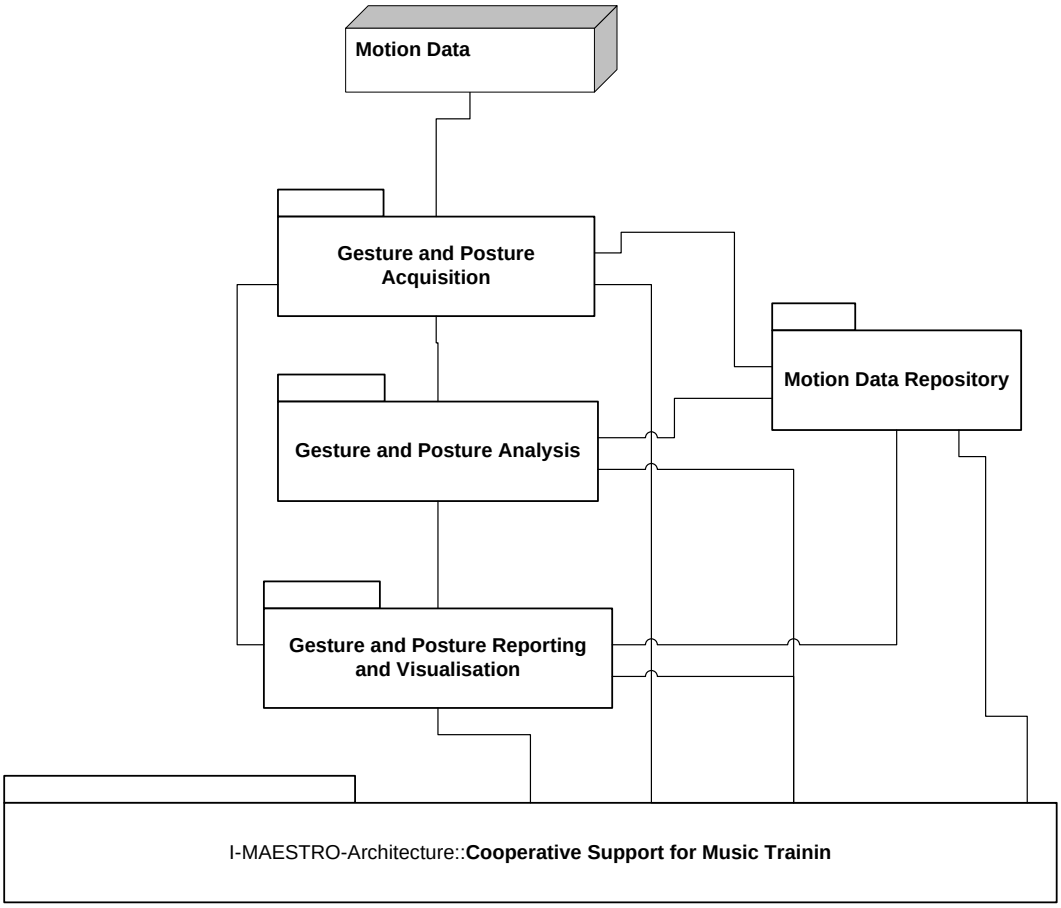


Figure: Gesture and Posture Acquisition and Processing Tool

Gesture and Posture Acquisition and Processing Tool consists of Gesture and Posture Acquisition module that takes as input 3D Motion Data from the Motion Capture System. The Motion Data is used for Gesture and Posture analysis of the Actor. The Motion Data can be saved, loaded, deleted etc. using Motion Data Repository. The Gesture and Posture Reporting and Visualisation module outputs the Motion Data on the screen or sends it to a remote user through the Cooperative Support for Music Training. The Motion Data may also be passed directly from Acquisition module to Reporting and Visualisation module without analysis by the Gesture and Posture Analysis module. This functionality is useful for “enhanced mirror”

capability. Each tool is connected directly to Cooperative support for Music Training to send and receive commands and information through the P2P Network.

9.3 Gesture-controlled Creative Multimedia Interface

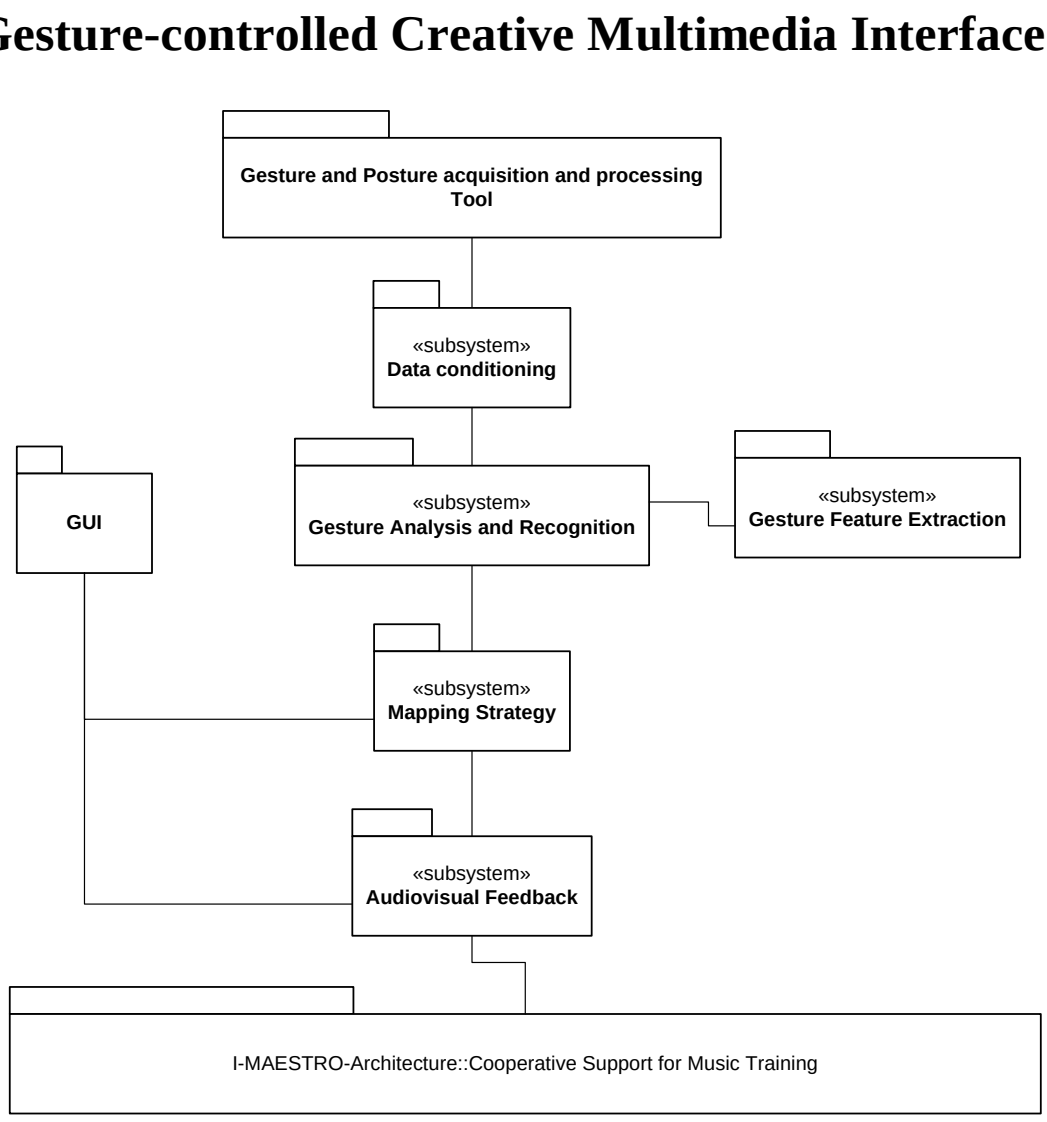


Figure: Gesture-controlled Creative Multimedia Interface

Gesture-controlled Creative Multimedia Interface performs *transformation* of data into audio and visual feedbacks to allow different forms of interpretations, with a user configurable mapping strategies interface.

As shown in Figure, the Creative Multimedia Interface receives input data of performance from the Gesture and Posture acquisition and processing tool. This data is conditioned by the Data conditioning module and is passed on to the Gesture Analysis and Recognition module. This module extracts Gesture features from the Motion data (with the help of Gesture Feature extraction module), which are then mapped on to multimedia output using a mapping strategy (as defined by the Mapping Strategy module).

Audiovisual Feedback module outputs the results of mapping in accordance to the input data. It is also connected to the Cooperative Support for Music Training to send and receive commands and information through the P2P network. Both the Mapping Strategy module and Audiovisual Feedback modules are connected to a local GUI module, using which the user can change parameters for mapping strategy as well as for interactive multimedia rendering.

9.4 Gesture and Posture Assessment Support

Gesture and Posture Assessment Support provides the Teacher with facilities for assessment of a Student's performance in terms of Gesture and Posture, using 3D Motion Data of the performance.

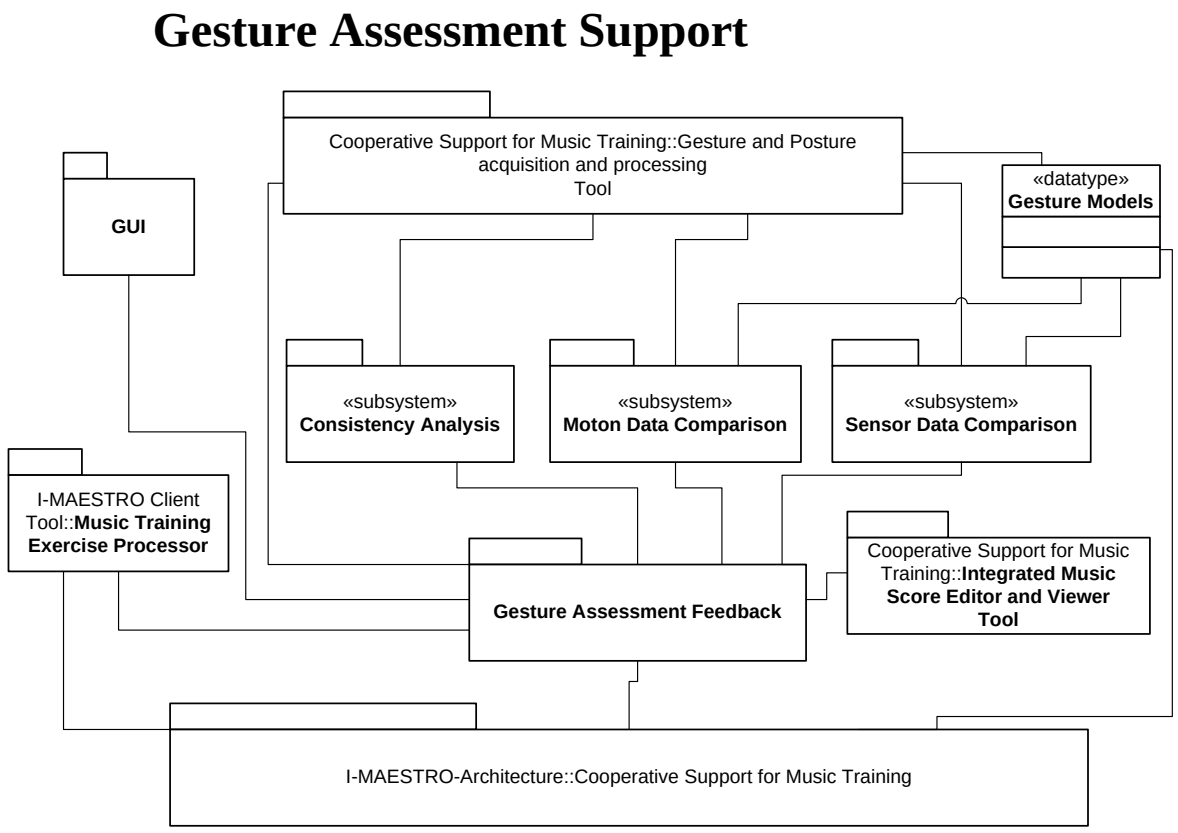


Figure: Gesture Assessment Support

As can be seen in Figure "Gesture Assessment Support", 3D Motion Data of a performance is acquired using the Gesture and Posture acquisition and processing Tool. This Motion Data can either be of a Student's performance or of a Gesture Model (by teacher). The Motion Data for a Student's performance is used by the

- Consistency Analysis, for monitoring the consistency of Bow gestures of Student using graphs and charts. The analysis for these graphs/charts is done in this module and the results are fed to the Gesture Assessment feedback for visualisation and output.
- Motion Data Comparison, for comparing a Student's performance with either (a) a Gesture model or (b) with another Student's performance.
- Sensor Data Comparison, for comparing a Student's performance with either (a) a Gesture model or (b) with another Student's performance.

All aforementioned modules are connected to the Gesture Assessment Feedback module which outputs the assessment results and visualisation through a local GUI module. Additionally profile history of the usage may be sent via the Cooperative Support for Music Training (P2P) to the School Server or the Teacher. The Gesture Assessment Feedback module is also connected to the Integrated Music Score Editor and Viewer tool for Annotation functionalities.

The exercises, for which the performance is to be assessed are provided and executed by the Music Training Exercise Processor.

Each tool is connected directly to Cooperative support for Music Training to send and receive commands and information through the P2P Network.

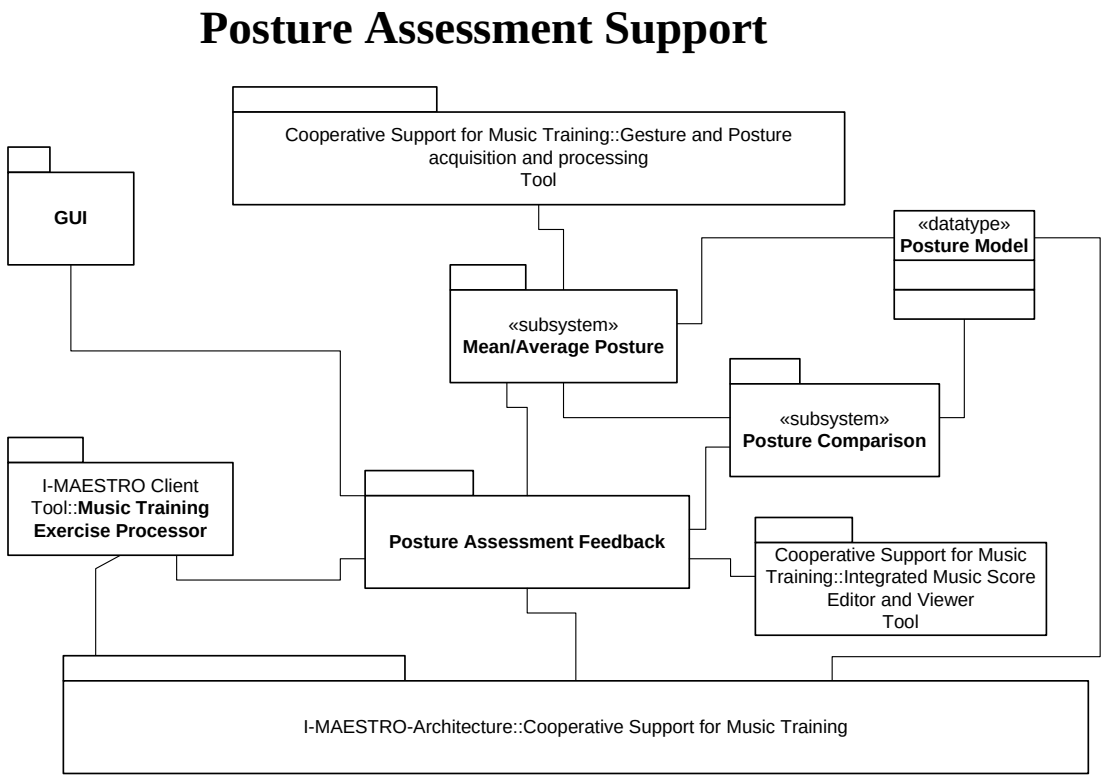


Figure: Posture Assessment Support

For Posture Assessment (see Figure "Posture Assessment Support") 3D Motion Data of a performance is acquired using the Gesture and Posture acquisition and processing Tool. This Motion Data can either be of a Student's performance or of a Posture Model (by teacher). The Motion Data for a Student's performance is used by the Mean/Average Posture module, which provides a static posture by taking mean/average of the varying Posture for comparison with a Posture Model (or a mean/average posture from another Student's performance Motion Data). This comparison is carried out by the Posture Comparison module. Posture Assessment Feedback outputs the assessment results and visualisation through a local GUI module. Additionally profile history of the usage may be sent via the Cooperative Support for Music Training (P2P) to the School Server or the Teacher. The Posture Assessment Feedback module is also connected to the Integrated Music Score Editor and Viewer tool for annotation functionalities.

The exercises, for which the performance is to be assessed are provided and executed by the Music Training Exercise Processor.

Each tool is connected directly to Cooperative support for Music Training to send and receive commands and information through the P2P Network.

9.5 Gesture & Posture Acquisition & Processing Tool Connection with Cooperative Support

These are the class diagrams for the Gesture and Posture tools.

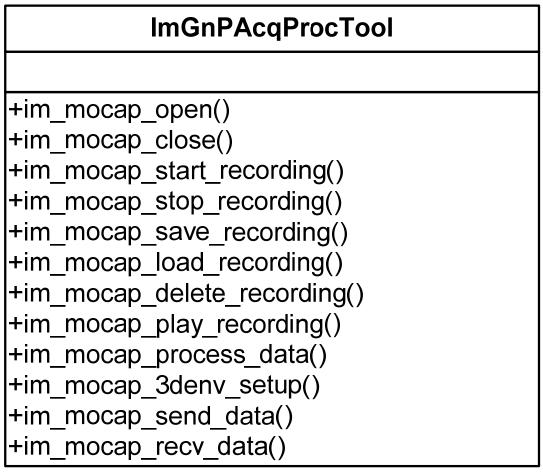


Figure: Class Diagram for Gesture and Posture Acquisition and Processing Tool

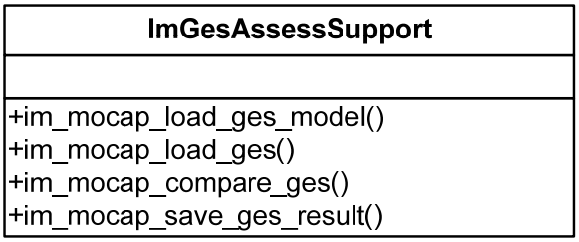


Figure: Class Diagram for Gesture Assessment Support

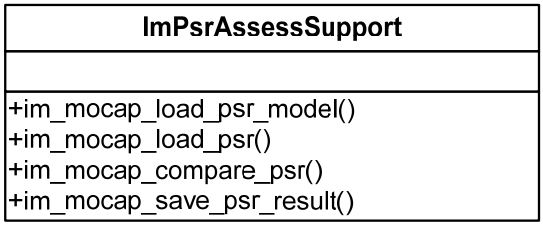


Figure: Class Diagram for Posture Assessment Support

For each method of the classes illustrated above, a table is specified that describes the functionality, input & output parameters and request & response sample messages for the method.

im_mocap_open()	
Method	im_mocap_open()
Description	Open a connection with the Motion Capture System to capture Motion Data of a performance
Input parameters	
Output parameters	I-MAESTRO Object containing the Motion Data
Request Sample Message	<message> <request> Request to open a connection with the Mocap System </request> </message>
Response Sample Message	<message> <response> Motion Data in c3d format </response> </message>

im_mocap_close()	
Method	im_mocap_close()
Description	Close a previously opened connection with the Motion Capture System
Input parameters	
Output parameters	Acknowledgement
Request Sample Message	<message> <request> Request to close connection with the Mocap System </request> </message>
Response Sample Message	<message> <response> Connection closed successfully </response> </message>

im_mocap_start_recording()	
Method	im_mocap_start_recording()
Description	Start recording Motion Data of a performance
Input parameters	
Output parameters	I-MAESTRO Object with Motion Data recording of a performance
Request Sample Message	
Response Sample Message	

im_mocap_stop_recording()	
Method	im_mocap_stop_recording()
Description	Stop recording Motion Data of a performance

Input parameters	
Output parameters	
Request Sample Message	
Response Sample Message	

im_mocap_save_recording()	
Method	im_mocap_save_recording()
Description	Save Motion Data of a performance to a file
Input parameters	I-MAESTRO Object with Motion Data of performance
Output parameters	File Object
Request Sample Message	
Response Sample Message	

im_mocap_load_recording()	
Method	im_mocap_load_recording()
Description	Load Motion Data of a performance from a file
Input parameters	File Object
Output parameters	I-MAESTRO Object with Motion Data of performance
Request Sample Message	
Response Sample Message	

im_mocap_delete_recording()	
Method	im_mocap_delete_recording()
Description	Delete Motion Data of a performance
Input parameters	I-MAESTRO Object with Motion Data of performance
Output parameters	
Request Sample	

Message	
Response Sample Message	

im_mocap_play_recording()	
Method	im_mocap_play_recording()
Description	Play recorded Motion Data of a performance
Input parameters	I-MAESTRO Object with Motion Data of performance
Output parameters	3D Visualisation of Motion Data
Request Sample Message	
Response Sample Message	

im_mocap_process_data()	
Method	im_mocap_process_data()
Description	Process the 3D Motion Data to create a wire-frame representation of Actor in 3D space
Input parameters	I-MAESTRO Object containing the 3D Motion Data
Output parameters	I-MAESTRO Object containing the processed 3D Motion Data
Request Sample Message	
Response Sample Message	

im_mocap_3denv_setup()	
Method	im_mocap_3denv_setup()
Description	Set up 3D environment i.e. lighting etc.
Input parameters	I-MAESTRO Object containing the 3D Motion Data
Output parameters	I-MAESTRO Object containing the processed 3D Motion Data
Request Sample Message	
Response Sample Message	

im_mocap_load_ges_model()	
Method	im_mocap_load_ges_model()
Description	Load Bow Gesture Model
Input parameters	

Output parameters	I-MAESTRO Object containing Gesture Model (loaded from File or Cooperative Support for Music Training)
Request Sample Message	
Response Sample Message	

im_mocap_load_ges()	
Method	im_mocap_load_ges()
Description	Load a previously saved Gesture of Student's performance
Input parameters	
Output parameters	I-MAESTRO Object containing Gesture
Request Sample Message	
Response Sample Message	

im_mocap_compare_ges()	
Method	im_mocap_compare_ges()
Description	Compare Gestures of two performances (one of which could be a Bow Gesture Model)
Input parameters	▪ I-MAESTRO Object ▪ I-MAESTRO Object
Output parameters	I-MAESTRO Object containing Gesture comparison results (visualisation)
Request Sample Message	
Response Sample Message	

im_mocap_save_ges_result()	
Method	im_mocap_save_ges_result()
Description	Save I-MAESTRO Object containing Gesture comparison results (visualisation) in a File
Input parameters	I-MAESTRO Object containing Gesture comparison results (visualisation)
Output parameters	File Object
Request Sample Message	
Response Sample Message	

im_mocap_load_psr_model()	
Method	im_mocap_load_psr_model()

Description	Load Posture Model
Input parameters	
Output parameters	I-MAESTRO Object containing Posture Model (loaded from File or Cooperative Support for Music Training)
Request Sample Message	
Response Sample Message	

im_mocap_load_psr()	
Method	im_mocap_load_psr()
Description	Load a previously saved Posture of Student's performance
Input parameters	
Output parameters	I-MAESTRO Object containing Posture
Request Sample Message	
Response Sample Message	

im_mocap_compare_psr()	
Method	im_mocap_compare_psr()
Description	Compare Postures of two performances (one of which could be a Posture Model)
Input parameters	▪ I-MAESTRO Object ▪ I-MAESTRO Object
Output parameters	I-MAESTRO Object containing Posture comparison results (visualisation)
Request Sample Message	
Response Sample Message	

im_mocap_save_psr_result()	
Method	im_mocap_save_psr_result()
Description	Save I-MAESTRO Object containing Posture comparison results (visualisation) in a File
Input parameters	I-MAESTRO Object containing Posture comparison results (visualisation)
Output parameters	File Object
Request Sample Message	
Response Sample Message	

im_mocap_send_data()	
Method	im_mocap_send_data()
Description	
Input parameters	
Output parameters	
Request Sample Message	
Response Sample Message	

im_mocap_recv_data()	
Method	im_mocap_recv_data()
Description	
Input parameters	
Output parameters	
Request Sample Message	
Response Sample Message	

9.6 Gesture and Posture Acquisition and Processing Tool User Interface

The User Interface for Gesture and Posture Acquisition and Processing Tool will provide connection with the Motion capture system to record, 3D Motion data, process 3D Motion data, save, load, play and delete recordings etc.

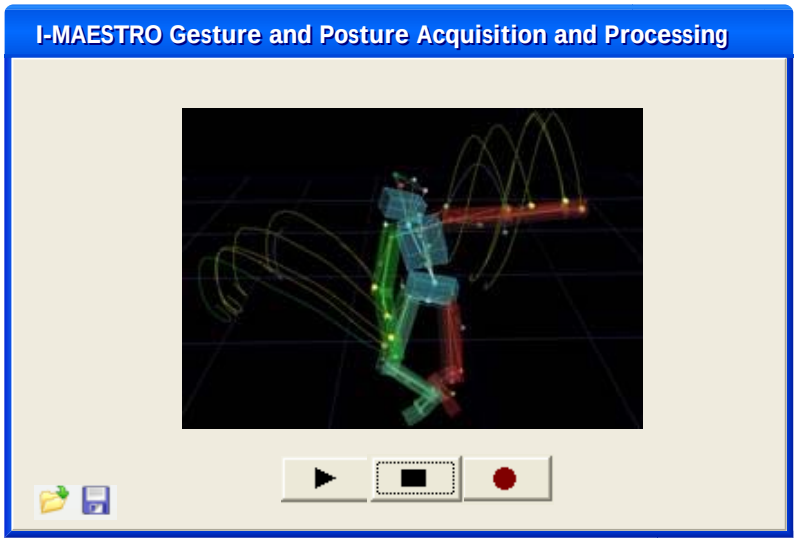


Figure: Gesture and Posture Acquisition and Processing Tool User Interface

9.7 Gesture and Posture Acquisition and Processing Tool Configuration

The Gesture and Posture Acquisition and Processing Tool requires a number of configuration parameters such as

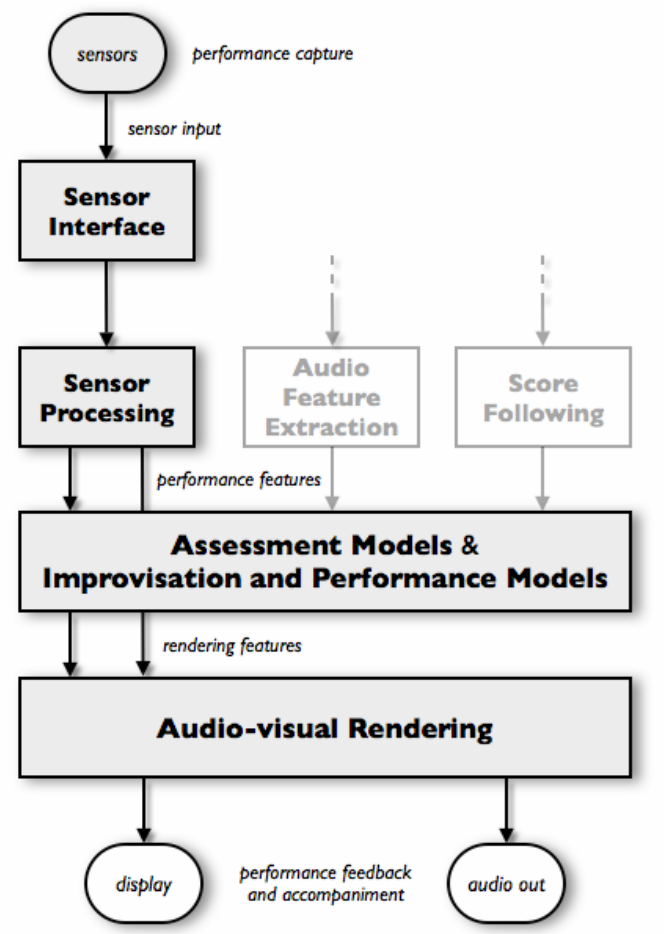
- Sampling frequency
- Number of markers used in the capture session
- Length (time) of the capture session
- Etc.

These configuration data are important to allow the data to be render, reconstructed and analysed correctly.

10 Specification of Sensors Acquisition and Processing Tool

10.1 Overview of Sensor Processing data flow and components

The Sensor technology and Sensor Processing Modules developed in WP5.2 are summarized in the following data flow diagram (for an overview of the Practice Training environment see the General Overview document of the I-MAESTRO Specification).



Specific components supporting Assessment and Performance Models based on sensor input will use simultaneously the sensor *and* audio performance parameters.

10.2 Specification of Sensor Processing components

All modules are implemented as autonomous Max/MSP modules. Max/MSP modules in general can have multiple *control* inputs and outputs (passing of messages and values using an event model). Messages and values sent to an inlet of a Max/MSP module correspond to methods implemented by the module.

10.2.1 Sensor Technology

For the violin bow, the sensor technology will be primarily based on accelerometers. Prototype of the wireless bow measurement system will be composed of a small electronic board with a microcontroller,

accelerometers (example ADXL202 accelerometers from Analog Devices). Digital radio transmitter can be also added, but is optional. The use of Bluetooth Technology is possible and will be evaluated. Other sensor such as FSR (Force Sensitive Sensor), optical sensor, bending sensors can be also used, especially of body movement such as arm are captured.

10.2.2 Sensor Interface and communication

The sensor interface depends whether wireless communication is used. Concerning *hardwar*e, the sensor interface can be either directly integrated in the board containing the sensors, or it can be a separate box, containing an A/D conversion system, or in the case of wireless transmission, containing the radio receiver element. The communication protocol will be based on Ethernet communication protocol on a local network.

The software will integrate a module allowing for the handling of the communication protocol, primarily based on the OpenSoundControl protocol[4]. OpenSoundControl ("OSC") is a protocol for communication among computers, sound synthesizers, and other multimedia devices that is optimized for modern networking technology.

Initialization module	
Description	Initialize all parameters, recall presets, start/stop sensor data stream.
Instantiation Arguments	arg 1: preset number (integer corresponding to a particular type of experiment, sensor set, version, etc.)
Attributes	none
Methods	Messages to set parameters for the setup, initialization, and start/stop. Preset number: recall a given preset of parameterization Read/Write: read and write presets Reset OSC: reset communication OSCparam: set OSC parameters (Host IP, Interface IP, port) InterfaceParam : set interface parameters DSP parameters
Inlets	inlet 1: messages
Outlets	outlet 1: messages

OSC communication module	
Description	Module managing OSC communication
Instantiation Arguments	arg 1: IP address arg 2: Port number arg 3: Device list
Attributes	none
Methods	Messages to set parameters of the OSC communication module IPhost: IP of the host computer IPinterface IP of the interface Port: port number Devicelist: parameters of a given interface
Inlets	inlet 1: Gesture features or characteristics
Outlets	outlet 1: message outlet 2: sensor data stream

10.2.3 Sensor Processing

The sensor processing includes various prepossessing such as filtering, scaling, downsampling of the sensor data stream. A second stage of processing concerns segmentation and feature extractions. This stage will also

[4] see http://www.cnmat.berkeley.edu/OpenSoundControl

require the use of the audio stream and/or communication with Audio Feature Extraction module, in order to disambiguate sensor segmentation and sensor feature extraction algorithms.

Sensor analysis module	
Description	Module estimating an arbitrary sensor descriptor from an sensor data stream
Instantiation Arguments	static parameters of the extraction algorithm (e.g. output control rate)
Attributes	not defined
Methods	messages to set parameters of the analysis algorithm
Inlets	inlet 1: Sensor data stream (gesture data stream)
Outlets	outlet 1: stream of floating-point values (descriptors)

Acceleration peaks extractor module	
Description	Extract maximum and minimum in data stream form accelerometers and perform basic segmentation
Instantiation Arguments	arg 1: minimum time between peaks arg 2: maximum time between peaks arg 3: threshold for peak analysis
Attributes	not defined
Methods	Messages to set parameters of the extraction algorithm MinTime: minimum time between peaks MaxTime: maximum time between peaks Threshold: threshold for peak analysis
Inlets	inlet 1: accelerometer data stream
Outlets	outlet 1: vector with maximum and minimum

10.2.4 Support of Assessment, Improvisation and Performance Models

WP 5.4 integrates various developments related to Sensor Processing to support Assessment Models as well as Improvisation and Performance models defined in the following.

Gesture recognition and follower module	
Description	Perform gesture recognition and/or gesture following.
Instantiation Arguments	arg 1: algorithm used
Attributes	not defined
Methods	Messages to set parameters of the extraction algorithm algo : algorithm used learn: set in training mode process: process training examples play: analyse continuously data stream import: import training data from a file export: export processed training data into a file clear: clear training examples
Inlets	inlet 1: sensors features stream
Outlets	outlet 1: likelihood vector (for recognition) outlet 2: time index vector (for following)

Recoding and retrieval of sensor and performance data module	
Description	Read/Write sensor data and sensor feature data into files
Instantiation Arguments	none
Attributes	not defined

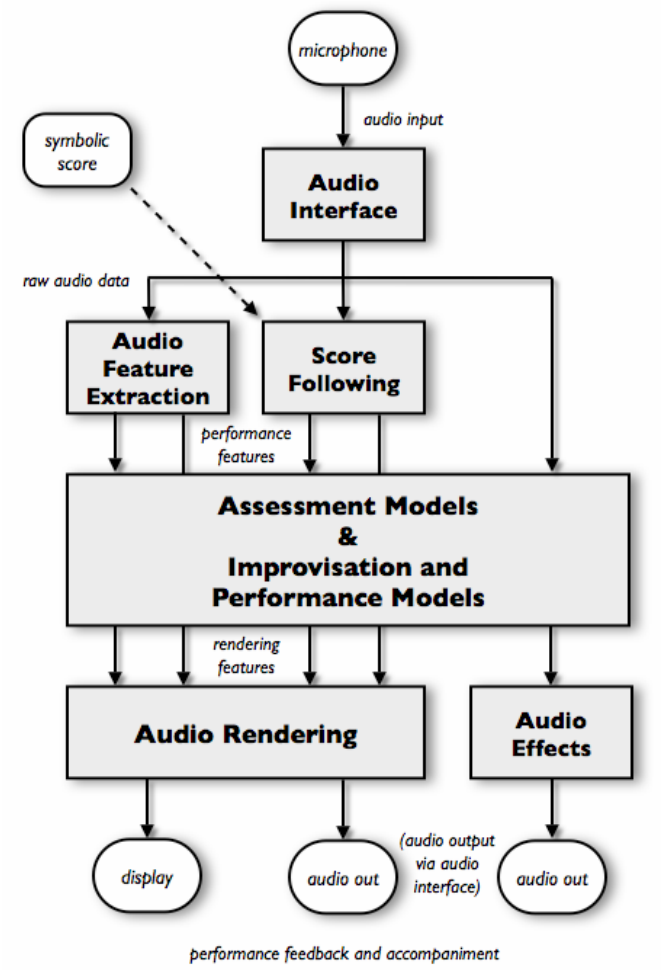
Methods	Messages to read/write files and set parameters. read ‘filename’ ‘parameters’: write ‘filename’ ‘parameters’
Inlets	inlet 1: message, sensor data
Outlets	outlet 1: message outlet 2: sensor data

Sensor to sound mapping patch	
Description	Performs the mapping between gesture and sound
Instantiation Arguments	none
Attributes	not defined
Methods	Messages to set parameters start stop
Inlets	inlet 1: sensor data stream inlet 2 : sensor features data stream
Outlets	outlet 1: audio signal outlet 2: audio signal

11 Specification of Audio Processing Tools and Score Following

11.1 Overview of Audio Processing data flow and components

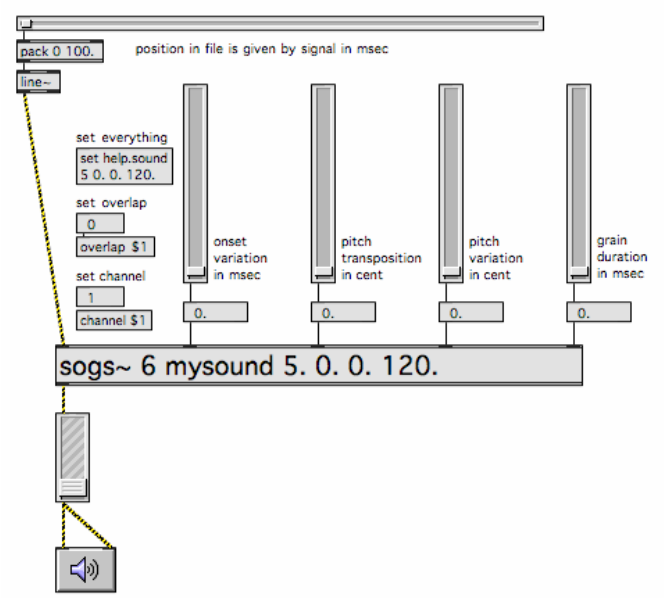
The Audio Processing Modules assembled and partly developed in WP5.4 are summarized in the following data flow diagram (for an overview of the Practice Training environment see the General Overview document of the I-MAESTRO Specification).



11.2 Specification of Audio Processing and Score-following components

All modules are implemented as autonomous Max/MSP modules. Max/MSP modules in general can have multiple *audio* inputs and outputs (block-wise processed audio data flow) as well as multiple *control* inputs and outputs (passing of messages and values using an event model). Connections between modules are clearly distinguished in these two categories. Messages and values sent to an inlet of a Max/MSP module correspond to methods implemented by the module (see Max/MPS documentation for further information on Max/MSP).
As a convention audio processing modules in Max/MSP have a name ending with "~" (tilde) as the modules sogs~ in the following screen shot of a simple Max/MSP *patch*.[5]

[5] Max/MSP documents are referred to as *patches* and an open window containing the modules and connections in the Max/MSP editor is called a *patcher*.



A part of the components such as the Score-following module are based on the FTM[6] extension for Max/MSP developed at IRCAM. FTM introduces the possibility to send references to complex data structures (matrices, sequences, dictionaries, etc.) between Max/MSP objects and to further modulize Max/MSP applications.

11.2.1 Implementation of audio processing modules in Max/MSP

Various ways to implement Max/MSP audio modules within WP5.4 are envisaged:

- Max/MSP *abstractions* using Max/MSP standard modules and the *Gabor*[7] module library based on IRCAM's FTM extension
- Max/MSP *externals* written in C using the Max/MSP API published within the Max/MSP SDK
- Modules compliant to the VST plug-in standard[8] supported by Max/MSP

In either case a Max/MSP module is completely defined by the following elements:

- ***Module Name***
- ***Instantiation Arguments*** (a list of numbers and symbols to further define and initialize the module following the module name in a module box)
- ***Attributes*** (named instantiation arguments given with a name prefixed by "@" followed by a single value given after the positional instantiation arguments)
- **Methods** (functions bound to messages accepted by the module)
- ***Inlets***
- ***Outlets***

[6] Schnell Norbert, Borghesi Riccardo, Schwarz Diemo, Bevilacqua Frédéric, Müller Remy, *FTM — Complex data structures for Max*. International Computer Music Conference (ICMC). Barcelona : Septembre 2005

Also see http://www.ircam.fr/ftm/ for further documentation concerning FTM.

[7] Schnell Norbert, Schwarz Diemo, *Gabor, Multi-Representation Real-Time Analysis/Synthesis*. COST-G6 Conference on Digital Audio Effects (DAFx). Madrid : Septembre 2005, p. 122-126

[8] Virtual Studio Technology, see www.steinberg.net

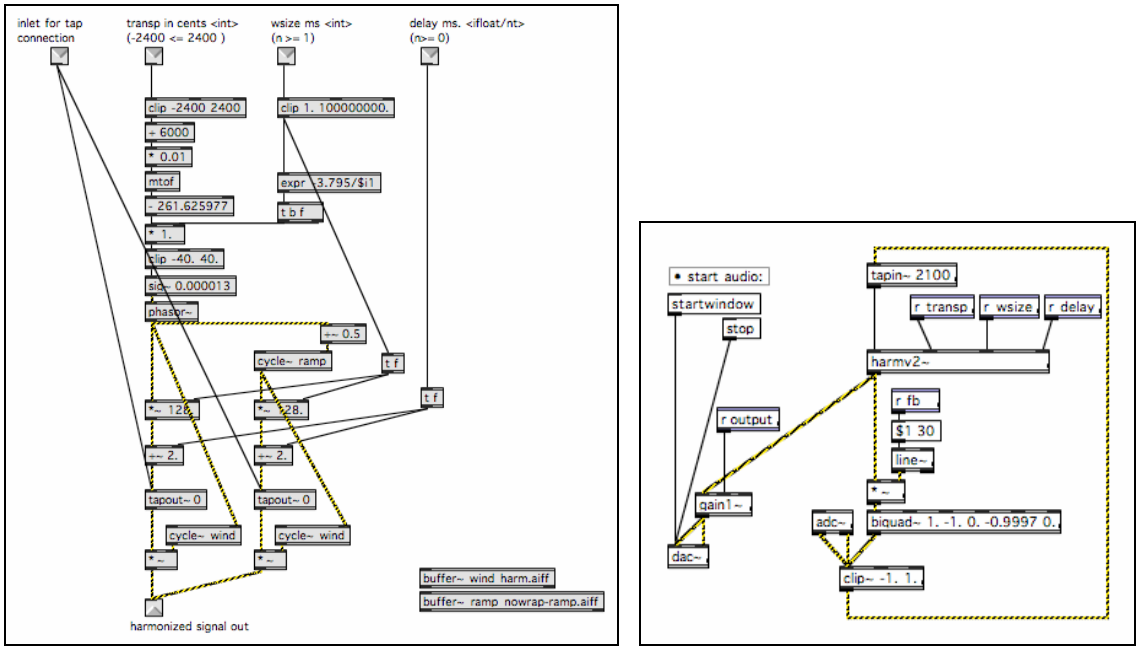
The following table will be used for the definition and documentation of Max/MSP modules within the specification related to WP5.4:

Module Name	
Description	Description of the module's functionalities
Instantiation Arguments	Description of the module's positional instantiation arguments in a list: arg 1 (*type of first arg1*): description of the first instantiation argument arg 2 (*type of first arg1*): description of the second instantiation argument …
Attributes	Description of the module's named instantiation arguments in a list: *attrA* (*type of first attrA*): Description of the attribute *attrA* *attrB* (*type of first attrB*): Description of the attribute *attrB* …
Methods	Description of the messages accepted by the module in a list: *messageA*: Description of the message *messageA* arg 1 (*type of first arg1*)*:* Description of the first argument of *messageA* arg 2 (*type of second arg2*)*:* Description of the second argument of *messageA* … *messageB*: Description of the message *messageB* arg 1 (*type of first arg1*): Description of the first argument of *messageB* arg 2 (*type of second arg2)*: Description of the second argument of *messageB* … … Messages with a single argument can be described more compactly as *messageC* (number): Description of the message *messageB* Messages without arguments can be described as *messageD*: Description of the message *messageB*
Inlets	Description of the module's inlets in a list: inlet 1 (*type of inlet 1*): Description of first inlet inlet 2 (*type of inlet 2*): Description of second inlet …
Outlets	Description of the module's outlets in a list: outlet 1 (*type of outlet 1*): Description of the first outlet outlet 2 (*type of outlet 2*): Description of the second outlet …

Max/MSP abstractions

Max/MSP *abstractions* are defined by an ordinary Max/MSP *patch* (a Max/MSP document) and can be used in other Max/MSP patches in the same way as any other Max/MSP module. The inlet and outlet objects used in the definition of the abstraction are showing up as the inlets and outlets of the abstraction in the surrounding patch.

The following images show the implementation of the harmv2~ module of the *Jimmies* library (see section *Audio Effect modules* below) based on standard Max/MSP modules and an extract from simple audio application patch using the harmv2~ module:



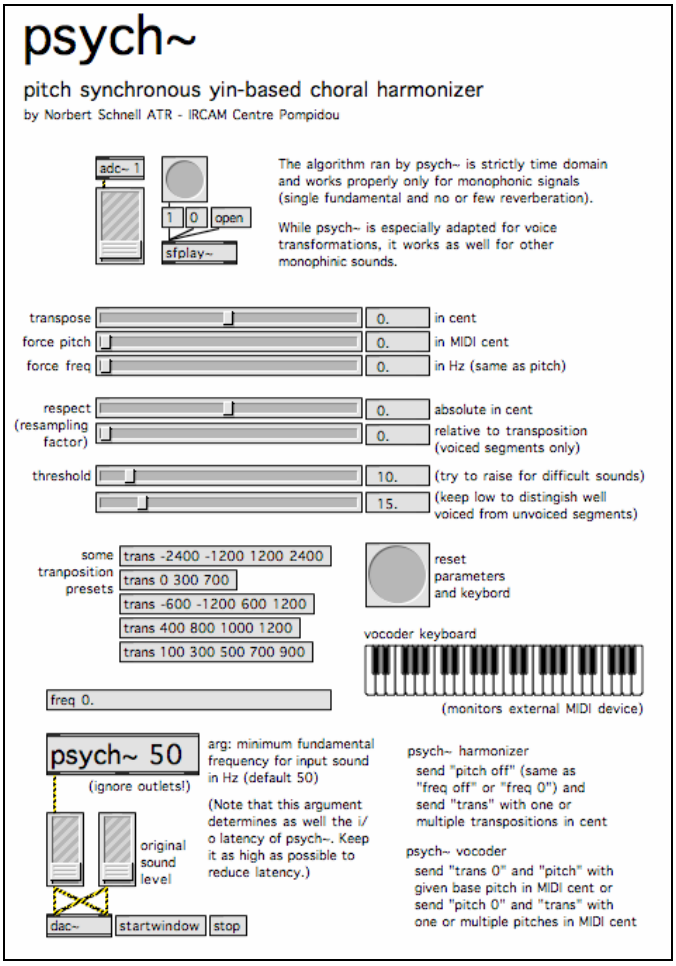
The used abstraction's module name (here harmv2~) is the filename of the Max/MSP patch document, by which it is defined (here harmv2~.pat).

Max/MSP externals

Max/MSP provides the possibility to implement modules in C. The Max/MSP community refers to C these modules as Max/MSP *externals*. The API for the implementation is defined in the Max/MSP SDK provided for *Windows XP* as well as *Mac OS X*.
An external has to implement the following elements in form of C functions compliant to the SDK:

- The modules (class) definition function with the following declarations
  - basic module declaration including the *constructor* and *deconstructor* methods (C functions) as well as a declaration of the module as audio (DSP) module
  - method declarations for messages (binding a message name to a C function)
  - method declarations for inlets (binding a message name to a C function)
- Implementation of the *constructor* method (accessing the instantiation arguments), including module initialisation and the creation of inlets and outlets
- Implementation of the *deconstructor* method (accessing the instantiation arguments)
- Implementation of the methods bound to messages and inlets
- Implementation of an audio processing initialisation method (called when Max/MSP audio processing is started)
- Implementation of the audio processing method called for each block audio samples

The module psych~ developed at IRCAM is an example of a C written Max/MSP *external*. The following images shows a screen shot of the *help patch* of the psych~ module explaining its functionality and providing a simple application for trying the module. This help patch together with the modules definition is a full example of the specification and basic documentation of a Max/MSP module within WP 5.4.



VST plug-ins

VST is a wide spread audio plug-in standard by *Steinberg AG*. VST plug-ins can be embedded to Max/MSP patches using the Max/MSP module vst~ implementing a VST host. The module is instantiated with the name of the plug-in as an argument and provides audio inlets and outlets as well as messages mapped to the control parameters defined by the VST plug-in.
For the development of VST plug-ins, an SDK including detailed documentation is available from Steinberg[9].

11.2.2 Audio Interface

The (software) audio interface is provided within the Max/MSP environment. Max/MSP modules are provided representing audio inputs (adc~) and outputs (dac~). The module adstatus permits to set and get the values of various parameters of the audio interface such as the sample rate and the audio input/output buffer size (see Max/MSP documentation for further description).

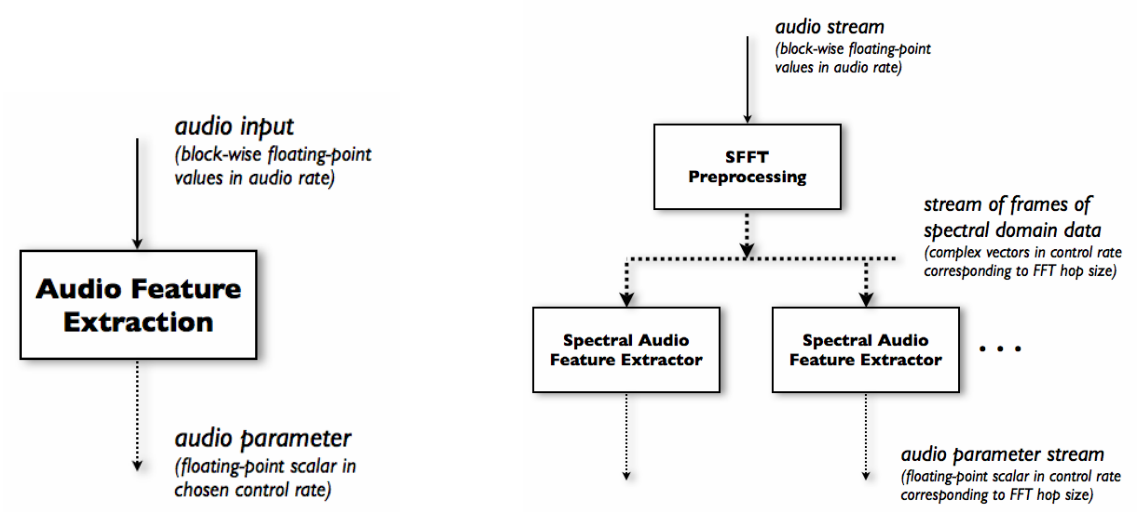
[9] VST-SDK, Steinberg's Virtual Studio Technology Software Development Kit, see www.steinberg.net.

11.2.3 Audio Feature Extraction Modules

In the following, the Audio Feature Extraction modules are defined by the audio and control input/output data flow as well by a detailed specification of the modules' parameters and functionalities. Generic specifications and examples are given.

Architecture and data flow

The Audio Feature Extraction Modules in general are Max/MSP modules accepting an audio stream at the input and providing a stream of scalar values at the output. In Max/MSP these modules are implemented with an audio inlet and a control outlet outputting a floating-point value. Since many feature extraction modules use a *Short Time Fourier Transform* (SFFT) processing scheme, it is desirable to share the FFT computation by multiple of such modules especially if they are used in parallel as shown in the figure on right. The two types of modules can be distinguished as *Time Domain Audio Feature Extraction Modules* and *Frequency Domain Audio Feature Extraction Modules.*



Module Definitions

Generic specifications are given for Time Domain Audio Feature Extraction Modules and Frequency Domain Audio Feature Extraction Modules as well as for the SFFT pre-processing stage.
A concrete example for the specification of a Time Domain Audio Feature Extraction Module is given by the module yin~ of the IRCAM-ATR modules (see table below).
Frequency Domain Audio Feature Extraction Modules following the given scheme are implemented using the FTM extension for Max/MSP allowing sending vectors and matrices between Max/MSP objects (see http://www.ircam.fr/ftm/). Complex (frequency domain) vectors are implemented as floating-point matrices, FTM *fmat* objects with two columns containing the real and imaginary values.

Time Domain Audio Feature Extraction Module	
Description	Module estimating an arbitrary audio descriptor from an audio stream
Instantiation Arguments	static parameters of the extraction algorithm (e.g. output control rate)
Attributes	dynamic parameters of the extraction algorithm
Methods	messages to set parameters of the extraction algorithm
Inlets	inlet 1 (audio): audio stream
Outlets	outlet 1 (number): stream of floating-point values

yin~	
Description	Fundamental frequency extractor using the YIN algorithm. The module provides also a periodicity factor and an estimation of the instantaneous signal energy.
Instantiation Arguments	arg 1: input down-sampling factor (0 = off, 1 | 2 | 3 = downsampling by 2 | 4 | 8) arg 2: lowest estimated frequency arg 3: output period in msec
Attributes	threshold (number 0…1): initialize estimation threshold for yin algorithm
Methods	threshold (number 0…1): set the estimation threshold for yin algorithm
Inlets	inlet 1 (audio): audio input stream
Outlets	outlet 1 (number): estimates frequency in Hz outlet 2 (number): estimated energy as a linear factor outlet 3 (number): estimated periodicity/harmonicity factor between 0 and 1 (0 = inharmonic/noisy, 1 = harmonic/periodic)

Frequency Domain Audio Feature Extraction Module	
Description	Module estimating an arbitrary audio descriptor from an SFFT input stream
Instantiation Arguments	arg 1: FFT size additional args: static parameters of the extraction algorithm
Attributes	dynamic parameters of the extraction algorithm
Methods	messages to set parameters of the extraction algorithm
Inlets	inlet 1 (FTM fmat): stream of FFT frames, N x 2 matrices with N complex values corresponding to the lower half of a complex DFT spectrum (N = FFT size / 2 + 1)
Outlets	outlet 1 (number): values of the estimated descriptor (unit to be given)

SFFT preprocessing module	
Description	Module calculating a stream of SFFT frames from an audio stream input
Instantiation Arguments	arg 1: FFT size arg 2: SFFT hop size
Attributes	wind: set window type (hann | hamming | Blackman | blackman-harris)
Methods	None
Inlets	inlet 1 (audio): audio input stream
Outlets	outlet 1 (FTM fmat): stream of FFT frames, N x 2 matrices with N complex values corresponding to the lower half of a complex DFT spectrum (N = FFT size / 2 + 1)

11.2.4 Score-following Module

IRCAM has been on Score-following technology since its very beginning. The development of the latest generation of Score-following tools based on Hidden Markov modules has been started at IRCAM in 2000[10].

[10] Nicola Orio and Diemo Schwarz. *Alignment of Monophonic and Polypophonic Music to a Score*. In *Proceedings of the ICMC*, Havana, Cuba, 2001.

Nicola Orio and F. Déchelle. *Score Following Using Spectral Analysis and Hidden Markov Models*. In *Proceedings of the ICMC*, Havana, Cuba, 2001.

Nicola Orio, Serge Lemouton, Diemo Schwarz, and Norbert Schnell. Score Following: *State of the Art and New Developments*. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, Montreal, Canada, May 2003.

Diemo Schwarz, Nicola Orio, and Norbert Schnell. *Robust Polyphonic Midi Score Following with Hidden Markov Models*. In *Proceedings of the ICMC*, Miami, Florida, November 2004.

Diemo Schwarz, Arshia Cont, and Norbert Schnell. *From boulez to ballads: Training ircam's score follower*. In *Proceedings of International Computer Music Conference (ICMC)*. Barcelona, September 2005.

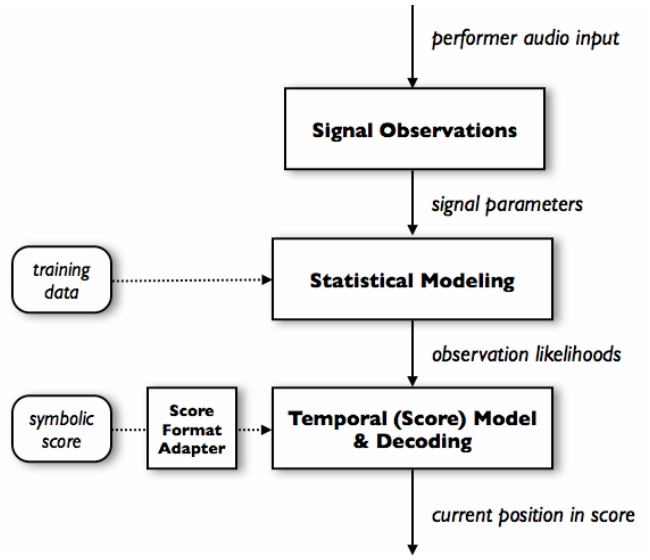
Arshia Cont, Diemo Schwarz, and Norbert Schnell. *Training ircam's score follower*. In *IEEE International Conference on Acoustics and Speech Signal Processing (ICASSP)*. Philadelphia, March 2005.

Arshia Cont, Diemo Schwarz, and Norbert Schnell. *Training ircam's score follower*. In *AAAI Fall Symposium on Style*

In the following the architecture and functionalities of the components being developed in the framework of WP5.4 based on the current Score-following technology are specified.

Architecture and data flow

The Score-following module references a symbolic score representation. It requires an audio stream at the input and provides at run-time references to the recognized score positions at the output.
The score-following module is based on Hidden-Markov Models (HMM). The algorithm can be described with three distinguished processing stages as shown in the following figure:



A first implementation of the module will incorporate all three steps in one integrated component. A future modulized version of the Score-following algorithm will be available as a set of modules and give the possibility to flexibly adapt the Score-following technology to different instruments, playing styles and score models.
The Score Format Adapter is a sub-component permitting to easily adapt the Score-following module to different symbolic score representations. The current Score-following module developed at IRCAM uses an FTM *track* object containing *sccob* objects represented by an independent Max/MSP module to represent the symbolic score to follow. The reference to the *track* object containing the score can be given as instantiation argument (as named FTM reference) or set by a message to the Score-following module with an appropriate FTM *track* object as argument. The adapter translates dynamically (on-the-fly) the given score representation of the currently considered temporal window around the current estimated position into its internal score representation.

Module Definition

The score following module receives the audio input from the performer and continuously estimates and outputs the current position in the performed score. The score can be given at initialisation or set at runtime with a message. Various option messages change the way the score is parsed into the internal score model. The principal follower output is in the form of an abstract score position, depending on the score representation front-end, which can also contain a direct reference to an object in the score and to its properties (e.g. cue number). For some applications, inspection data can be output, which is internal data such as the analysed signal parameters, observation likelihoods, and HMM state. These can serve for testing the setup (is the signal coming in), recording new instrument data, or debugging.

---

and Meaning in Language, Art and Music. Washington DC, 2004.

Score-following module		
Description	Recognition of the current position in the performed score according to audio input from the performer	
Instantiation Arguments	reference to the symbolic score	
Attributes	inspect:	configure additional inspection outputs (on|off), switch inspection on
Methods	set (score ref.):	set score to follow
	start (number):	start follower at given position (default: beginning of score) with given time scaling factor (default 1)
	stop:	stop follower
	maxdiff (number):	quantisation time (ms) for fusing notes into chords and suppressing short rests in the score (default 30 ms)
	rests (0|1):	modeling of rests on/off
	tune (number):	reference tuning frequency for A4 (default 440 Hz)
	setpdf (symbol):	set preset training data / choose preset
	verbose (0-5):	verbosity of debug output (0–5)
	inspect (0|1):	switch inspection output on or off
Inlets	inlet 1:	audio stream (performance) and message input
Outlets	outlet 1:	recognised current score position
	outlet 2:	signal for score-performance mismatch (performer error)
	outlet 3:	signal for end-of-score
	outlet 4:	inspection data (only when inspection option is set)

11.2.5 Audio Effect Modules

The Audio Effect modules require one or multiple audio streams at the input and provide one or multiple audio stream at the output. Max/MSP provides building blocks to create Effect Modules within the standard libraries. The modules of the *Jimmies* library developed at IRCAM, a set of classical Audio Effect modules, are examples of modules created in form of Max/MSP *abstractions* using the standard Max/MSP modules. The library contains the following modules:

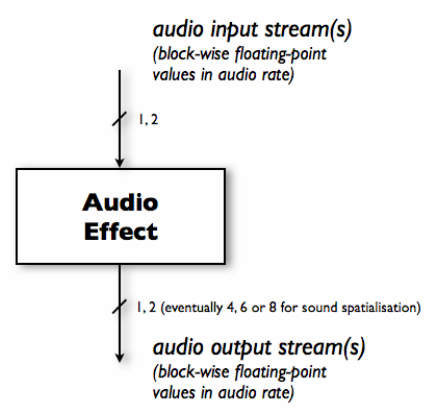
"Jimmies" Classical effects	
stchorus1~	stereo chorus
delay2~	delay line
flange1~	flanger
fshift1~	frequency shifter
harmv2~	hamonizers
phaseshift1~	phase shifter
rev4~	reverberation
rmod1~	ring modulator
"Jimmies" Filters	
bpass1~	band-pass filter
comb2~	comb filters
hpass1~, hpass2~	1st and 2nd order high-pass filters
lpass1~, lpass2~	1st and 2nd order low-pass filters
bstop1~	notch filter

An example of an advanced Audio Effect module is given by the module psych~ mentioned above.

In the following a generic definition for Audio Processing modules is given as well as an example definition of the psych~ module.

Architecture and data flow

In general modules are mono (one input, one output) or stereo (two inputs, two outputs). Stereo effects such as chorus or reverberation may have mono input (one input, two outputs) and modules for multi-channel spatialisation have multiple outputs (one or two inputs, typically 4, 6 or 8 outputs).



Within a Max/MSP patch multiple Audio Effect modules can easily used in series or in parallel.

Modules Definition

Audio Effect module	
Description	Generic Audio Effect module
Instantiation Arguments	static parameters of the audio processing algorithm (e.g. gain)
Attributes	named static parameters of the extraction algorithm
Methods	messages to set parameters of the extraction algorithm
Inlets	inlet 1, 2, … (audio): audio input stream
Outlets	outlet 1, 2, … (audio): transformed audio output stream

psych~	
Description	Pitch transposition module for monophonic sounds using a PSOLA algorithm
Instantiation Arguments	arg 1: lowest transposed frequency
Attributes	none
Methods	pitch (list of floats): force one or multiple output pitches in MIDI cent freq (list of floats): force one or multiple output pitches in Hz trans (list of floats): set one or multiple transpositions MIDI cent respect (float): set spectral resampling amount in MIDI cent
Inlets	inlet 1 (audio): audio input stream
Outlets	outlet 1 (audio): pitch transposed audio output stream

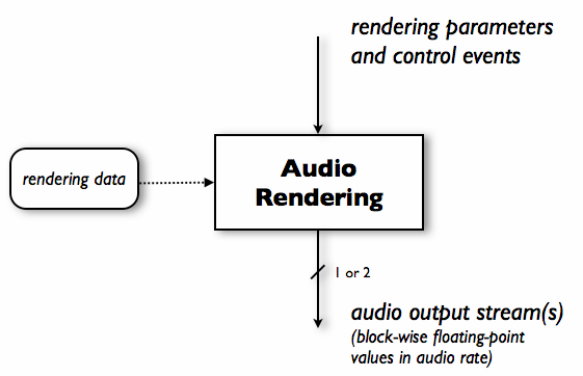
See above for the Max/MSP help patch of the psych~ module.

11.2.6 Audio Rendering Modules

The Practice Training environment includes audio rendering facilities such as sound file playing, rendering of Standard MIDI files using the General MIDI specification and various synthesis algorithms.

Architecture and data flow

Audio Rendering modules produce mono or stereo audio streams driven in real-time by control events and synthesis parameters. Player modules or resynthesis modules access rendering media such as sound or MIDI files in real-time (files based) or before starting the rendering process (RAM based).



The Max/MSP standard modules as well as the FTM module libraries give the possibility of separating data modules from synthesis/player modules, which refer to the rendering data (by name or reference). This allows for a modular and clearer definition of modules managing data (import/export, visualisation editing, etc.) and modules performing the actual rendering.

Module Definitions

The following modules are defined in the following:

- A generic audio file player module
- A generic MIDI file player module
- A generic General MIDI based synthesizer
- A generic synthesis module

Audio player	
Description	Generic integrated audio file player module
Instantiation Arguments	arg 1: number of output channels arg 2 (optional): file name
Attributes	none
Methods	open: open audio file through file chooser open (symbol): open audio file with a the given files name start: start or restart playing startat (number): start or start playing at position given in msec stop: stop playing and reset position to beginning pause: pause playing locate (number): set or jump to position given in msec speed: change playing speed
Inlets	no input apart from the defined messages
Outlets	outlet 1: audio stream (left channel for stereo files) outlet 2: audio stream right channel (for stereo files only)

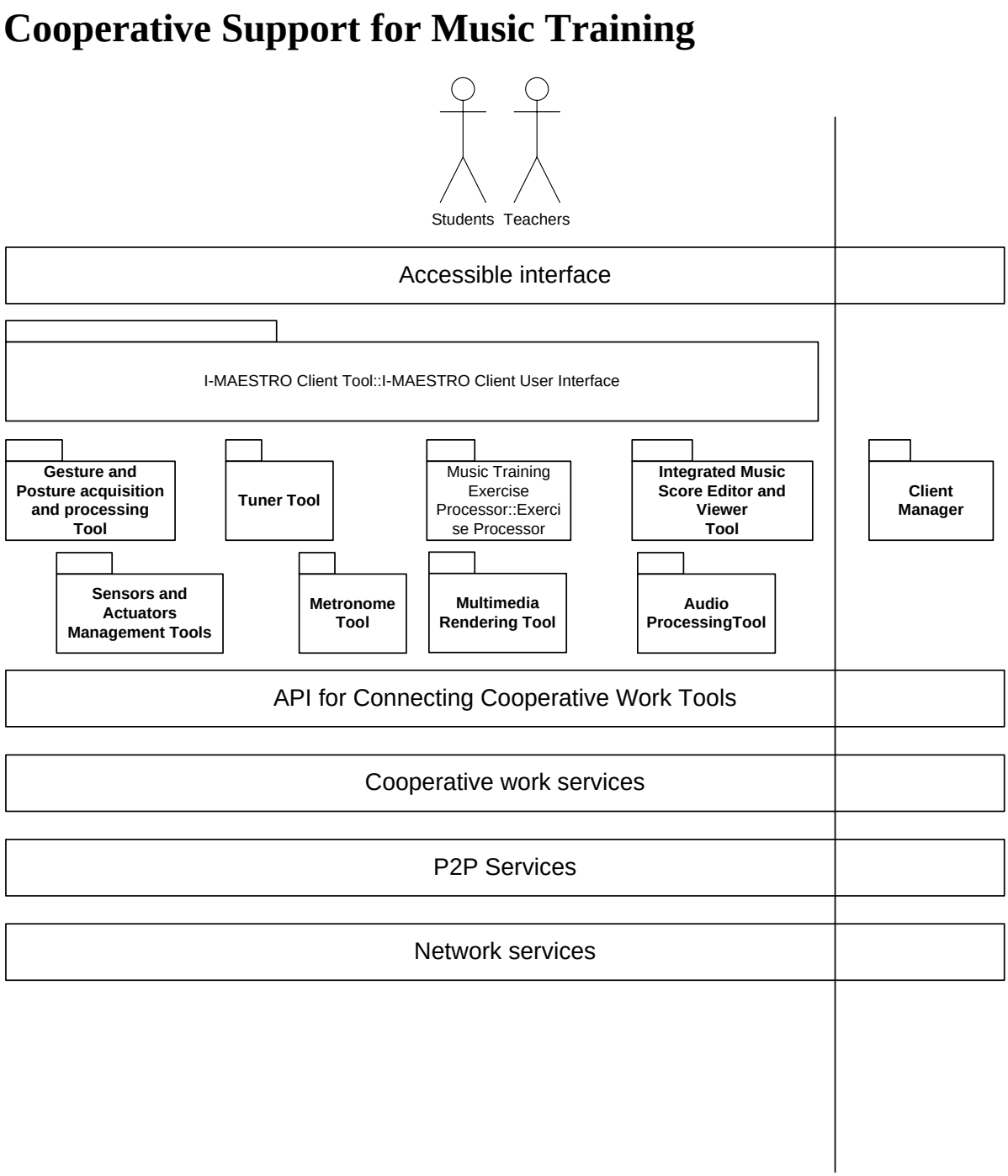
MIDI player	
Description	Generic integrated MIDI file player module
Instantiation Arguments	arg 1 (optional): file name
Attributes	none
Methods	open: open MIDI file through file chooser open (symbol): open MIDI file with a the given files name start: start or restart playing startat (number): start or start playing at position given in msec stop: stop playing and reset position to beginning pause: pause playing locate (number): set or jump to position given in msec speed: change playing speed
Inlets	no input apart from the defined messages
Outlets	outlet 1: stream of MIDI events

GM MIDI synthesizer	
Description	Generic MIDI based synthesizer module
Instantiation Arguments	none
Attributes	none
Methods	gain (number): level control in dB
Inlets	input 1: stream of MIDI events
Outlets	outlet 1: audio stream left channel outlet 2: audio stream right channel

Synthesis module	
Description	Generic synthesis module
Instantiation Arguments	static parameters of the synthesis algorithm
Attributes	dynamic parameters of the synthesis algorithm
Methods	messages to set parameters of the extraction algorithm
Inlets	input 1, 2, …: synthesis control parameters
Outlets	outlet 1, 2, …: audio output streams

12 Specification of Cooperative Support for Music Training

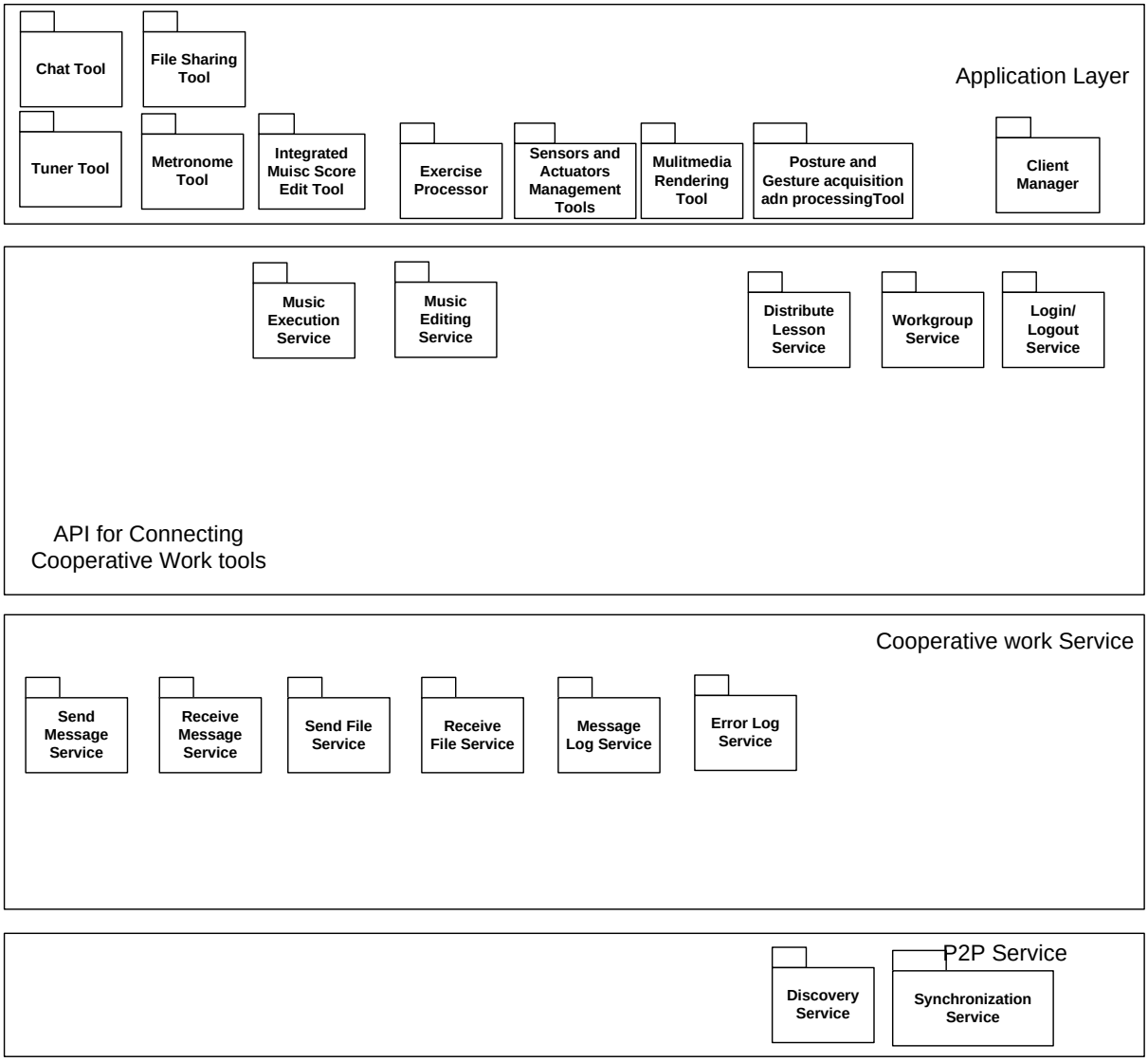
12.1 Cooperative work support overview



Teachers and Student 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.

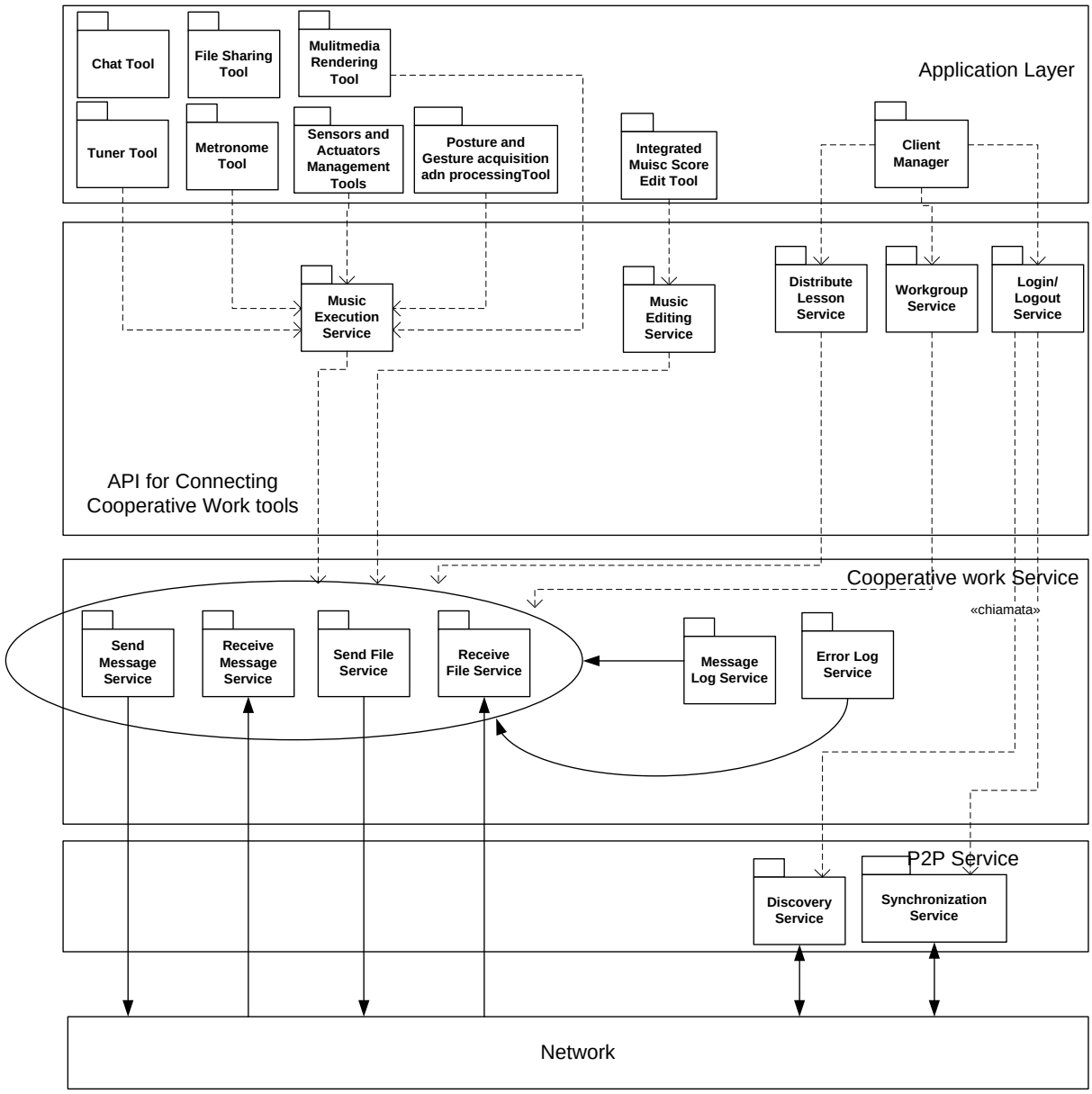
As showed in the previous figure **Cooperative Support for Music Training** contains three layers:

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

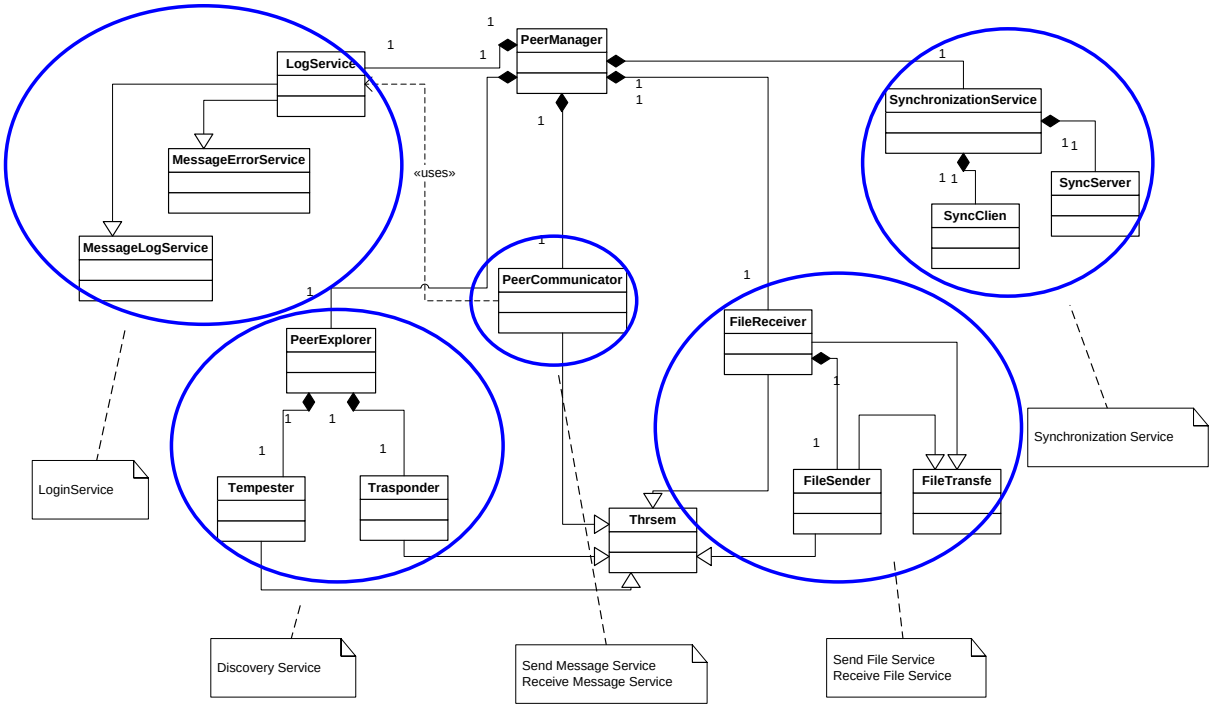


This figure represent a more detailed view of the Cooperative Support layers.
The **Application Layer** represents the tools with cooperative functionalities.
The **API for Connecting Cooperative Work tools** is a layer providing the cooperative functionalities which are implemented by the **Cooperative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

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


Application Layer
Chat Tool
File Sharing Tool
Mulitmedia Rendering Tool
Tuner Tool
Metronome Tool
Sensors and Actuators Management Tools
Posture and Gesture acquisition adn processingTool
Integrated Muisc Score Edit Tool
Client Manager
Music Execution Service
Music Editing Service
Distribute Lesson Service
Workgroup Service
Login/ Logout Service
API for Connecting Cooperative Work tools
Cooperative work Service
«chiamata»
Send Message Service
Receive Message Service
Send File Service
Receive File Service
Message Log Service
Error Log Service
P2P Service
Discovery Service
Synchronization Service
Network

12.2 Class Diagram of Cooperative Work Service and P2P Service



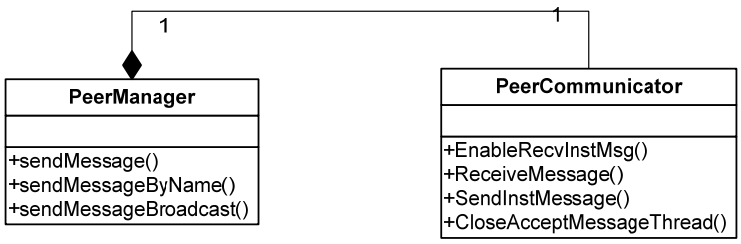
.

Cooperative Work Service and P2P Service involve the following classes:

- **PeerManager Class:** it wraps Cooperative Work Service and P2P Service and it separates hem from the API layer and Application Layer. It has method to start and stop the services and to exchange information with them.
- **MessageLogService** and **MessageErrorService:** implement the logic of the Log Service;
- **PeerExplorer:** define the behaviou of the Discovery Service using the classes Transponder and Tempester.
- **PeerCommunicator:** to send and receive messages (Send Message Service and Receive Message Service).
- **FileReceiver** and **FileSender:** to send and receive file
- **SynchronizationService:** it implements the algorithm and the thread to synchronize
- **ThrSem:** it provides support to implement threads

12.3 Send Message Service

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



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.

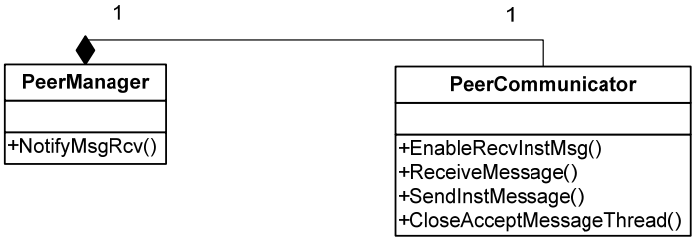
When a service want to send a message to all peer over the network it uses sendMessageBroadcast, which sends a message to all registered peer in P2P Network.

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 IpRecipientAddress – Ip Address of the recipient in the form xxx.xxx.xxx.xxx String message – the message to deliver
Output parameters	Int 0 if the message is sent correctly, -1 otherwise

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

12.4 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 IPaddresssender – The ip address of the message's sender String message – the message arrived
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

12.5 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.

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

trasferringFile	
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

getFileName	
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

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

12.6 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.

startReceiveFile	
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

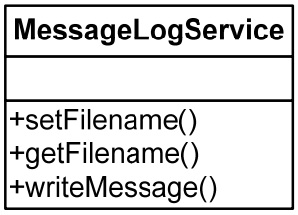
stopReceiveFile	
Method	stopReceiveFile
Description	It stops the thread waiting for connection coming from peers of the P2P Network
Input parameters	None
Output parameters	TRUE if the service stops successfully FALSE otherwise

receivingFile	
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

12.7 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 create 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



setFileName	
Method	setFileName
Description	Set 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

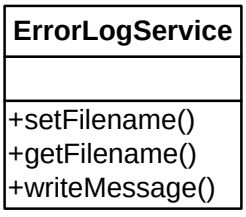
getFileName	
Method	getFileName
Description	Get the name of the file set to save messages
Input parameters	None
Output parameters	String File Name - name of the file where the log are saved

writeMessage	
Method	writeMessage
Description	It 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 arrive
Output parameters	TRUE if the writing succeeds FALSE otherwise

12.8 Error Log Service

Error Log Service maintains trace of all errors happened in the Cooperative Work 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 create 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 PeerManger 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



setFileName	
Method	setFileName
Description	Set 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

getFileName	
Method	getFileName
Description	Get 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

writeMessage	
Method	writeMessage
Description	It 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

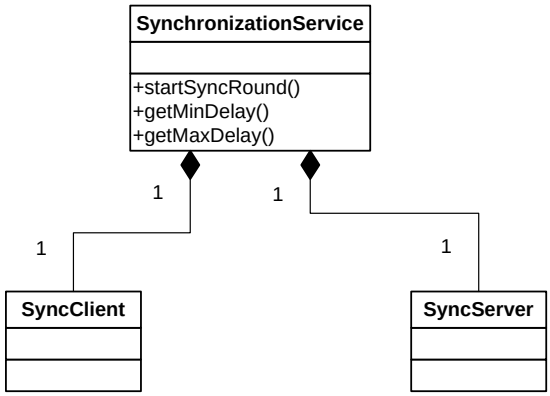
12.9 Discovery Service

Discovery Service is used to interrogate the P2P Network to find all connected peer. Discovery Service fills a PeerList where information of discovered peers is saved. It contains the name and ip address of the peer. PeerTable has to be available for API and Cooperative Work Service layers.

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 name 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

12.10 Synchronisation Service



This module uses a specific synchronization algorithm to maintain synchronization between peer of P2P Network. It can be used to synchronize a Max/MSP clock put inside each lesson; each tool refers to this clock to execute a command when they receive cooperative messages.

startSyncRound	
Method	startSyncRound
Description	It starts execution of the synchronization algorithm.
Input parameters	None
Output parameters	None

getMinDelay	
Method	getMinDelay
Description	It returns the minimum delay calculated by the synchronization algorithm.
Input parameters	None
Output parameters	Int delay – delay in milliseconds

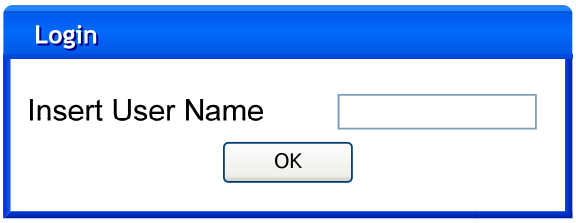
getMaxDelay	
Method	getMaxDelay
Description	It returns the maximum delay calculated by the synchronization algorithm.
Input parameters	None
Output parameters	Int delay – delay in milliseconds

13 Client Manager

The Client Manager 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 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.

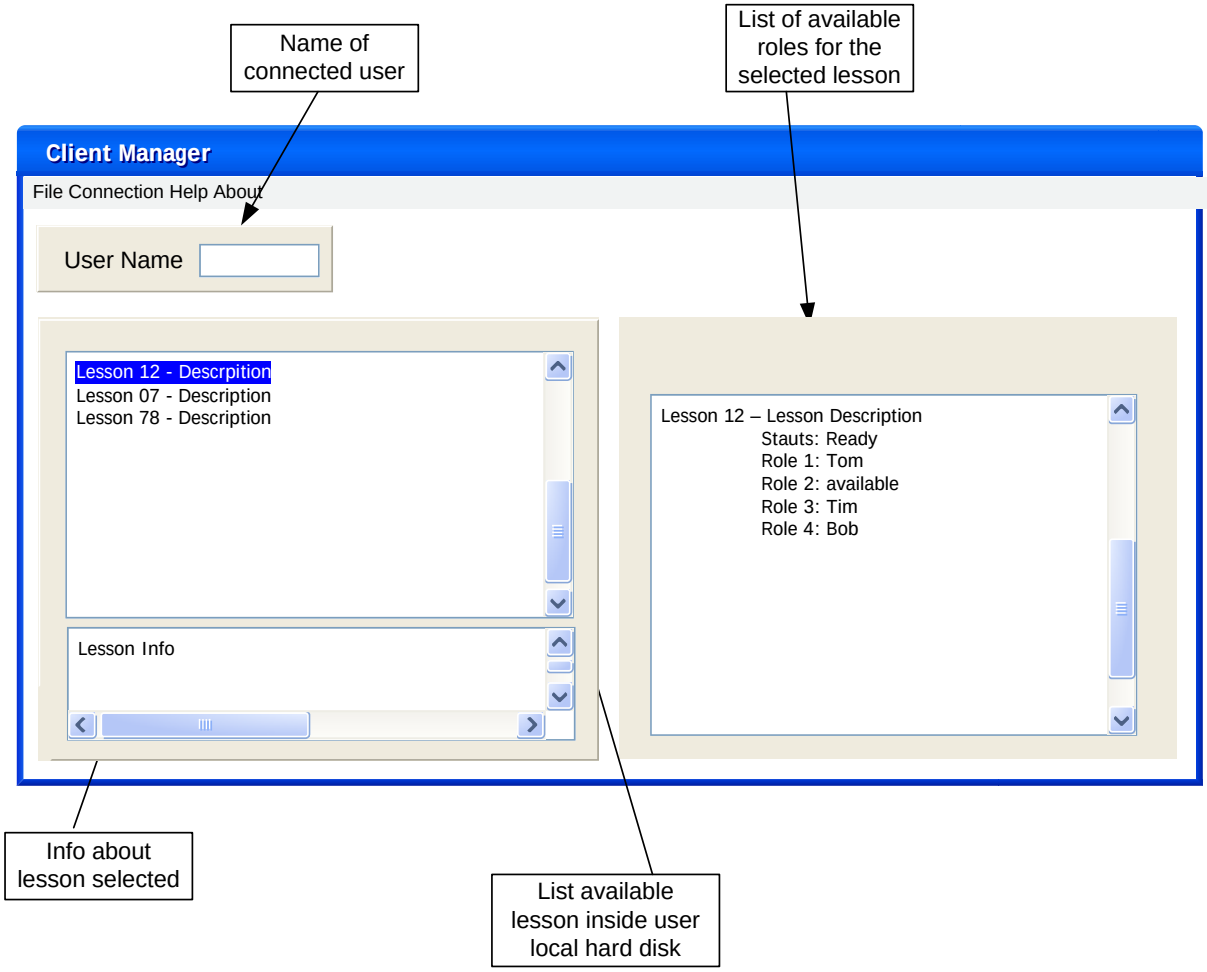


After the login the User has the possibility to see a list of lessons shared in the P2P network. After the selection of a Lesson, the User can see a description and a list of available role of the Lesson and can choose only one of them. After this choice, the Client Manager communicates to the other peer 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.

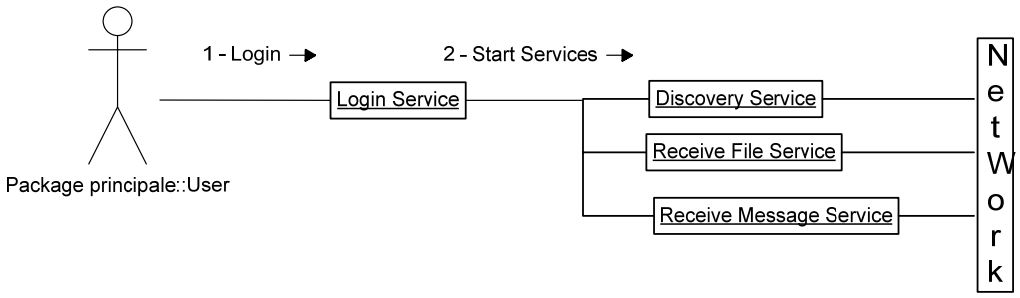
Lesson downloaded is automatically started by the Client Manager as an independent process. Now, the execution of the lesson doesn't depend from the Client Manager anymore, but it follows the logic decided by the lesson creator. Because we talk about cooperative lessons, each lesson has to have a Music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



Sequence of operations to setup a cooperative Lesson

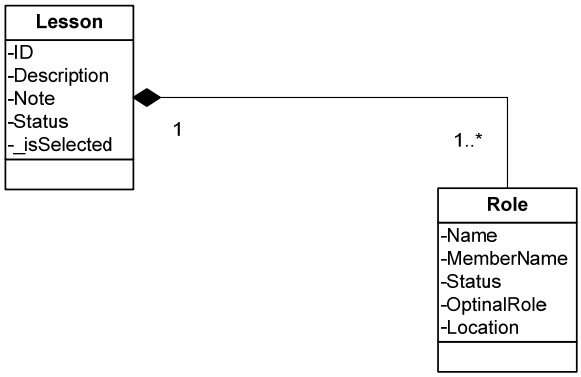
1) The User uses a Web Interface to search lesson inside a School Server and download its metadata information in xml format. Information is saved in a specific directory. This point doesn't concern the Client Manager and P2P Network.

2) When the user opens client manager, he chooses "Connect" from the "Connection" menu, inserts his name using the login form and the p2p services start.



3) The Client Manager automatically searches for the xml file of the lesson inside a specific directory (the previous one, where lesson information has been saved).

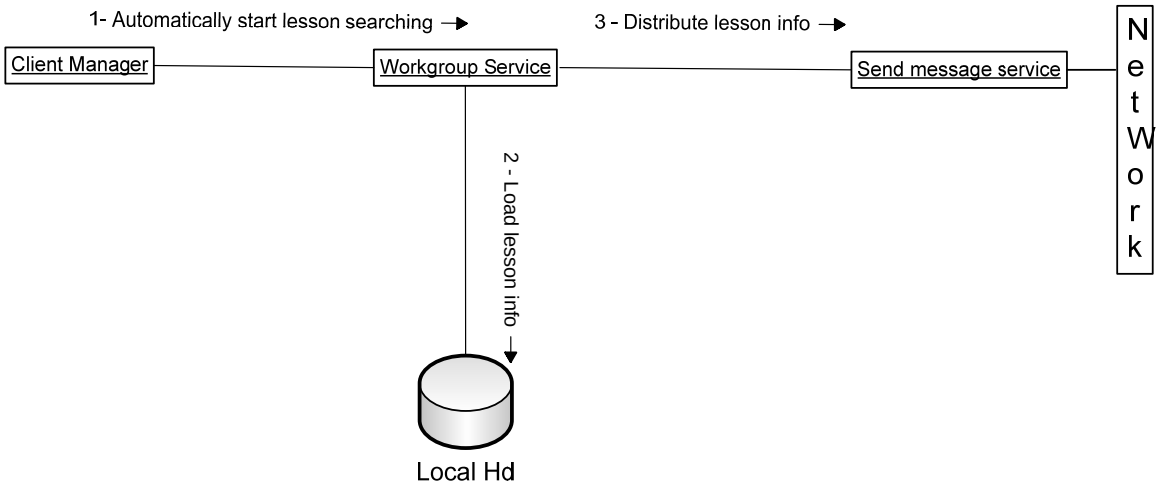
When the xml file is found and loaded, the ClientManager creates a Lesson object and Role objects depending on the info defined in the metadata. The Lesson objects are made by Roles objects and they are put in a Lesson vector. Note that lesson and role info are the same info described in the cooperative lesson metadata as in §13.5.



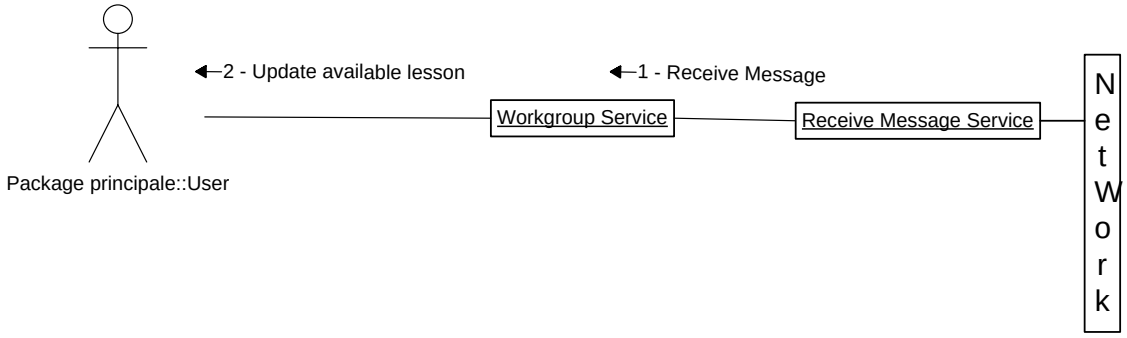
Then the Client Manager sends the xml metadata information to all peers in the P2P Network. In this way each peer can see the available lessons for cooperative work.

The format of the message is : SENDLESSON-<xml message of the lesson metadata>.

When a peer receives a message with SENDLESSON headline, it recovers the body part of the message, it parses the xml and it creates the correspondent lesson and roles objects and adds the lesson to the lesson vector.



4) Now the User receives the lesson arriving from the other peer and he can see roles linked to the lesson in his GUI.



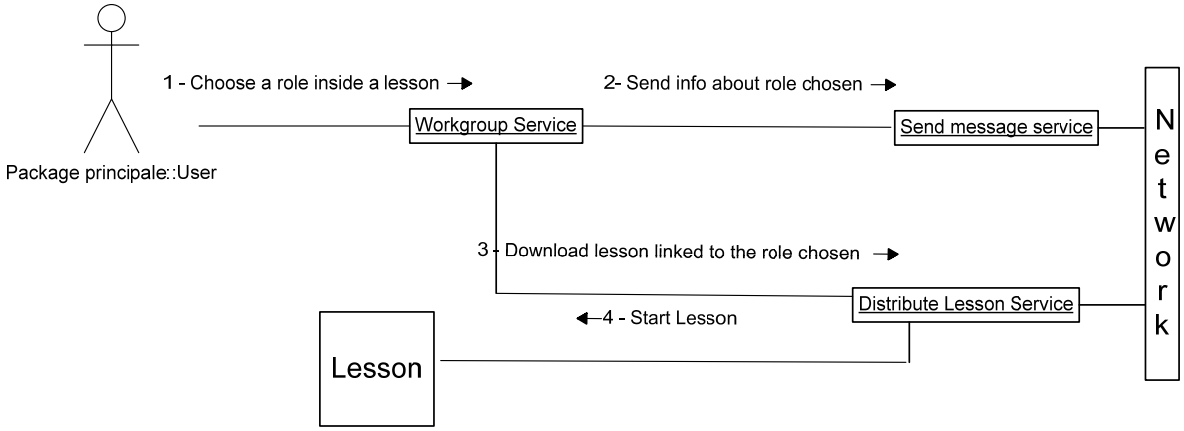
5) The User can select a lesson and see the list of available roles and can decide to associate only to one of these roles (the roles with the status READY). When User selects his role, the information is sent in the P2P network and all peers update the role info. The role pass from the status READY 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, lunching 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.

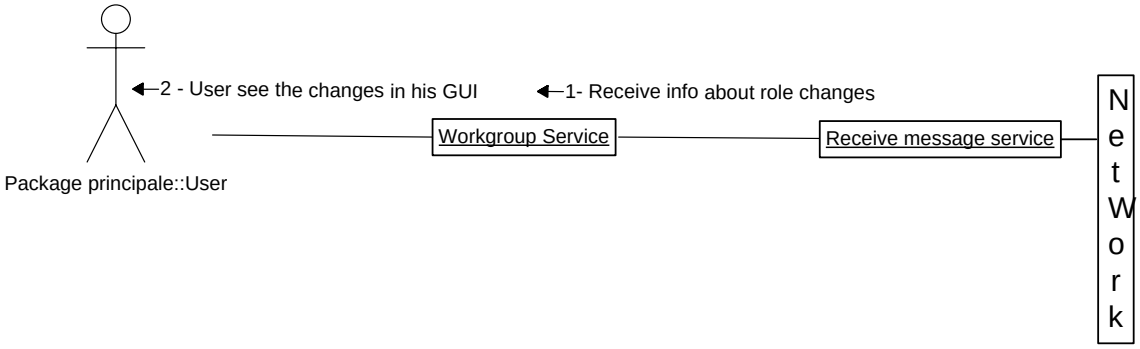
The format of the message is: ROLECHOOSE -<lessonid>-<rolename>-<username>.

The <lessonid> parameter corresponds to the "ID" parameter of the CooperativeLesson description metadata (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 update the username info.



6) All 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 singol 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 so it has no Music Execution Service with CWS Service and P2P service.

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 startup 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 its lesson id of the lesson chosen from the user in the Cleint Manager. The knowledge the lesson id is useful when the Music Execution Service send a message coming from one of the cooperative tool inside the lesson. In this way, before the string of the message, it is possible to insert the lesson id in this way:

<lessonid>-<message>:

1) <lessonid>: ID values of the lesson loaded and defined inside the metadata
2) <message>: any message type that the Music Execution Service sends in the P2P Network.

When a Music Execution Service of a generic lesson receives a message, the first step is to understand if the message is for itself or no. Comparing the <lessonid> of the message with the own lessonid , each lesson can accept only the its message and discard the others.

The Music Execution Service can also receive from the Client Manager the name of the User connected with each role. With these info, it is possible to manage the lesson execution, for example enabling or disabling users who are following the same lesson. Obviously this feature depends on the lesson logic defined by the lesson creator.

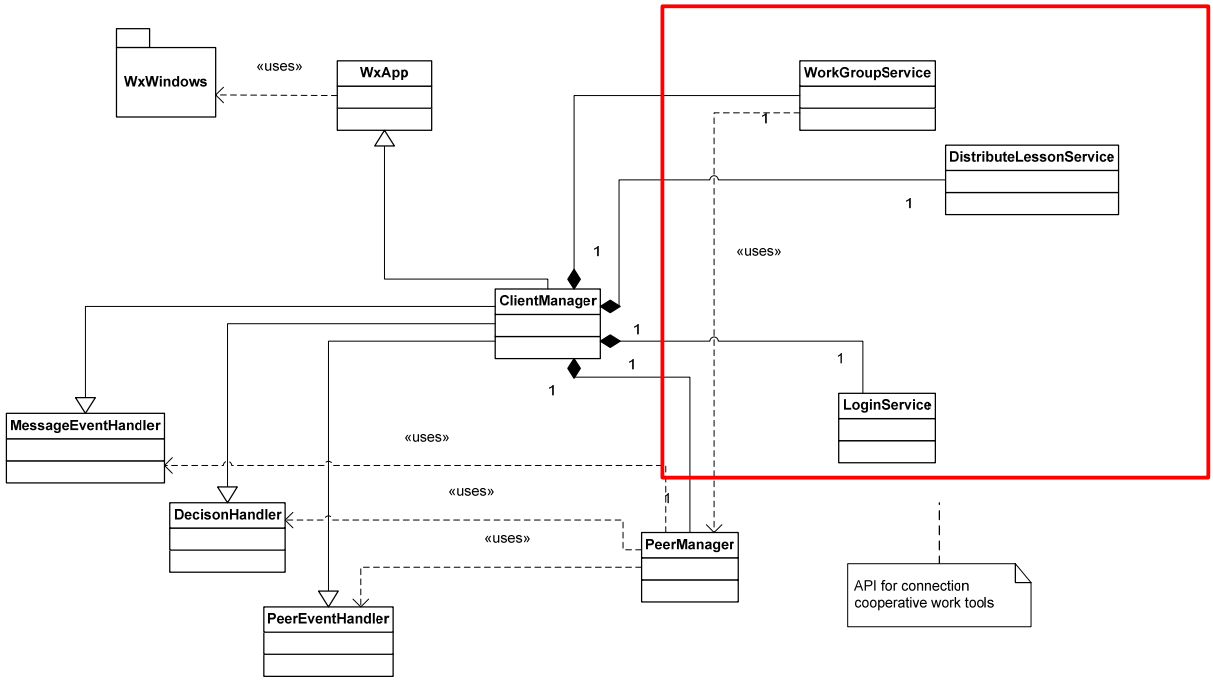
Then, after the correct check of <lessonid>, the <message> is parsed to understand:

1) The type of the command (START command, STOP command, a 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 correct outlet of the Music Execution Service and it is used from the other Max/MSP object linked to the outlet.

13.1 Class Diagram of ClientManager and its services



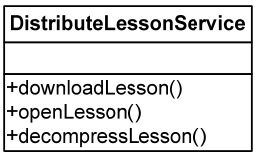
The figure shows the class diagram of the Client Manager and the services of the API layer that it uses.
Client Manager is a Wx application and it is composed by:

- **WorkGroupService:** it provides methods and functionalities to show cooperative lesson, add user to a selected role, remove user from a role, update 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.
- **LoginService**: it registers user name when a user opens the ClientManager and starts and stops services of P2P Layer.
- **PeerManager**: a class which links the ClientManager with the Cooperative work service and P2P services.

ClientManager also inherit from **MessageEventHandler**, **DecisionHandler** and **PeerEventHandler** to manage event append in the beneath layer.

13.2 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.



downloadLesson	
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

openLesson	
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

decompressLesson	
Method	decompressLesson
Description	It decompress the lesson and recreate 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

13.3 Login/Logout Service

With this module Students and Teacher can access to the P2P Network and attend to a cooperative sessions. Login Service requests user name when a User opens the Client manager, then it starts ad sets up services of Cooperative Work Service and P2P Service Layers.

login	
Method	login
Description	it save the name of the user and it starts the services of P2P Service Layer.
Input parameters	string user name- the user name used inside the p2p network
Output	None

parameters	

logout	
Method	logout
Description	Disconnect the user stopping all the active services of the P2P Service Layer. It is executed when user closes the Client Manager.
Input parameters	None
Output parameters	None

13.4 Workgroup Service

Workgroup Service keeps information about available lessons and roles inside P2P network. and maintain trace of available roles inside them. Info are updated automatically whenever a configuration change happens (e.g. a new user connects to an available role, or a user loads a new cooperative lesson).

addRoleUser	
Method	addRoleUser
Description	add the user to the Role chosen. Information is sent to all peer of the P2P Network.
Input parameters	String Lesson, - Lesson choosen by the user String Role, - Role choosen by the user String user_name – username to be linked to the previous role
Output parameters	Boolean - TRUE if user is correctly added FALSE otherwise

removeRoleUser	
Method	removeRoleUser
Description	Remove the present user from the Role selected. Information is sent to all peer of the P2P Network.
Input parameters	String lesson - Lesson choosen by the user String role - Role choosen by the user
Output parameters	Boolean - TRUE if user is correctly removed FALSE otherwise

addLesson	
Method	addLesson
Description	Add a new lesson to the lesson list with all its available roles. The method adds the lesson to the local lesson list of the peer and sends the lesson also to the other peer in P2P Network.
Input parameters	String Lesson info.
Output parameters	

getLessonList	
Method	getLessonList
Description	Request the list of available Lessons to a peer. When a new peer connects to the P2P Network, it requests the lesson list to one of the other peer. Since each peer is equal, all peer has the same lesson list, so it needs to only one connection to one peer to receive the full list.
Input parameters	None

Output parameters	String list – It is the list of available lesson info in xml format.

downloadLesson	
Method	downloadLesson
Description	It calls the Distribution Lesson Service to download the lesson linked to the role chosen
Input parameters	Lesson URI
Output parameters	None

13.5 Cooperative Session Data

In this section the information used to define a cooperative Lesson are described. Information are 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.

CooperativeLesson Description		
Data	**Type**	**Description**
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.
Role description		
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.
Status	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

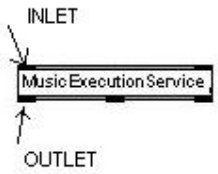
Cooperative Lesson XML formalisation

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=”Status" 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>
```

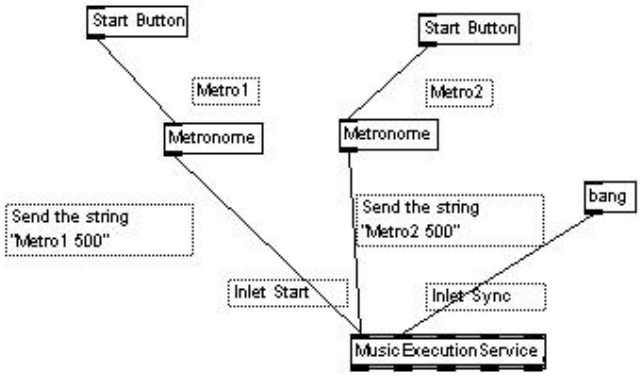
13.6 Music Execution Service

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.
The Object has several Inlets used to receive information from other Max Object and several Outlets used to send information to other objects.
Each Method described below represent an Inlet or an Outlet of the Music Execution Service.
To use this service is mandatory to create an object inside Max Designer Interface.
At this point each tool inside a cooperative lesson can communicate with other tool sending message to the right Inlets and receiving them from the right Outlets.



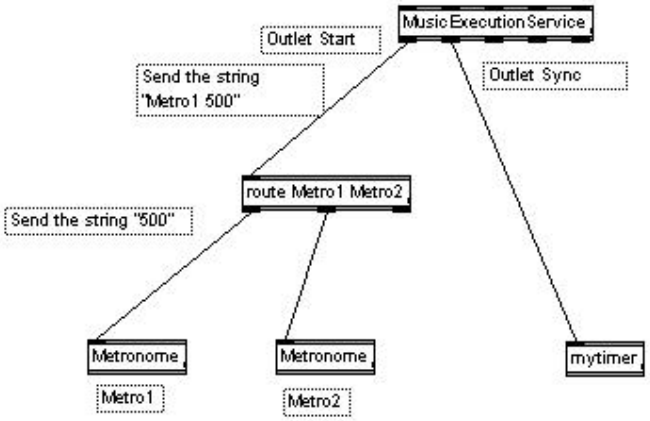
The figure above represents a prototype of Music Execution Service Max External. It will have more Inlets and Outlets, one for each provided functions.

An example of possible connection between the Music Execution Service and , e.g., two metronome tools (Metro1 and Metro2, they are Max/MSP external too) and a timer used for synchronization is displayed in the figures below:



Teacher Lesson

This image represents an example of the execution of a start command in a cooperative lesson. User presses the “start button” to start Metro1 and Metro1 send the command <ToolName> <delay> (in this case Metro1 500) to the Start Inlet of Music Execution Service Object. Music Execution Service distributes the command to all other peer of P2PNetwork.



Student Lesson

The figure of the Student Lesson shows the use of the same two metronomes (Metro1 and Metro2) of the previous example and one timer with the support of Music Execution Service.

The first outlet of the Music Execution Service represents the Start command arriving from the metronome inside another lesson (the Teacher Lesson).
When the start command arrives, it exits from the first outlet and it enters in the route object.
The route object, depending on the first parameter of the outlet Start, chooses the specific name of the recipient tool, and it delivers the remaining parameters (in this case the values of 500) to the correct tool.
In this case the Metro1 has received from another Metro1 the start command to be performed after 500 milliseconds; so the Metro1 will use the mytimer, which is synchronised with all the other mytimer of the lesson, to execute start command at the right moment.

To synchronise mytimer, the Teacher (see Teacher Lesson figure), send a "bang" message to the Sync Inlet. The synchronization process starts and at the end, a "bang" message exit from the Outlet Sync of each peer and the mytimer inside each lesson is reset.

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.

The Music Execution Service provides the following functions:

Inlet - Start	
Method	Inlet - Start
Description	Send a command in a cooperative way to start a specific tool. The command arrives from a tool which specified its name and the delay to wait before starting.
Input parameters	String ToolName - The name of the tool which has sent the message Integer delay - The delay to wait before the start execution
Output parameters	None
Sample Message	<ToolName> <delay>

Outlet - Start	
Method	Outlet - Start
Description	It sends the start command arriving from the P2P Layer to a Max/MSP Route or some other object to forward the command to the correct tool
Input parameters	None
Output parameters	String ToolName - The name of the tool which has sent the message Integer delay - The delay to wait before the start execution
Sample Message	<ToolName> <delay>

Inlet – Stop	
Method	Inlet – Stop
Description	Send a command in a cooperative way to stop a specific tool. It is specified the name of the tool to be stopped.
Input parameters	String ToolName - The name of the tool to be stopped
Output parameters	None

Request Sample Message	<ToolName>
Response Sample Message	None

Outlet - Stop	
Method	Outlet - Stop
Description	It sends the stop command arriving from the P2P Layer to a Max/MSP Route or some other object to forward the command to the correct tool
Input parameters	None
Output parameters	String ToolName - The name of the tool to be stopped
Request Sample Message	<ToolName>
Response Sample Message	None

Inlet- SendMessage	
Method	Inlet- SendMessage
Description	Send a generic message in a cooperative way to a specified tool and role
Input parameters	String Role - One of the role defined inside the lesson – if Role=ALL means that the message is for all roles in the lesson. String ToolName - The name of the tool which has sent the message String message – It contains the message to deliver to the Tool. The message is parsed by the specific tool.
Output parameters	None
Sample Message	It is a text string with: <Role> <ToolName> <message>
Response Sample Message	None

Outlet - ReceiveMessage	
Method	Outlet - ReceiveMessage
Description	It sends from the Outlet of Music Execution Service the message from the P2P network and it can use a Max/MSP Route or some other object to forward the command to the correct tool and/or role.
Input parameters	None
Output parameters	String Role - One of the role defined inside the lesson – Role=ALL means that the message is for all roles in the lesson. String ToolName - The name of the tool which has sent the message String message – It contains the message to deliver to the Tool. The message is parsed by the specific tool.
Sample Message	It is a text string with: <Role> <ToolName> <message>

Inlet - SetMetroSpeed	
Method	Inlet - SetMetroSpeed
Description	Send a command in a cooperative way to set the speed of a metronome
Input parameters	String ToolName – Name of the specific metronome Enum string value_of_note - Possible values are:2/1, 1/1, 1/2, 1/4, 1/8, 1/16, 1/32, 1/64 Integer beat_per_minute – it defines the metronome's speed
Output parameters	None
Sample Message	<ToolName><value_of_note> <beat_per_minute>

Outlet - GetMetroSpeed	
Method	Outlet - GetMetroSpeed
Description	Send from the Outlet of Music Execution Service a message from the P2P network and forward it to the specified metronome tool to set metronome's speed, using for example Max/MSP.
Input parameters	None
Output parameters	String ToolName – Name of the specific metronome Enum string value_of_note - Possible values are: 2/1, 1/1, 1/2, 1/4, 1/8, 1/16, 1/32, 1/64 Integer beat_per_minute – it defines the metronome's speed
Request Sample Message	<ToolName><value_of_note> <beat_per_minute>

Inlet - Sync	
Method	Inlet - Sync
Description	Send a message to start a synchronization round
Input parameters	String "bang" – Max/MSP command
Output parameters	None
Sample Message	It is a text string with: "bang" word.

Outlet - Sync	
Method	Outlet - Sync
Description	Send a "bang" message to reset the Max/MSP timer inside the lesson, used for the tool's synchronization
Input parameters	None
Output parameters	string **"**bang**"** – Max/MSP command
Sample Message	It is a text string with: "bang" word.

Inlet - getListRoles	
Method	Inlet - getListRoles
Description	Send a command to recover the list of roles inside the lesson.
Input parameters	None
Output parameters	None

Outlet - getListRoles	
Method	Outlet - getListRoles
Description	Receive the list of roles inside the lesson.
Input parameters	None
Output parameters	String roles list - a list of roles

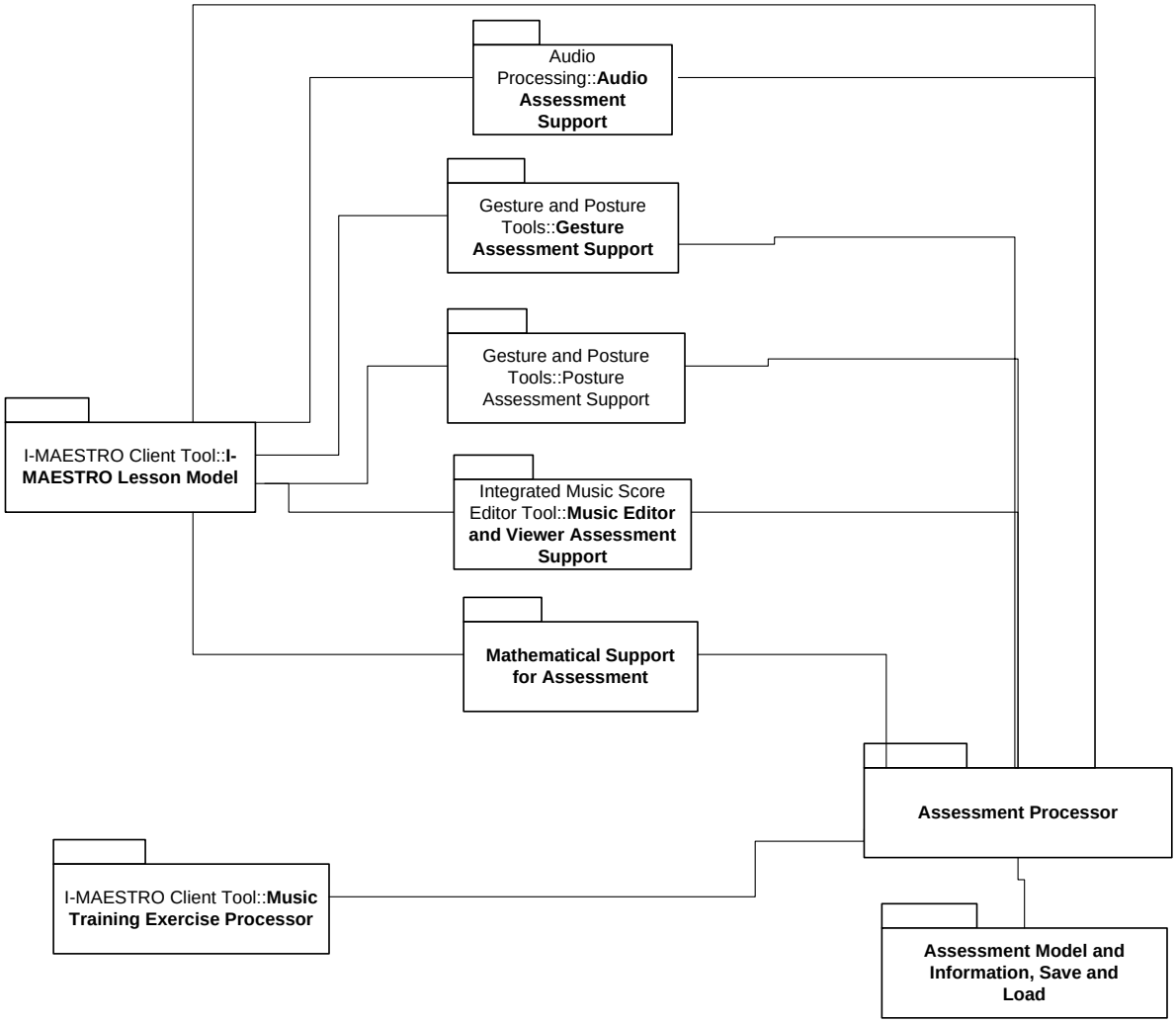
Inlet - getListConnectedRoles	
Method	Inlet - getListConnectedRoles
Description	Send a command to recover the list of connected roles inside the lesson.
Input parameters	None
Output parameters	None

Outlet - getListConnectedRoles	
Method	Outlet - getListRoles
Description	Receive the list of connected roles inside the lesson.
Input parameters	None
Output parameters	String roles list - a list of connected roles

14 Specification of Assessment Support

14.1 Assessment Support Overview

Assessment Support



Assessment Support applies assessment models taking into account profile, exercise, context, audio processing results, symbolic processing results, sensors, gesture and posture, etc.
The main modules are:

- **Assessment Processor**, which is able to manage the global assess merging various kind of assessment coming from tools Assessment Support(Gesture and Posture, Audio, Music Editor and Viewer). It is also can exchange command with the Music Training Exercise Processor.
- **Assessment Model and Information, Save and Load.** It represents the correct model and information to use for assess.
- **Mathematical Support for Assessment** provides support to Assessment Processor for calculating assessments.

14.2 General Assessment model and information

:
The purpose of assessment is to check what is being taught and to guide teaching to supervise progress toward skill achievement. The first step in any assessment model should focus on determining what the student knows or does not know. Based on this the teacher develops instructional materials corresponding to the student's level. In this initial assessment, a test model is constructed for each pedagogical purpose, and different scores are established to verify mastery and no mastery.

Assessment procedures should be designed to provide students a chance to demonstrate their capabilities and to improve them. Student evaluation involves making remarks, determining meters of the student achievement, keeping records and doing choices on the foundation of the collected student information. It is important for a efficient assessment model, to have a predetermined criteria to evaluate student's capabilities (the student scored should be based on a criteria predefined). No assessment tool is effective if is not used in a regular basis, and to develop effective assessment requires a constant revision based on feedback from the teachers and students.

For evaluation effectiveness is necessary to have an atmosphere in which all students are motivated to do their best and to focus on the assignment. The instructions should be clear and comprehensible; all the musical examples should be reproduced with good fidelity and should be clearly audible to every student. Besides each student should have the same amount of time to answer the questions, all necessary materials, instruments, and equipment should be available and in good working order. Ideally, when the test evaluation ask the student to sing, play instruments, or move, the student's response should be audiotaped (or videotaped better).

The teacher must decide the most appropriate time to set the assessing in each specific subject, taking into account:

- to estimated the necessary time to cover the key knowledge and skills.
- to estimated the length of time required for the students to complete the writings works, time to rehearsal, etc.
- If there is any connexion to others subjects that can interfered in the final results.
- Availability of the sources (library and others) facilities.

14.3 Pupil profile on the Server Side

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
   Document   : iMaestroStudent.xsd
   Author     : Fundación Albéniz
   Description:
      Definition of Student schema for I-MAESTRO project.
-->

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:annotation>
      <xsd:documentation xml:lang="en">
         Student Schema for I-MAESTRO project.
         Fundación Albéniz
      </xsd:documentation>
   </xsd:annotation>

   <xsd:include schemaLocation="iMaestroGeneric.xsd"/>

   <xsd:element name="studentProfile" type="StudentProfileType"/>

   <xsd:complexType name="StudentProfileType">
      <xsd:sequence>
```

```
        <xsd:element name="personalData" type="PersonalData"/>
        <xsd:element name="physicalInformation" type="PhysicalInformation"/>
        <xsd:element name="studentGoal" type="LearningGoal"/>
        <xsd:element name="generalSkills" type="GeneralSkills"/>
        <xsd:element name="musicalSkills" type="MusicalStudiesType"/>
    </xsd:sequence>
</xsd:complexType>

  <xsd:complexType name="MusicalStudiesType">
    <xsd:retriction base="xsd:string">
        <xsd:enumeration value="elemental"/>
        <xsd:enumeration value="professional"/>
        <xsd:enumeration value="highStudies"/>
        <xsd:enumeration value="phD"/>
    </xsd:retriction>
</xsd:complexType>

<xsd:simpleType name="GenderType">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="FEMALE"/>
        <xsd:enumeration value="MALE"/>
        <xsd:enumeration value="UNKNOWN"/>
    </xsd:restriction>
</xsd:simpleType>


<xsd:simpleType name="EthnicType">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="ASIAN"/>
        <xsd:enumeration value="BLACK"/>
        <xsd:enumeration value="WHITE"/>
        <xsd:enumeration value="UNKNOWN"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="HandicapType">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="TOTAL-BLIND"/>
        <xsd:enumeration value="PARTIAL-BLIND"/>
        <xsd:enumeration value="OTHER"/>
        <xsd:enumeration value="NONE"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="PreferenceHandType">
    <xsd:restriction base="xsd:string">
        <xsd:pattern value="leftHanded|rightHanded"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="PersonalData">
    <xsd:sequence>
        <xsd:element name="firstName" type="xsd:string"/>
        <xsd:element name="lastName" type="xsd:string"/>
        <xsd:element name="birthDate" type="xsd:date"/>
        <xsd:element name="sex" type="GenderType"/>
        <!-- it could be defined or reused a LanguageType for
        motherLanguage -->
        <xsd:element name="motherTongue" type="xsd:string"/>
        <xsd:element name="ethnic" type="EthnicType" />
        <xsd:element name="physicalTuition" type="xsd:boolean"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="PhysicalInformation">
    <xsd:sequence>
        <xsd:element name="weight" type="xsd:decimal"/>
        <xsd:element name="height" type="xsd:decimal"/>
        <!-- Usual practice of fitness -->
        <xsd:element name="fitness" type="xsd:boolean"/>
        <xsd:element name="handicap" type="HandicapType" minOccurs="0"/>
        <xsd:element name="handType" type="PreferenceHandType"/>
    </xsd:sequence>
</xsd:complexType>
```

```
  <xsd:simpleType name="InterestType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="PROFESSIONAL"/>
      <xsd:enumeration value="AMATEUR"/>
      <xsd:enumeration value="DILETTANTE"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="DedicationType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="FULLTIME"/>
      <xsd:enumeration value="PARTTIME"/>
      <xsd:enumeration value="SPORADIC"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:complexType name="LearningGoal">
    <xsd:sequence>
      <xsd:element name="interest" type="InterestType"/>
      <xsd:element name="dedication" type="DedicationType"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:simpleType name="StudiesType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Primary"/>
      <xsd:enumeration value="Secondary"/>
      <xsd:enumeration value="University"/>
      <xsd:enumeration value="PhD"/>
      <xsd:enumeration value="None"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="LanguageType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Italian"/>
      <xsd:enumeration value="German"/>
      <xsd:enumeration value="English"/>
      <xsd:enumeration value="French"/>
      <xsd:enumeration value="Russian"/>
      <xsd:enumeration value="Spanish"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="PerformingArtsType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="theatre"/>
      <xsd:enumeration value="dance"/>
      <xsd:enumeration value="other"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:complexType name="GeneralSkills">
    <xsd:sequence>
      <xsd:element name="computers" type="ValorationRange"/>
      <xsd:element name="studies" type="StudiesType"/>
      <xsd:element name="languages" type="LanguageType"/>
      <xsd:element name="performingArts" type="PerformingArtsType" minOccurs="0"/>
  </xsd:complexType>
</xsd:schema>
```

14.4 Local History and Data on the Client Side

The main purpose of the historical data is to validate the evolution of the Student's progress through all the data collected by the system. The student evaluation should be continuous processes which follow the progress of a student over a significant period of time. For this reason, all the student's results of the "skill and capacities" will be check it and save it by the system, and it will be possible for the student to see in short or long term his results (i.e. last two weeks, form January till Mach, etc).

The historical data (from the students) will determinate in which areas the student is (or not) progressing.

Check and analyse the improving percentages. All this scores will be collected and enter into the database, so it can give meaningful information to the teacher to help drive instruction. Monitor progress will be use to assess students academic performance and evaluate the effectiveness of instruction. (Also monitor progress will be implemented with individual students or group students).
The learning process of the students will be collected in three different categories:

1 Initial evaluation: Review of the initial technical and theoretical knowledge of the student.
2 Formative evaluation: Assessment of the speed of assimilation and comprehension of the knowledge that the student is acquiring. The student's academic achievements will be measured on a regular basis (weekly or monthly).
3 Final evaluation: Assessment of the knowledge acquired during the learning process.

Client-side history and data will be needed to run the different exercises and lessons, providing the necessary data to each particular design and programming. It should be fully compatible, syncronized and able to update the server side information.

14.5 Assessment Support for Symbolic Training

Assessment task for symbolic training assess the capacity to recall information from the student's knowledge, and the capabilities to analyse and synthesis the new information and concepts. For symbolic training is important to assess the development of the following skills in each student:

- Ability to read at sight music of all the periods (including early notation and twentieth century)
- Ability to recall important works, composer of Western Music, sources, different aspects of the major historical periods and styles.
- Ability to place music in its proper cultural context
- Ability to recognized and described properly a number of representative compositions from the major historical periods and styles of Western Music.
- Ability to write music, demonstrating and understanding theoretical terms, symbols, concepts (demonstrate knowledge of traditional and modern compositional styles and capability to transfer it to one's own individual style of composition).
- Ability to instrumentation, arranging skills and score knowledge for various instrumental ensembles.
- etc.

14.6 Assessment Support for Practice Training

Practice Trainings are learning programs to teach the fundamental principles of the musical practice: rhythm training, ear training, development principles of movements, etc. The main goal of evaluation in the practice training model is to check the study progression and to consolidate the musical principles. The techniques of evaluation will depend on its purpose (professional or amateur) and subject.

An example of a way to evaluate the general skills in practice training is to measure the students' control capabilities in different subjects like:

- Musical performance (or basic principles of musicianship):
  - Evaluate the student ability to:
  - Adequate use of dynamics, articulation, accents and breathing, know when to make retards, etc.
  - Phrasing (types, emotional content, beginning, ending, developing, etc.)

  - Perform appropriately representative repertory of the instrument, demonstrating musicianship, technical proficiency, and interpretative understanding
  - Historical approach of genres: 17th and 18th centuries, 19th century, 20th century Perform in small and large ensembles.
- Improvisation Skills
  - Evaluate the student ability to:
  - Improvise on the instrument (making the accompaniment or the principal voice) Compose and develop original melodies appropriate to the instrument, accompaniments, and short pieces in a variety of genres and styles vocally, and instrumentally).
  - Improvise in small ensembles.

- Movement Skills
  - Evaluate the student's ability to:
  - Coordination
  - Flexibility degree
  - Ability of the mind to prepare-supervise the muscular activity
  - Right use of awareness/preparatory movement exercises before playing.

An example of a way to evaluate the basic skills for bowed string instruments is to measure the students' control capabilities in different subjects like:

- Playing position: posture, instrument hold
- Left hand: fingering, position and shifting, vibrato, harmonics, left hand pizzicato
- Right hand: tone production, bowings,
- Etc

14.7 XML Schema for Symbolic and Practice Training Assessment

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
   Document   : iMaestroAssessmentParameters.xsd
   Author     : Fundación Albéniz
   Description:
      Purpose of XML Schema document follows.
-->

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:annotation>
      <xsd:documentation xml:lang="en">
         Assessment Parameters for I-MAESTRO project.
         Fundación Albéniz
      </xsd:documentation>
   </xsd:annotation>

   <xsd:include schemaLocation="iMaestroGeneric.xsd"/>

   <xsd:group name="instrumentPerformanceSkills">
      <xsd:sequence>
         <xsd:element name="technique" type="ValorationRange"/>
         <xsd:element name="execution" type="ValorationRange"/>
         <xsd:element name="expression" type="ValorationRange"/>
      </xsd:sequence>
   </xsd:group>

   <xsd:group name="bowPlayerPostureAssessment">
      <xsd:sequence>
```

```
        <xsd:element name="bodyCorrectness" type="xsd:boolean"/>
        <xsd:element name="leftArmCorrectness" type="xsd:boolean"/>
        <xsd:element name="rightArmCorrectness" type="xsd:boolean"/>
        <xsd:element name="leftElbowCorrectness" type="xsd:boolean"/>
        <xsd:element name="rightElbowCorrectness" type="xsd:boolean"/>
        <xsd:element name="leftWristCorrectness" type="xsd:boolean"/>
        <xsd:element name="rightWristCorrectness" type="xsd:boolean"/>
        <xsd:element name="leftHandCorrectness" type="xsd:boolean"/>
        <xsd:element name="rightHandCorrectness" type="xsd:boolean"/>
        <xsd:element name="feetPositionCorrectness" type="xsd:boolean"/>
    </xsd:sequence>
</xsd:group>

<xsd:group name="instrumentLeftHandAssessment">
    <xsd:sequence>
        <xsd:element name="fingeringPosition" type="ValorationRange"/>
        <xsd:element name="leftThumbTechnique" type="ValorationRange"/>
        <xsd:element name="fingerFlexibility" type="ValorationRange"/>
        <xsd:element name="fingerPressure" type="ValorationRange"/>
        <xsd:element name="fingerIndependence" type="ValorationRange"/>
        <xsd:element name="trills" type="ValorationRange"/>
        <xsd:element name="doubleStops" type="ValorationRange"/>
        <xsd:element name="positionShifting" type="ValorationRange"/>
        <xsd:element name="firstPosition" type="ValorationRange"/>
        <xsd:element name="secondPosition" type="ValorationRange"/>
        <xsd:element name="thirdPosition" type="ValorationRange"/>
        <xsd:element name="fourthPosition" type="ValorationRange"/>
        <xsd:element name="fifthPosition" type="ValorationRange"/>
        <xsd:element name="higherPositions" type="ValorationRange"/>
        <xsd:element name="scaleTechnique" type="ValorationRange"/>
        <xsd:element name="vibratoTechnique" type="ValorationRange"/>
        <xsd:element name="nonTemperedIntonation" type="ValorationRange"/>
        <xsd:element name="temperedIntonation" type="ValorationRange"/>
        <xsd:element name="naturalHarmonics" type="ValorationRange"/>
        <xsd:element name="artificialHarmonics" type="ValorationRange"/>
        <xsd:element name="pizzicato" type="ValorationRange"/>
    </xsd:sequence>
</xsd:group>

<xsd:group name="instrumentRightHandAssessment">
    <xsd:sequence>
        <xsd:element name="toneProduction" type="ValorationRange"/>
        <xsd:element name="appropiateBowing" type="ValorationRange"/>
        <xsd:element name="stringCrossing" type="ValorationRange"/>
        <xsd:element name="detaché" type="ValorationRange"/>
        <xsd:element name="legato" type="ValorationRange"/>
        <xsd:element name="portato" type="ValorationRange"/>
        <xsd:element name="loure" type="ValorationRange"/>
        <xsd:element name="collé" type="ValorationRange"/>
        <xsd:element name="martellato" type="ValorationRange"/>
        <xsd:element name="martelé" type="ValorationRange"/>
        <xsd:element name="spiccato" type="ValorationRange"/>
        <xsd:element name="flyingSpicatto" type="ValorationRange"/>
        <xsd:element name="saltellato" type="ValorationRange"/>
        <xsd:element name="sautille" type="ValorationRange"/>
        <xsd:element name="ricochet" type="ValorationRange"/>
        <xsd:element name="hookedBowing" type="ValorationRange"/>
        <xsd:element name="rimbalzato" type="ValorationRange"/>
        <xsd:element name="tremolo" type="ValorationRange"/>
        <xsd:element name="sulPonticello" type="ValorationRange"/>
        <xsd:element name="colLegno" type="ValorationRange"/>
        <xsd:element name="doubleStops" type="ValorationRange"/>
        <xsd:element name="accordAttack" type="ValorationRange"/>
        <xsd:element name="pizzicato" type="ValorationRange"/>
    </xsd:sequence>
</xsd:group>

<xsd:group name="cooperativeWorkAssessment">
    <xsd:sequence>
        <xsd:element name="leading" type="ValorationRange"/>
        <xsd:element name="dynamicsCoordination" type="ValorationRange"/>
        <xsd:element name="intonationCoordination" type="ValorationRange"/>
        <xsd:element name="timberCoordination" type="ValorationRange"/>
        <xsd:element name="tempoCoordination" type="ValorationRange"/>
        <xsd:element name="followingDirector" type="ValorationRange"/>
```

```
        <xsd:element name="phrasingImitation" type="ValorationRange"/>
        <xsd:element name="bowsCoordination" type="ValorationRange"/>
    </xsd:sequence>
</xsd:group>

<xsd:group name="classroomAssessment">
    <xsd:sequence>
        <xsd:element name="activeStudentNumber" type="xsd:integer"/>
        <xsd:element name="listenerStudentNumber" type="xsd:integer"/>
    </xsd:sequence>
</xsd:group>

<xsd:group name="symbolicTrainingAssessment">
    <xsd:sequence>
        <xsd:element name="sightReading" type="ValorationRange"/>
        <xsd:element name="sightSinging" type="ValorationRange"/>
        <xsd:element name="rhytmicalReading" type="ValorationRange"/>
        <xsd:element name="transposedReading" type="ValorationRange"/>
        <xsd:element name="rhytmicalReading" type="ValorationRange"/>
        <xsd:element name="polyRhythm" type="ValorationRange"/>
        <xsd:element name="dictationAbilities" type="ValorationRange"/>
        <xsd:element name="intonationAccuracy" type="ValorationRange"/>
        <xsd:element name="harmonicEar" type="ValorationRange"/>
        <xsd:element name="perfectPitch" type="ValorationRange"/>
        <xsd:element name="intervalsRecognition" type="ValorationRange"/>
        <xsd:element name="innerEar" type="ValorationRange"/>
        <xsd:element name="chordsDictation" type="ValorationRange"/>
        <xsd:element name="atonalDictation" type="ValorationRange"/>
        <xsd:element name="formalAnalysis" type="ValorationRange"/>
        <xsd:element name="harmonicAnalysis" type="ValorationRange"/>
        <xsd:element name="melody4VoicesHarmonisation" type="ValorationRange"/>
        <xsd:element name="bass4VoicesHarmonisation" type="ValorationRange"/>
        <xsd:element name="musicalForms" type="ValorationRange"/>
        <xsd:element name="modulationAndProgression" type="ValorationRange"/>
        <xsd:element name="middleAgeKnowledge" type="ValorationRange"/>
        <xsd:element name="reinassanceKnowledge" type="ValorationRange"/>
        <xsd:element name="baroqueKnowledge" type="ValorationRange"/>
        <xsd:element name="classicKnowledge" type="ValorationRange"/>
        <xsd:element name="romanticKnowledge" type="ValorationRange"/>
        <xsd:element name="impressionisticKnowledge" type="ValorationRange"/>
        <xsd:element name="XXCenturyKnowledge" type="ValorationRange"/>
        <xsd:element name="ethnicMusicKnowledge" type="ValorationRange"/>
        <xsd:element name="popKnowledge" type="ValorationRange"/>
        <xsd:element name="jazzKnowledge" type="ValorationRange"/>
        <xsd:element name="flamencoKnowledge" type="ValorationRange"/>
        <xsd:element name="electroacusticsKnowledge" type="ValorationRange"/>
        <xsd:element name="impressionisticKnowledge" type="ValorationRange"/>
        <xsd:element name="classicHarmony" type="ValorationRange"/>
        <xsd:element name="counterpoint" type="ValorationRange"/>
        <xsd:element name="orchestration" type="ValorationRange"/>
        <xsd:element name="creativity" type="ValorationRange"/>
    </xsd:sequence>
</xsd:group>

<xsd:group name="generalPracticeAssessment">
    <xsd:sequence>
        <xsd:element name="rhythmMemory" type="ValorationRange"/>
        <xsd:element name="melodicMemory" type="ValorationRange"/>
        <xsd:element name="harmonicMemory" type="ValorationRange"/>
        <xsd:element name="feelMeaningMemory" type="ValorationRange"/>
        <xsd:element name="projectRxpressiveness" type="ValorationRange"/>
        <xsd:element name="imagination" type="ValorationRange"/>
        <xsd:element name="dynamics" type="ValorationRange"/>
        <xsd:element name="articulation" type="ValorationRange"/>
        <xsd:element name="accents" type="ValorationRange"/>
        <xsd:element name="breathing" type="ValorationRange"/>
        <xsd:element name="phrasing" type="ValorationRange"/>
        <xsd:element name="historicalContextSuitability" type="ValorationRange"/>
        <xsd:element name="melodicImprovisation" type="ValorationRange"/>
        <xsd:element name="groupImprovisation" type="ValorationRange"/>
        <xsd:element name="stypeImprovisation" type="ValorationRange"/>
        <xsd:element name="movementCoordination" type="ValorationRange"/>
        <xsd:element name="flexibilityMovement" type="ValorationRange"/>
    </xsd:sequence>
</xsd:group>
```

```
  <xsd:complexType name="practiceTrainingAssessmentParameter">
    <xsd:sequence>
      <xsd:group ref="instrumentPerformanceSkills" maxOccurs="1"/>
      <xsd:group ref="bowPlayerPostureAssessment" maxOccurs="1" />
      <xsd:group ref="instrumentLeftHandAssessment" maxOccurs="1" />
      <xsd:group ref="instrumentRightHandAssessment" maxOccurs="1" />
      <xsd:group ref="cooperativeWorkAssessment" maxOccurs="1" />
      <xsd:group ref="classroomAssessment" maxOccurs="1" />
      <xsd:group ref="symbolicTrainingAssessment" maxOccurs="1" />
      <xsd:group ref="generalPracticeAssessment" maxOccurs="1" />
    </xsd:sequence>
  </xsd:complexType>


</xsd:schema>
```

14.8 Assessment model in Class Rooms

This model presents a wide variety approach towards music tuition or educational courses. The class room models include several possible forms like: teacher to student, teacher to many students, etc.
In the class room setting, the musical knowledge and skills will be measured by different collective test form to determine the level of technical skills, ear test and musical hearing, etc. There are different methods to recording observations in the classroom setting:

- Particular records: The teacher makes notes about the student's progress (i.e. student's work habits, contributions and discussions whit the others students, etc.)
- Rating scales: The teacher completes for each student a rating scale from the student progress in different subjects.
- Checklist: checklist of observable activities (i.e. ability to listen others, play together, etc.)

Usually to measure the student's knowledge and skills in creating and performing (i.e. artistic abilities, repertoire), the students are assessed individually.

14.9 Assessment model for self assessment

Opportunities are limited for students who wish continue learning outside of a classroom setting. For this reason, different educational trainings are accessible for these students and also several self assessment models. The self assessment model should review all the basic skills and principles of the instrumental technique and theoretical knowledge.
A possible scale for self assessment model can include the satisfaction level form the student in different subjects (i.e. scale from 1: very dissatisfied to 5: very satisfied), and measurements of effort (i.e. a scale from 1: very hard to 5: very easy)

14.10 Assessment model for Cooperative work

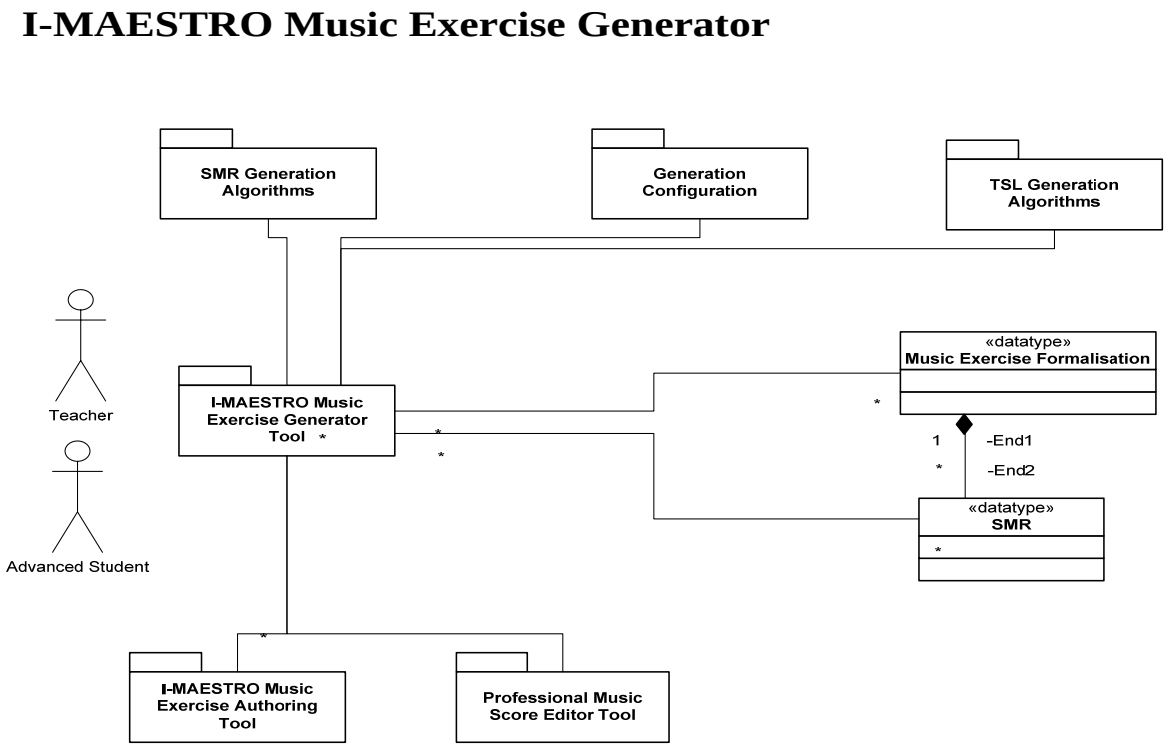
Cooperative classes and workshops models favour a group approach towards instruction. This training can assist many students at once in synchronous string classes, coaching sessions, master classes, and group performances. In this model opportunities are given for group of students to communicate with one another for discussion groups, chamber music experience, etc. Thought group playing the student learn the skills required to perform in an ensemble. The way to evaluate them is by checking between sections of the group, orchestra and soloists, etc.

14.11 Assessment Support User Interface

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
   Document   : iMaestroUserInterface.xsd
   Author     : Fundación Albéniz
   Description:
      Metadata of lessons from teacher pespective.
-->

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:annotation>
      <xsd:documentation xml:lang="en">
         User Interface Parameters for I-MAESTRO project.
         Fundación Albéniz
      </xsd:documentation>
   </xsd:annotation>

   <xsd:include schemaLocation="iMaestroStudent.xsd"/>
   <xsd:include schemaLocation="iMaestroAssessmentParameters.xsd"/>

   <xsd:element name="exerciseParameters" type="ExerciseParametersType"/>

   <xsd:complexType name="ResultType">
      <xsd:restriction base="xsd:string">
         <xsd:enumeration value="newExercise"/>
         <xsd:enumeration value="repeat"/>
         <xsd:enumeration value="re-evaluate student"/>
         <xsd:enumeration value="modify assessment"/>
         <xsd:enumeration value="change exercise type"/>
      </xsd:restriction>
   </xsd:complexType>

   <xsd:complexType name="ExerciseParametersType">
      <xsd:sequence>
      <xsd:element name="goal" type="xsd:string"/>
      <xsd:element name="mandatory" type="xsd:boolean"/>
      <xsd:element name="storable" type="xsd:boolean"/>
      <xsd:element name="targettedStudent" type="StudentProfileType"/>
      <xsd:element name="assessmentParameters" type="AssessmentParametersType"/>
      <xsd:element name="courseFlow" type="ResultType" maxOccurs="1"/>
      </xsd:sequence>
   </xsd:complexType>
```

15 Specification of I-MAESTRO Music Exercise Generator

15.1 Music Exercise Generator overview



The I-MAESTRO Exercise Generator is part of the I-MAESTRO Production Tools. It is complementary to the Music Exercise Authoring Tool in providing automated generation of Exercises. The main outputs produced by the Exercise Generator are Music Exercise Formalisations in Training Specification Language (TSL) and Music Notation in MPEG Symbolic Music Representation (SMR), which can be used by the Music Exercise Formalisation.

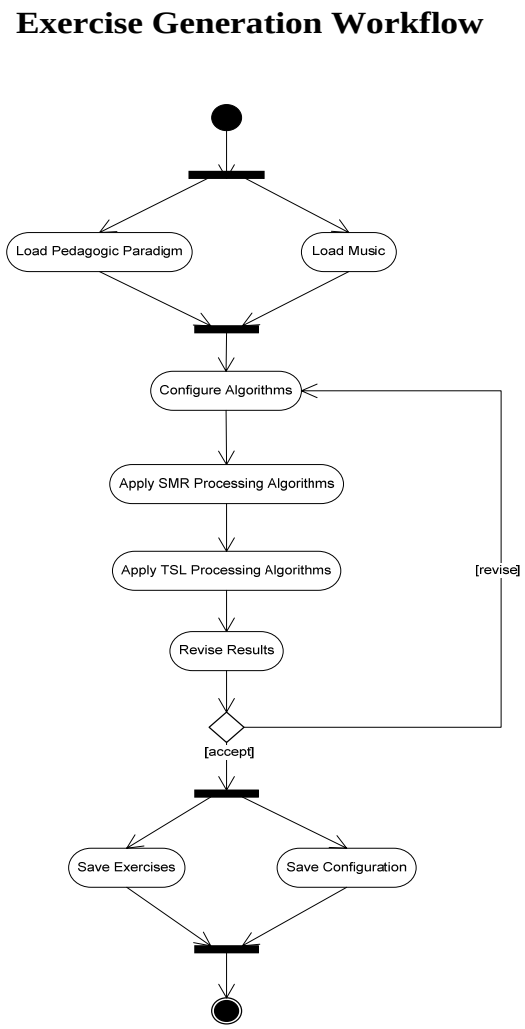
The music generated in SMR can be based on a provided piece of music, from which the music is excerpted. Annotations in SMR can be used to control the excerption and processing. Alternatively music can be synthesised according to generation rules. The processing uses SMR Generation Algorithms provided in the package of that name.

The exercise generator generates exercise descriptions, which can be interpreted by the Music Training Exercise Processor. The creation of the Exercise Formalisation is based on pedagogic paradigms, which describe the structure of the created exercises. The main function of the Exercise Generator is to combine a pedagogic paradigm, music material, and user-defined parameters to generate a variety of exercises.

Several design decisions have still to be made, since this is a novel concept and some of the prerequisite work and resources are not yet available. This includes the question whether the Generator Configuration will be stored in the Pedagogical Paradigm or separately, and the question, whether the Exercise Generator Tool should be integrated into the Exercise Authoring Tool. Also many details depend on the SMR and TSL formats that are currently being developed.

15.2 Exercise Generation Workflow

The basic Exercise Generation workflow consists of three main stages as shown in following activity diagram.



Firstly, for preparation the pedagogic paradigm and the music material are loaded and configuration of the algorithms is set up. Secondly, the algorithms are applied and the exercises and music material are created. Thirdly, possibly after revision, the result and the configuration are saved.

This workflow represents – apart from the revision loop – a batch processing approach. This requires the User (normally a Teacher) to have good knowledge of the algorithms involved. To allow a more explorative approach, especially for not technically inclined users, more revision points and interactivity will be added at where appropriate. For example, showing the SMR processing results using the score display, and give the user an opportunity to modify the Annotations and/or the Configuration to match her/his intention before applying the TSL processing.

There are two types of Exercises: Symbolic Training Exercises and Practice Training Exercise. In Symbolic Training, the Exercises focus on theoretical concepts. The typical input will therefore be either multiple-choice or the input of a small number of individual notes, either with the mouse, a keyboard, or an acoustic instrument. In Practice Training, the input will consist mainly of recordings of student performances, both acoustical and video/3D sensors. These will be analysed by the tools developed in Audio Processing Tools and Score Following.

Although both types deal with different content, they are similar from the generation aspect. Knowledge Units are structurally different and do not need the generation features provided here, since they offer mainly existing media material with little interactivity.

15.3 TSL Generation Algorithms

The generation of exercises represented in TSL uses Pedagogic Paradigms, which function as templates for the exercises generated. The concrete extent of functionality that can be implemented will evolve during the project, as the TSL and pedagogic paradigms are developed. Nevertheless, some basic algorithms classes can already be identified and are listed in the following. For these classes general post-conditions are specified, that are valid for all algorithms of that class. In addition there may be special constraints for specific music material, Pedagogic Paradigms, or functionality, which need to be added as necessary in the course of development.

15.3.1 Content Insertion

The first and simplest case is inserting content into the template, e.g. music examples in SMR.
General Post-Conditions:

1. The result must be correct TSL.
2. The result must contain the specified content at the specified position.

15.3.2 Exercises with Music Variations

Using one exercise template to generate multiple exercises by creating variations on the music. This is mainly a multiple application of Content Filling after applying SMR Variation Algorithms.
General Post-Conditions:

1. The result must be correct TSL.
2. Variations must not be identical

15.3.3 Content Recombinations

The content of a lesson can be intensively trained by recombining its elements, e.g. in a multiple choice exercise.
General Post-Conditions:

1. The result must be correct TSL.
2. The combination of elements must not contain identical elements
3. Successive combinations must not be identical

15.3.4 Exercise Variations

Content can be trained intensively by using different types of exercises on it. This may include applying different SMR Algorithms as appropriate to the different exercise types.
General Post-Conditions:

1. The result must be correct TSL.
2. Successive exercises must not be identical.

15.4 SMR Generation Algorithms

SMR generation algorithms are used for the generation of music material for exercises. We can only implement a small part of the vast space of possible algorithms, of which many may be interesting in particular pedagogic situations. To overcome this problem we implement basic algorithms and offer ways of extending and combining these that allow the user to recombine algorithms. The algorithm set of will further evolve during the course of the project as the progress in pedagogic concepts entails the development of new algorithms.

The SMR Generation Algorithms can be divided into two groups: Synthesis and Variation. SMR Synthesis Algorithms for synthesising SMR (as opposed to creating variations) provide music material without music input.

Generation of basic musical elements

The generation of basic musical elements like scales and chords, arpeggios will be realised by pattern descriptions, that specify the basic form of the element (e.g. a major triad chord as c-e-g or, if useful, in a more abstract form, such as: minor third, major third). As an extension the root note, numbers of repetitions, and note values will be variable and controlled by parameters.

Generation of constrained musical elements

Generation of melodic and rhythmic material corresponding to musical constraints like key, time signature, rhythmic patterns, and harmonic sequence. This algorithm includes mainly mapping of pitch and rhythmic values to fit a constraint pattern.

15.4.1 SMR Variation

These algorithms provide essential musical variations, such as the following:

Chromatic transposition

This chromatic transposition *ct(n,x)* changes the pitch of every processed note *n* by an interval *x* that is provided as a parameter, such that $cp(n) = cp(ct(n,x)) + x$, where *cp*(*n*) is the chromatic pitch of *n.* The transposition must produce enharmonically correct pitches, as these are necessary for any exercises involving music notation. Enharmonic correctness in this case means that the interval *x* is consistently treated as the same diatonic interval and the produced pitches are consistent with it. E.g. a Tritone (6 semitones) can either be treated as an Augmented Fourth or as an Diminished Fifth, leading to completely different diatonic notes.

Diatonic transposition

Diatonic transposition *dt(n,x,s)* changes the pitch of each processed note by the same diatonic interval, i.e. they have the same distance on a scale *s*, which is usually a subset of the 12 semitones of every octave, called the degrees. The general condition is $dg(n) = dg(dt(n,x,s),s) + x$, where *dg*(*n*,*s*) returns the degree of note *n* with respect for scale *s*. In addition we require $acc(n,s) = acc(dt(n,x,s),s)$, indicating that the

accidental of *n* with respect to its scale degree in s is kept constant, and $\text{sgn}((dt(n,x,s)-n) = \text{sgn}(x)$ which states that all transpositions must be made in the same direction. However, $cp(n) = cp(dt(n,x,s)) + x$ does not hold in general, but where *s* is symmetric, i.e. where all intervals in s are of the same size.

Change in key signature

A change in key signature entails either a transposition (in this case the appropriate direction of the transposition has to be found) or it puts the existing notes into a new tonal context, which entails an adaptation of the accidentals.

Rhythmic transformation

A rhythmic transformation of can always be reduced to changing start times and durations of notes. The questions, how these changes are controlled, and which constraints must be satisfied, depend very much on the pedagogical application. Transformations that are commonly used are the change of position (within a composition or locally e.g. from the beat to a syncope) subdivision (binary – swing- ternary – dotted – double-dotted) and duration (also articulation).

Change in tempo

A change in tempo is a specific rhythmic transformation, where all start times and durations are multiplied by a constant factor. In addition, the tempo indications in the SMR must be adapted.

Application of rhythmic patterns to melodic material

This is a special case of rhythmic transformation, which takes the rhythmic properties of one set of notes and changes the start times and durations of another set accordingly. This may have implications for the structure of the music (i.e. rests or surrounding notes will have to be added or deleted to maintain metrical structure). The specifics of these adaptations depend on the pedagogical purpose.

Application of melodic material to rhythmic patterns

This is the inverse of the previous case, where the pitch values of a set of notes are taken as a model for another set of notes. Again adaptations may be useful, but specifics are not yet clear.

Excerption

The algorithms must comprise extraction of voices, segments and selections based on rules. This is performed by copying the notes marked up to a new SMR unit, which can then be further processed. For efficient and effective use, the annotation of relevant sets of notes has to comply with the conventions used in the Generation algorithms. Otherwise, manual identification of the note sets will be necessary.

Application of style parameters

With more complex algorithms, combined of the basic building blocks above, it will be possible to take into account style, when generating or varying music. This can mainly be done through the choice of appropriate parameters and patterns (e.g. create a ‘swing’ feeling by dividing crotchets in 2:1 ratio), or the selection of specific material (e.g. marked by specific annotations). This will give a high-level interface, especially useful to users who lack the time or skills to define their own algorithms.

15.5 Generator Configuration

The Generator Configuration stores all the information on the used material and the application of the algorithms. It contains the information about which algorithms are to be used, the parameters for the individual algorithms, and rules controlling their application and combination.

Material

The definition, which parts of given music are to be used, is made by reference to annotations in the SMR. There can be different material for different algorithms, which may be combined (e.g. when applying a pattern).

Parameters

The parameters of algorithms, e.g. the interval of a transposition, need to be defined for every algorithm.

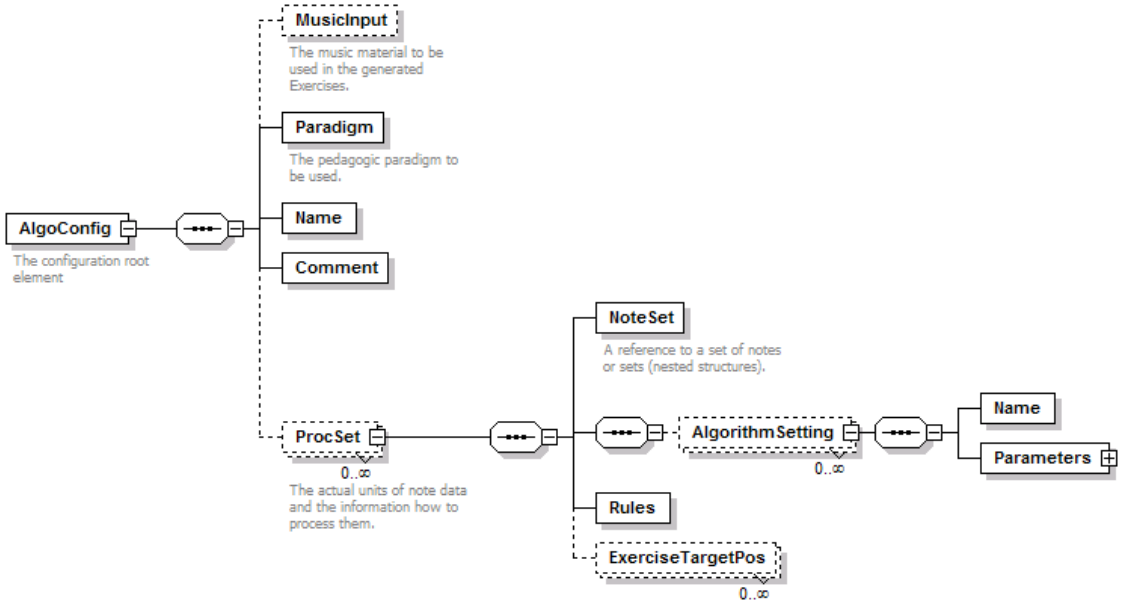
Rules

Generation Rules allow to specify the scope of the algorithm application, e.g. 'generate the c major scale and transpose it to all 12 semitones'. Therefore it allows simple scripting with loops over ranges of numbers and sets of music material, as well as conditions that specify. This feature allows programmers and users to create composed algorithms for later use or reuse, as well as is gives great flexibility for matching individual needs.

Further research needs to be done, to find out, whether it will be possible to use an existing scripting language like JavaScript, or whether a specialized solution is more suitable.

15.6 Format for Generation Configuration

The Generation Configuration will be stored as an XML file according to an XML Schema definition. The structure of the Schema is shown in the following diagram. Individual elements in the Schema depend on the details of the SMR and TSL.



15.7 Music Exercise Generator User Interface

The user interface of the Exercise Generator allows the User to load the data, define the configuration, run and control the application of the algorithms. The essential parts of the interface will be menus and forms that allow creation of a configuration and running it. In addition the Score Editor Module will be used to allow annotation of music and viewing created SMR, and the created Exercises in TSL will be displayed using the Exercise Authoring Tool (see chapter 7).
The Menus will provide the following basic functionality:

- Main Menu

    - Load Paradigm
    - Load SMR
    - Load Configuration
    - Save SMR
    - Save TSL
    - Save Configuration
- SMR Algorithm Menu
    - Add Algorithm
    - Remove Algorithm
- TSL Algorithm Menu
    - Add Algorithm
    - Remove Algorithm
- Processing
    - Start SMR Processing
    - Start TSL Processing
- Help
    - About
    - Help Index

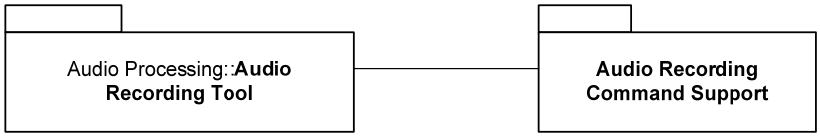
The loading and saving items will be implemented using Standard File dialogs, as far as they load from files. Algorithm Parameters and Rules will be editable in GUI forms in the main window, where also SMR will be displayed using the I-MAESTRO SMR viewer and the editor for editing Annotations. Whether the viewing and playback of generated exercises should better be handled in the main window or in a separate window, needs to be determined during early tests.

15.8 Music Exercise Generator Tool Profile

Module/Tool Profile	
Exercise Generator Tool	
Responsible Name	Tillman Weyde
Responsible Partner	LCU
Status (proposed/approved)	Proposed
Implemented/not implemented	Not implemented
Status of the implementation	0%
Executable or Library/module (Support)	Module or Executable (to be decided in due course)
Single Thread or Multithread	Single threaded
Language of Development	Java or C++
Platforms supported	Windows and Macintosh
Reference to the location of the source code demonstrator	
Reference to the location of the demonstrator executable tool for internal download	
Reference to the location of the demonstrator executable tool for public download	
Address for accessing to WebServices if any, add accession information (user and	

Passwd) if any		
Test cases (present/absent)	Present	
Test cases location	See DE 2.1.1d section 8.5	
Major Problems not solved		
Major pending requirements		
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
Exercise Authoring Tool		
Formats Used	Shared with	format name or reference to a section
MPEG SMR	All I-MAESTRO	MPEG-4 Symbolic Music Representation
TSL	Exercise Authoring, Player	To be developed
Generator Configuration		See above
Protocol Used	Shared with	Protocol name or reference to a section
Used Database name		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not

16 Specification of Audio Recording Tool

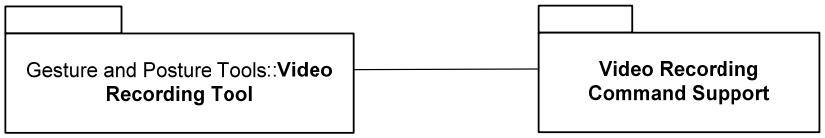


The Audio Recording Tool available in Max/MSP are *movie* and *imove* objects. They use QuickTime, so this program has to be installed in the system. It is possible to open an audio file (all QuickTime format are supported such as mp3, wav, aiff), play, stop, pause, resume ,go to a specific point, etc.
For more details see the Max/MSP reference manual.

There are many objects specifically designed to manipulate MIDI format. It is possible to prepare message in midi format, send in/out raw midi data, interpret, play and record midi data. It is also possible manage real time midi messages.

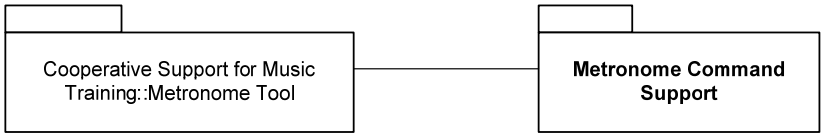
Principal objects that provide these capabilities are *midin*, *midiout*, *midiformat*, *midipars*, *midiflush*, *seq* (for midi recroding) and *rtin* (for real time message).

17 Specification of Video Recording Tool



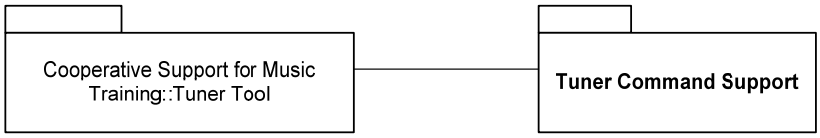
The Video Recording Tool available in Max/MSP are *movie* and *imove* objects. They use QuickTime, so this program has to be installed in the system. It is possible to open a video file (all QuickTime format are supported such as MPEG, MOV), play, stop, pause, resume , go to a specific point, etc. For more details see the Max/MSP reference manual).

18 Specification of Metronome Tool



The Metronome Tool allows to keep Tempo during a performance. There are two types of objects inside Max/MSP that can be used as metronome: *metro* and *qmetro* (based on jitter). It is possible to start and stop *metro* and it sends a bang message at regular intervals. This message can be used to turn on a led or to produce a sound useful to create a metronome. (For more details see the Max/MSP reference manual).

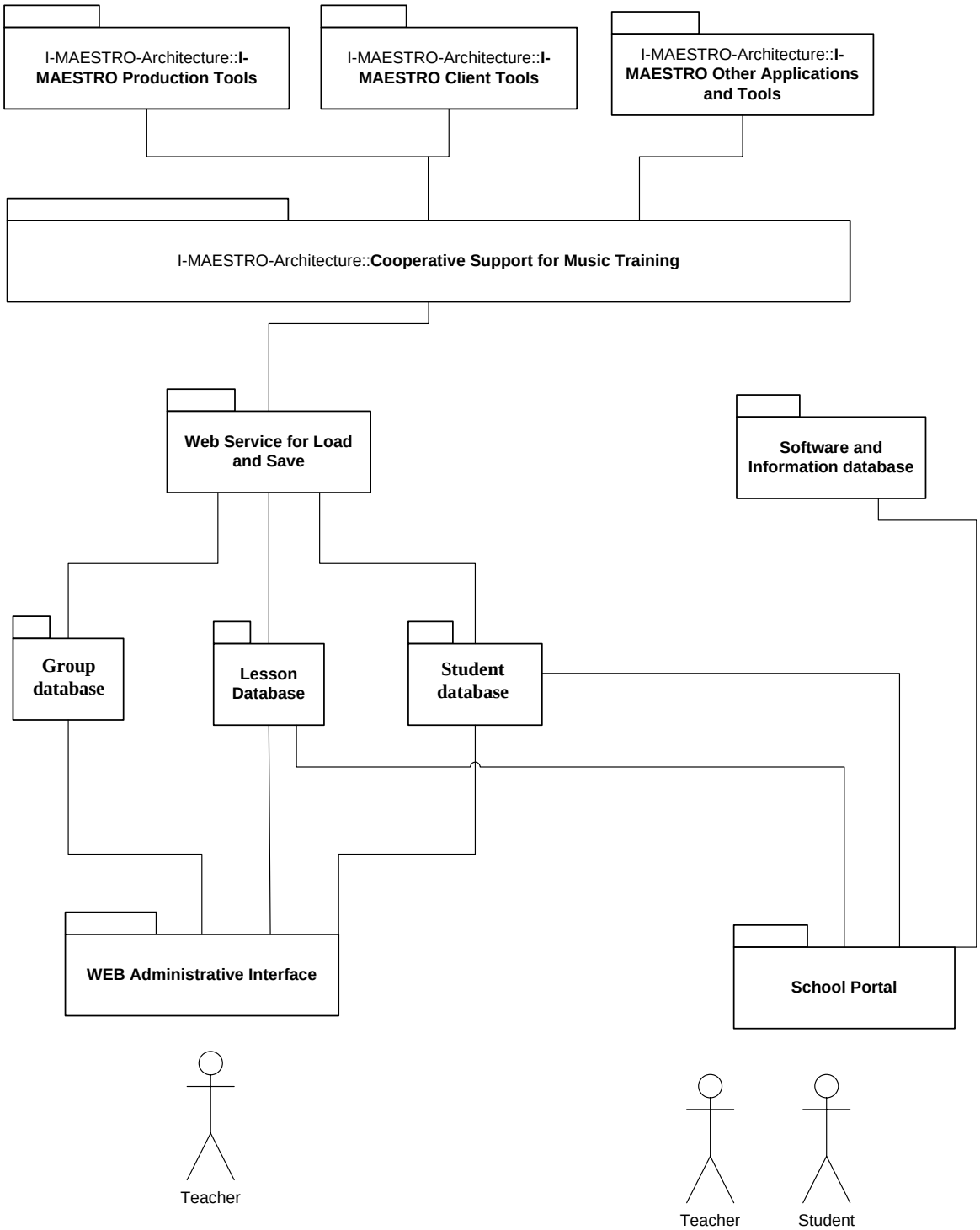
19 Specification of Tuner Tool



The Tuner Tool allows to tune instruments before a performance. It is possible to make a simple tuner using midi objects (e.g. using *mtof* object) to generate note with different frequencies. For more details see the Max/MSP reference manual)

20 Specification of I-MAESTRO School Server and Portal for the school

I-MAESTRO School Server



I-MAESTRO School Server contains all available I-MAESTRO contents used by tools and all information about registered users (Students and Teacher). Databases by the School Server contain Student profiles, Lesson archived, and information about workgroup used in cooperative works.
Databases have an internal access for:

- I-MAESTRO Production Tools, I-MAESTRO Client Tools and I-MAESTRO Other Applications and Tools through Web Services used by Cooperative Support for Music Training to load and save Lessons and Exercises
- Teachers who use Web Administrative Interface

and an external access through School Portal accessible from internet used by registered Teachers and Students.
Internal access gives full access to databases, while external one gives access only to Lesson database, Workgroup database and Software and Information database.

20.1 Lesson Database

Module/Tool Profile		
Lesson Database		
Responsible Name		
Responsible Partner	EXITECH	
Status (proposed/approved)	proposed	
Implemented/not implemented	Not implemented	
Status of the implementation		
Executable or Library/module (Support)		
Single Thread or Multithread	Multithread	
Language of Development		
Platforms supported		
Reference to the location of the source code demonstrator		
Reference to the location of the demonstrator executable tool for internal download		
Reference to the location of the demonstrator executable tool for public download		
Address for accessing to WebServices if any, add accession information (user and Passwd) if any		
Test cases (present/absent)		
Test cases location	http://	
Major Problems not solved	-- --	
Major pending requirements	-- --	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
Formats Used	Shared with	format name or reference to a

		section
Protocol Used	Shared with	Protocol name or reference to a section
Used Database name		
IMDB		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not

20.1.1 General Description of the Module

The I-MAESTRO school server store information about the Lessons. The Lessons data model seems to need a quite large number of tables that can be divided in two main categories: one for teacher data and the other for the student supplied data.

The tables involved in the teacher data should be:

- Lesson
- Lesson categories
- Lesson access rights
- Lesson difficulty levels

The tables involved in the student created data should refer to:

- Lesson attempts, states and results

The loader and saver web-service will mainly interact with the run time (student supplied) side. The administrative side (teacher provided data) will interact with the administrative interface.

20.1.2 User interface description of the tool

This module has no user interface since it is a database

20.1.3 Table description for database Lesson Database

Course Table

	Campo	Tipo	Collation	Attributi	Null	Predefinito	Extra
☐	id	int(10)		UNSIGNED	No		auto_increment
☐	category	int(10)		UNSIGNED	No	0	
☐	sortorder	int(10)		UNSIGNED	No	0	
☐	password	varchar(50)	latin1_swedish_ci		No		
☐	fullname	varchar(254)	latin1_swedish_ci		No		
☐	shortname	varchar(15)	latin1_swedish_ci		No		
☐	idnumber	varchar(100)	latin1_swedish_ci		No		
☐	summary	text	latin1_swedish_ci		No		
☐	format	varchar(10)	latin1_swedish_ci		No	topics	
☐	showgrades	smallint(2)		UNSIGNED	No	1	
☐	modinfo	longtext	latin1_swedish_ci		No		
☐	newsitems	smallint(5)		UNSIGNED	No	1	
☐	teacher	varchar(100)	latin1_swedish_ci		No	Teacher	
☐	teachers	varchar(100)	latin1_swedish_ci		No	Teachers	
☐	student	varchar(100)	latin1_swedish_ci		No	Student	
☐	students	varchar(100)	latin1_swedish_ci		No	Students	
☐	guest	tinyint(2)		UNSIGNED	No	0	
☐	startdate	int(10)		UNSIGNED	No	0	
☐	enrolperiod	int(10)		UNSIGNED	No	0	
☐	numsections	smallint(5)		UNSIGNED	No	1	
☐	marker	int(10)		UNSIGNED	No	0	
☐	maxbytes	int(10)		UNSIGNED	No	0	

☐	showreports	int(4)		UNSIGNED	No	0	
☐	visible	int(1)		UNSIGNED	No	1	
☐	hiddensections	int(2)		UNSIGNED	No	0	
☐	groupmode	int(4)		UNSIGNED	No	0	
☐	groupmodeforce	int(4)		UNSIGNED	No	0	
☐	lang	varchar(10)	latin1_swedish_ci		No		
☐	theme	varchar(50)	latin1_swedish_ci		No		
☐	cost	varchar(10)	latin1_swedish_ci		No		
☐	timecreated	int(10)		UNSIGNED	No	0	
☐	timemodified	int(10)		UNSIGNED	No	0	
☐	metacourse	int(1)		UNSIGNED	No	0	

New fields have to be added in order to fulfil I-MAESTRO requirements:

Filed Name	Type	Description
CourseLevelID	int	

Lesson Category Table (mdl_course_category)

	Campo	Tipo	Collation	Attributi	Null	Predefinito	Extra
☐	id	int(10)		UNSIGNED	No		auto_increment
☐	name	varchar(255)	latin1_swedish_ci		No		
☐	description	text	latin1_swedish_ci		No		
☐	parent	int(10)		UNSIGNED	No	0	
☐	sortorder	int(10)		UNSIGNED	No	0	
☐	coursecount	int(10)		UNSIGNED	No	0	
☐	visible	tinyint(1)			No	1	
☐	timemodified	int(10)		UNSIGNED	No	0	

Lesson difficulty levels (mdl_course_level)

Filed Name	Type	Description
LessonLevelID	Primary key	
Level code	varchar	Coded level string
Description	varchar	
Visible	Boolean	
Timemodified	Date	

Lesson access rights (mdl_course_accesses)

Filed Name	Type	Description
LessonAccesRightID	Primary key	
ShortCode	varchar	Coded level string

Description	varchar	
Visible	Boolean	

Lesson modules: (mdl_course_modules) – Stores info about single modules in a course section, which may consists of quizzes, internal resources (HTML), external resources (link to other sites or files), etc…

	Campo	Tipo	Collation	Attributi	Null	Predefinito	Extra
☐	id	int(10)		UNSIGNED	No		auto_increment
☐	course	int(10)		UNSIGNED	No	0	
☐	module	int(10)		UNSIGNED	No	0	
☐	instance	int(10)		UNSIGNED	No	0	
☐	section	int(10)		UNSIGNED	No	0	
☐	added	int(10)		UNSIGNED	No	0	
☐	score	tinyint(4)			No	0	
☐	indent	int(5)		UNSIGNED	No	0	
☐	visible	tinyint(1)			No	1	
☐	groupmode	tinyint(4)			No	0	

Lesson resources: (mdl_resources) - Stores info about active course external resources (link or files) in case that the module type (module field in mdl_course_modules).

	Campo	Tipo	Collation	Attributi	Null	Predefinito	Extra
☐	id	int(10)		UNSIGNED	No		auto_increment
☐	course	int(10)		UNSIGNED	No	0	
☐	name	varchar(255)	latin1_swedish_ci		No		
☐	type	varchar(30)	latin1_swedish_ci		No		
☐	reference	varchar(255)	latin1_swedish_ci		Si	*NULL*	
☐	summary	text	latin1_swedish_ci		No		
☐	alltext	text	latin1_swedish_ci		No		
☐	popup	text	latin1_swedish_ci		No		
☐	options	varchar(255)	latin1_swedish_ci		No		
☐	timemodified	int(10)		UNSIGNED	No	0	

The student's attempts and result are stored in a dedicated table: mdl_scorm_scoes_track structured as follows:

	Campo	Tipo	Collation	Attributi	Null	Predefinito	Extra
☐	id	int(10)		UNSIGNED	No		auto_increment
☐	userid	int(10)		UNSIGNED	No	0	
☐	scormid	int(10)			No	0	
☐	scoid	int(10)		UNSIGNED	No	0	
☐	element	varchar(255)	latin1_swedish_ci		No		
☐	value	longtext	latin1_swedish_ci		No		
☐	timemodified	int(10)		UNSIGNED	No	0	

20.2 Student and Teachers Database

Module/Tool Profile		
Student Database		
Responsible Name		
Responsible Partner	EXITECH	
Status (proposed/approved)	proposed	
Implemented/not implemented	Not implemented	
Status of the implementation		
Executable or Library/module (Support)		
Single Thread or Multithread	Multithread	
Language of Development		
Platforms supported		
Reference to the location of the source code demonstrator		
Reference to the location of the demonstrator executable tool for internal download		
Reference to the location of the demonstrator executable tool for public download		
Address for accessing to WebServices if any, add accession information (user and Passwd) if any		
Test cases (present/absent)		
Test cases location	http://	
Major Problems not solved	-- --	
Major pending requirements	-- --	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
Formats Used	Shared with	format name or reference to a section

Protocol Used	Shared with	Protocol name or reference to a section
Used Database name		
IMDB		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not

20.2.1 General Description of the Module

The I-MAESTRO school server store information about Students and Teachers. The data model requires a quite large number of tables that will contains all the Student and Teacher profile.

The databases tables should be:

- Students
- Teachers

The loader and saver web-service will mainly interact with the run time (student supplied) side. The administrative side (teacher provided data) will interact with the administrative interface.

Student Table (mdl_user) - Holds user information for each person

FIELD NAME	TYPE	Notes
id	int(10)	
confirmed	boolean	Confirmed enrolment via e-mail
policyagreed	boolean	Student accept the policy
deleted	boolean	Deleted user (history archived and visible)
username	varchar(100)	
password	varchar(32)	
idnumber	varchar(64)	
firstname	varchar(20)	
lastname	varchar(20)	
email	varchar(100)	
emailstop		
icq	varchar(15)	
skype	varchar(50)	
yahoo	varchar(50)	
aim	varchar(50)	
msn	varchar(50)	
phone1	varchar(20)	
phone2	varchar(20)	
institution	varchar(40)	
department	varchar(30)	
address	varchar(70)	
city	varchar(20)	
country	char(2)	
lang	varchar(10)	

theme	varchar(50)
timezone	varchar(100)
firstaccess	int(10)
lastaccess	int(10)
lastlogin	int(10)
currentlogin	int(10)
lastIP	varchar(15)
secret	varchar(15)
picture	
url	varchar(255)
description	text
mailformat	
maildigest	
maildisplay	
htmleditor	
autosubscribe	
trackforums	
timemodified	int(10)

Administrators table (mdl_user_admins) - One record per administrative user

id	Primary key
userid	User ID

Teachers table (mdl_ user_coursecreators) - One record per Teacher

id	Primary key
userid	User ID

Student activity table (mdl_user_students) - Holds student information

id	Primary key
userid	User ID
LessonID	Course student enrolled in
timestart	
timeend	
time	

mdl_user_teachers - One record per teacher per course

id	Primary key
userid	User ID
LessonID	Course student enrolled in
authority	
role	
editall	
timemodified	Most recent time teacher record modified

20.3 Workgroup Database

Module/Tool Profile		
Workgroup Database		
Responsible Name		
Responsible Partner	EXITECH	
Status (proposed/approved)	proposed	
Implemented/not implemented	Not implemented	
Status of the implementation		
Executable or Library/module (Support)		
Single Thread or Multithread	Multithread	
Language of Development		
Platforms supported		
Reference to the location of the source code demonstrator		
Reference to the location of the demonstrator executable tool for internal download		
Reference to the location of the demonstrator executable tool for public download		
Address for accessing to WebServices if any, add accession information (user and Passwd) if any		
Test cases (present/absent)		
Test cases location	http://	
Major Problems not solved	-- --	
Major pending requirements	-- --	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
Formats Used	Shared with	format name or reference to a section
Protocol Used	Shared with	Protocol name or reference to a section
Used Database name		
IMDB		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.

Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not

Groups Table_(mdl_groups)

GroupID	Primary key	
Name	varchar	
Description	Varchar	
Password	Varchar	
Lang	Varchar	
Theme	Varchar	
picture	URL	
hidepicture	Boolean	
Timecreated	Date	
timemodified	date	

Groups Lesson relation Table_(IMDB_lesson_groups)

ID	Primary key	
GroupID	varchar	
LessonID	Varchar	

Groups Users relation Table

ID	Primary key	
GroupID	varchar	
UserID	Varchar	
Timeadded	date	

20.4 Web Service for Load and Save

Module/Tool Profile	
Web Service for Load and Save	
Responsible Name	
Responsible Partner	EXITECH
Status (proposed/approved)	proposed
Implemented/not implemented	Not implemented
Status of the implementation	
Executable or Library/module (Support)	
Single Thread or Multithread	Multithread
Language of Development	JAVA
Platforms supported	Windows
Reference to the location of the source code demonstrator	
Reference to the location of the demonstrator executable tool for internal download	
Reference to the location of the demonstrator executable tool for	

public download		
Address for accessing to WebServices if any, add accession information (user and Passwd) if any		
Test cases (present/absent)		
Test cases location	http://	
Major Problems not solved	-- --	
Major pending requirements	-- --	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
Formats Used	Shared with	format name or reference to a section
Protocol Used	Shared with	Protocol name or reference to a section
Used Database name		
IMDB		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not

20.4.1 General Description of the Module

The I-MAESTRO school server Loader/Saver web-service is a tool that is capable of getting an I-MAESTRO object and putting it in the database, that is the saving function; and it is also capable, given an Object ID to return the I-MAESTRO object in the client compliant format, that is the loading function.

The load and save function are always considered from the point of view of the user, so that Load means “load from I-MAESTRO” and save means import inside I-MAESTRO database.

The I-MAESTRO Loader/Saver interacts with Cooperative Support for Music Training. This module offers the services for saving in the database objects and for loading objects from the database. This module can be split in the Loader and Saver in order to have a better understanding of the different behaviour of the two services. Both services of this module are implemented as web-services and therefore in the following specification, the WSDL of the proposed service together with samples of the SOAP messages will be reported. The services can operate in synchronous and asynchronous mode. In asynchronous mode a Listener approach will be implemented, in the sense that the user, in order to be allowed to use asynchronous operation must offer a web-service with a known interface to receive the result of the operation.
The saver module will automatically set the version of the object inside the object as soon as the object is saved increasing the number currently present in the database.

All services will be implemented in JAVA.

Saver

Saver/Indexer module offers basically the database commit service.

Loader

The loader module offers basically the checkout service.
The following table summarizes the methods of the Loader web-service a short description of the functionalities.

20.5 Web Administrative Interface

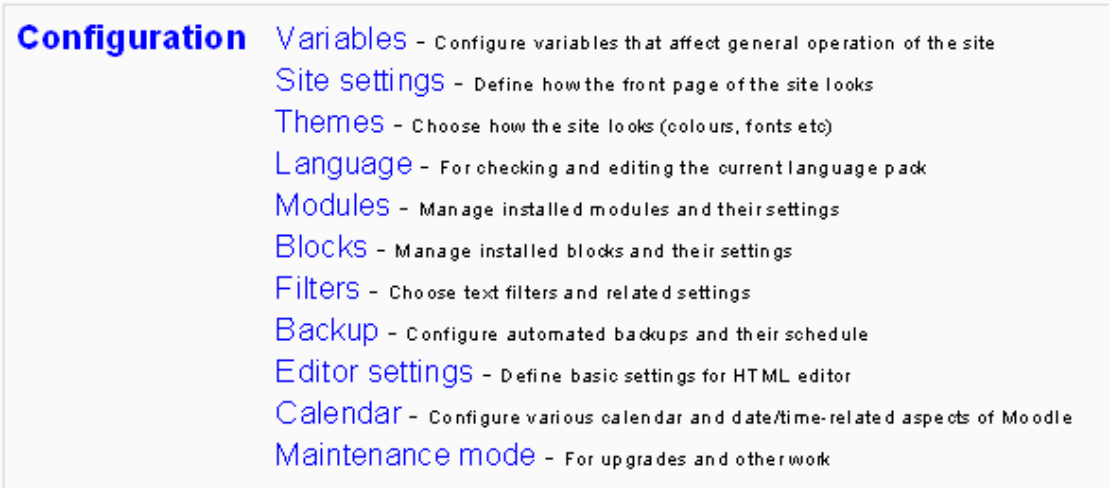
Module/Tool Profile		
Web Administrative Interface		
Responsible Name	Marius Spinu	
Responsible Partner	EXITECH	
Status (proposed/approved)	proposed	
Implemented/not implemented		
Status of the implementation		
Executable or Library/module (Support)		
Single Thread or Multithread		
Language of Development	PHP, JAVASCRIPT, JAVA	
Platforms supported	LINUX, WINDOWS	
Reference to the location of the source code demonstrator		
Reference to the location of the demonstrator executable tool for internal download		
Reference to the location of the demonstrator executable tool for public download		
Address for accessing to WebServices if any, add accession information (user and Passwd) if any		
Test cases (present/absent)		
Test cases location		
Major Problems not solved		
Major pending requirements		
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
Formats Used	Shared with	format name or reference to a section
Protocol Used	Shared with	Protocol name or reference to a section

Used Database name		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
MOODLE		
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
	MOODLE	GPL

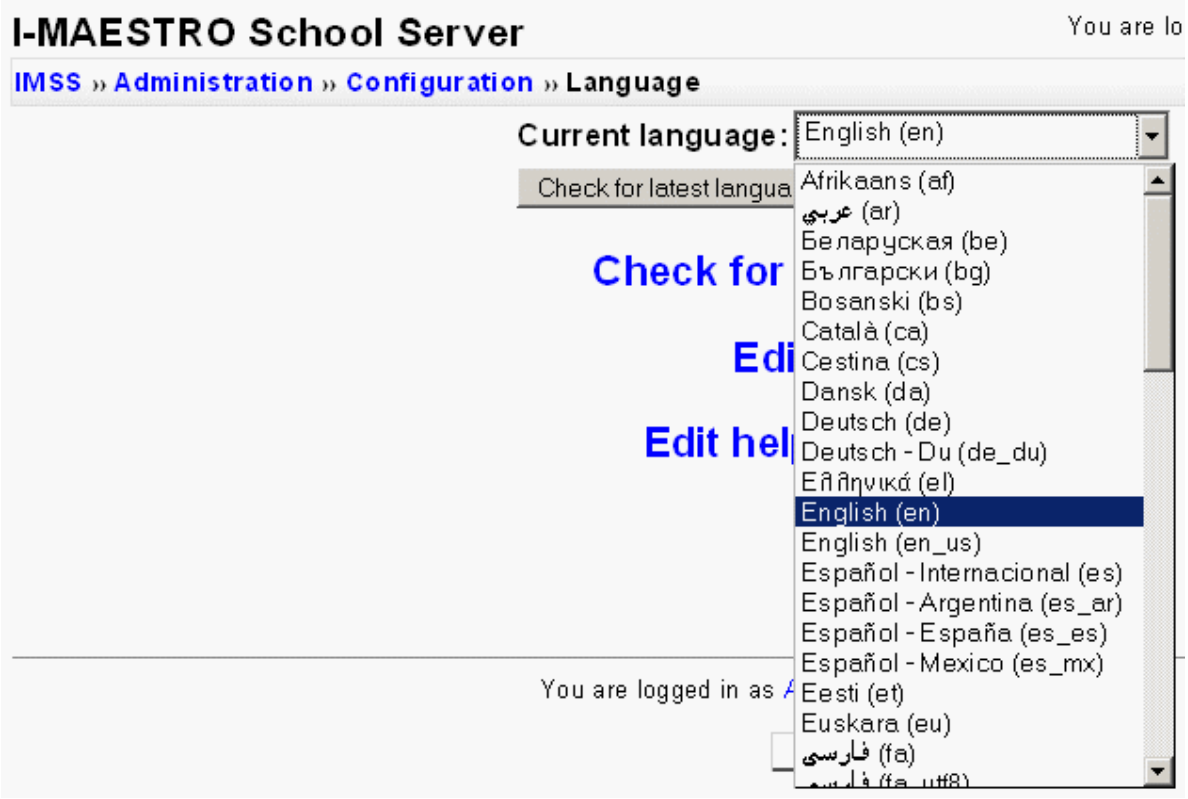
The School server administrative interface will allow to manage the school server. A Open Source Tool as MOODLE can be used in order to satisfy the requirements.

In MOODLE the administration functionalities are organised in more categories:

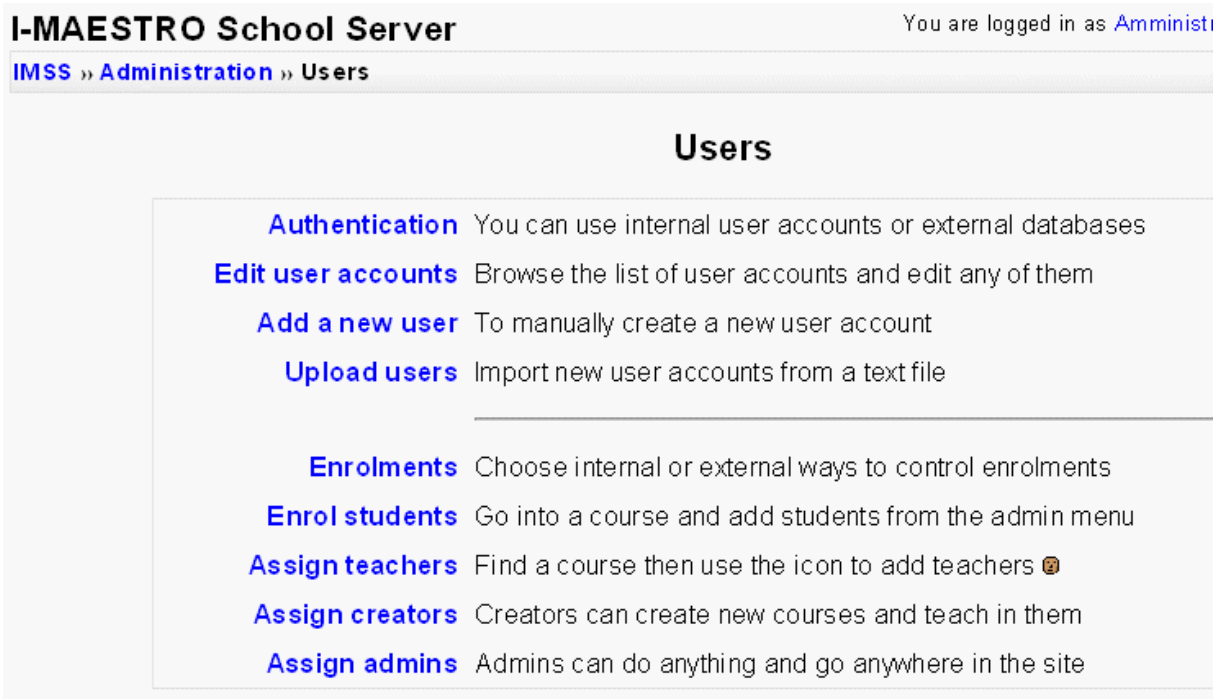
20.5.1 Configuration tool:



The variables item allows to modify Interface, Security, Mail, Regional, Permissions and other miscellaneous parameters. The language allows to quickly change the portal language:



A special section is dedicated to user management

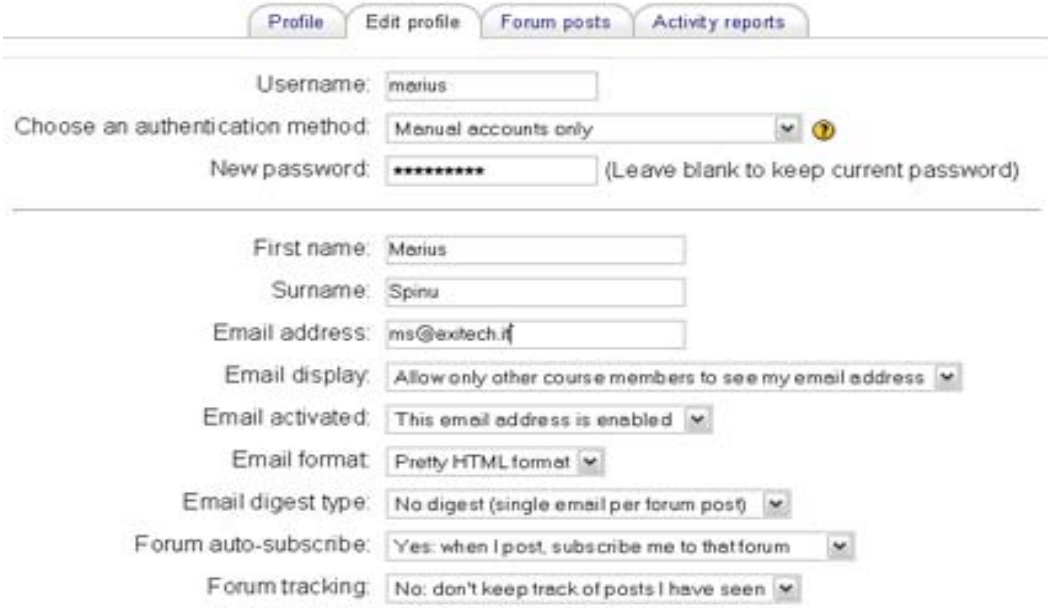


Some requested fields are already active in MOODLE and some new functionalities have to be implemented in order to fulfil with the user requirements.

20.5.2 Student profile management

STUDENT PROFILES	
Functionality	**Implemented in MOODLE?**
Create new profile	Yes
Save profile	Yes
Load (Open)	Yes
Editing profile	Yes
Delete	Yes
Profile comparison	No
Profile upload/download	No

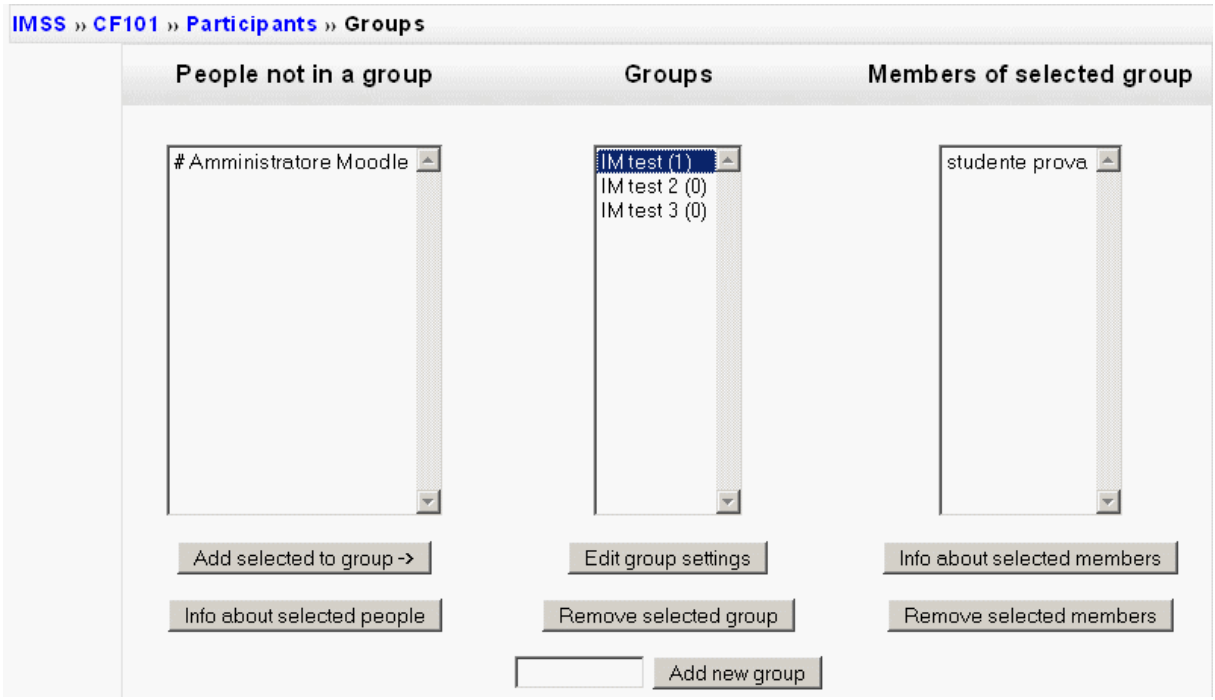
The MOODLE user interface related to the Student profile allows to create a new Student profile or to open/edit/update/delete an existing one.



20.5.3 Group profile management

In MOODLE the GROUP exist but can the groups are related only for one lessons/course. In I-MAESTRO the concept of group of students should be activated at a higher level.

As depicted in the next figure the MOODLE group management interface can be used but the functionalities have to be adapted to the project requirements.



Group profile	
Functionality	**Implemented in MOODLE?**
Create new group profile	Yes / to be changed
Save profile	Yes / to be changed
Load (Open)	Yes / to be changed
Editing profile	Yes / to be changed
Delete	Yes / to be changed
Profile comparison	No
Profile upload/download	No

20.5.4 Work profile management

The work profile concept is not implemented in MOODLE. It is quite similar to the group profile and should have the same graphic interface.

Work profile	
Functionality	**Implemented in MOODLE?**
Create new group profile	Yes / to be changed
Save profile	Yes / to be changed
Load (Open)	Yes / to be changed
Editing profile	Yes / to be changed
Delete	Yes / to be changed
Profile comparison	No
Profile upload/download	No

20.6 School Portal

Module/Tool Profile	
School Portal	
Responsible Name	Marius Spinu
Responsible Partner	EXITECH
Status (proposed/approved)	proposed
Implemented/not implemented	Not implemented
Status of the implementation	
Executable or Library/module (Support)	Web application
Single Thread or Multithread	Multithread
Language of Development	PHP
Platforms supported	Any OS
Reference to the location of the source code demonstrator	
Reference to the location of the demonstrator executable tool for internal download	
Reference to the location of the	

demonstrator executable tool for public download		
Address for accessing to WebServices if any, add accession information (user and Passwd) if any		
Test cases (present/absent)	present	
Test cases location		
Major Problems not solved	-- --	
Major pending requirements	-- --	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
Formats Used	Shared with	format name or reference to a section
Protocol Used	Shared with	Protocol name or reference to a section
Used Database name		
IMDB		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not

20.6.1 General Description of the Module

The School Portal is dedicated to the software download. It will contain all the I-MAESTRO tools, help files, configuration How-To, configuration tools, etc.

Large part of the portal should be publicly accessible but some software (according to the licenses) will be downloadable only for the registered users (Student or Teacher).

The portal will be structured in two levels

- Public level
- Controlled access level

In order to access the “Controlled access area” the I-MAESTRO Student or teacher account and password will be requested. The user activity should be monitored for statistical purposes.

20.7 Software and Information Database

Module/Tool Profile		
Software and Information Database		
Responsible Name	Marius Spinu	
Responsible Partner	EXITECH	
Status (proposed/approved)	proposed	
Implemented/not implemented	Not implemented	
Status of the implementation		
Executable or Library/module (Support)		
Single Thread or Multithread	Multithread	
Language of Development		
Platforms supported		
Reference to the location of the source code demonstrator		
Reference to the location of the demonstrator executable tool for internal download		
Reference to the location of the demonstrator executable tool for public download		
Address for accessing to WebServices if any, add accession information (user and Passwd) if any		
Test cases (present/absent)		
Test cases location		
Major Problems not solved	-- --	
Major pending requirements	-- --	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
Formats Used	Shared with	format name or reference to a section
Protocol Used	Shared with	Protocol name or reference to a section
Used Database name		
IMDB		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.

Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not

20.7.1 General Description of the Module

A database will be implemented in order to archive all the I_MAESTRO software tools and the information documents.

The database should be structured in some tables as:

- IMDB_downloadable_data
- IMDB_tools_access_rights
- IMDB_ downloadable resource table

20.7.2 Database tables description

IMDB_downloadable_data table

This table will contain the generic information about the downloadable data (software or information).

Field Name	Type	Notes
downloadableDataID	Int	Primary key
Name	varchar	
Description	varchar	
popup	varchar	
type	varchar(30)	

IMDB access right table

The access right table will associate a downloadable data with an userID for access right purposes.

Field Name	Type	Notes
downloadableDataID	Int	Primary key
userID	Int	

If a user is not present in the table he/she cannot download/view the files.

IMDB_ downloadable resource table

This table associate resources (mainly files) with the downloadable data

Field Name	Type	Notes
resourceID	Int	
name	varchar(255)	
type	varchar(30)	
reference varchar(255)		
summary text		
alltext text		

options varchar(255)		
timemodified	int(10)	

A table very similar with the mdl_resources table can be used in this case but in order to allow to one tool to be composed from more than one files an intermediary connection table can be considered:

Field Name	Type	Notes
downloadableDataID	Int	Primary key
resourceID	Int	

21 Annex A: Pre Existing Know – How

Table of the Pre-existing know how

The following table simplifies the assignment of the grants to the access at the pre-existing know-how/software and tools. According to this Consortium Agreement the rights to access at the know-how can be given only via written statements. This process can be very time-consuming thus the table reported is a way to simplify and accelerate the mechanism.

Contractor Provider	Pre existing know how tool, software, description	Providing status			Changed in WP	Month Availability	During Project		After Project conclusion		
		Source Code: Y language, N	Binary Code: Y/N	Operating Systems list			Licensing/free of Charge, FOC	Accessing Contractors	Licensing /free of Charge	Accessing Contractors	Licensing cost
DSI	WEDELMUSIC Editor: multimedia integration and distribution tools, image score, music notation, audio player, video player, animations, integration with authoring tools, watermarking scores, printing music, help support, transaction model, music notation conversions, o.	Source Code Available for the I-MAESTRO Framework , C++	Y	Windows, partially also Linux	NA	M1	Free of Charge (FOC) in I-MAESTRO Framework	DSI mainly, All for usage in the I-MAESTRO if needed	Free of Charge in I-MAESTRO Framework	All	Licensing based
DSI	Client certification and registration, **not completely of DSI.**	Source Code, C++, ASM	Y	Win, X86	NA	M1	FOC, I-MAESTRO Framework	DSI mainly, All	Licensing	All	
DSI	Feature extraction tools for audio files, algorithm for shrinking and stretching audio	Source Code, C++	Y	Win	NA	M1	FOC, I-MAESTRO Framework	All if necessary	Integrated in I-MAESTRO Framework	All	IMF share
DSI	Automatic formatting tools such as MILLA engine.	Source Code, C++	Y	Win	NA	M1	FOC, I-MAESTRO Framework	DSI mainly, All	Integrated in I-MAESTRO Framework	All	IMF share

UNIVLEEDS	2D video and 3D motion, acquisition and analysis modules	Source code, c++	Y	Windows	NA	M1	FOC, I-MAESTRO Framework	UNIVLEEDS mainly	Integrated in I-MAESTRO Framework	All	IMF share
UNIVLEEDS	Audio capture and analysis module	Source code, c++	y	Windows	NA	M1	FOC, I-MAESTRO Framework	UNIVLEEDS mainly	Integrated in I-MAESTRO Framework	All	IMF share
EXITECH, DSI	MPEG SMR player and light editor, as reported on the MPEG repository	Source Code Available for the I-MAESTRO Framework , C++	Y	Windows, partially also Linux	NA	M1	Free of Charge (FOC) in I-MAESTRO Framework	DSI mainly, All for usage in the I-MAESTRO if needed	Free of Charge in I-MAESTRO Framework	All	Related to the rules of MPEG ISO
DSI	MPEG21 authoring tool, player, object modelling tool, etc.	Source Code Available for the I-MAESTRO Framework , C++	Y	Windows, partially also Linux	NA	M1	Free of Charge (FOC) in I-MAESTRO Framework	DSI mainly, All for usage in the I-MAESTRO if needed	Licensing but free see on right	All	Licensing based on AXMEDIS model and mechanism
EXITECH	Portal support and framework for managing workgroups, mailing list, etc.	PHP, Source code	N	Linux	NA	M1	FOC	EXITECH, all as users	Licensing but free see on right	All	Free for 4 years after the project conclusion
IRCAM	SUIVI : software package for automatic accompaniment of a performer composed of a notation editor in JAVA, a learning mechanism based on HMM, algorithms for pitch and other signal features detection, synchronisation mechanism	Max/MSP source code (patches) including compiled object code (objects)	Y	Windows, MacOSX	NA	M1	Free of Charge (FOC) in I-MAESTRO Framework	All for usage in the I-MAESTRO if needed	Licensing	All	To be defined According to applications
IRCAM	Musique Lab : educational software in 3 modules (Annotation stand alone application, Composition Open Music based application, Live performing Max/MSP based stand	Source code accessible (access to patch edition) with OpenMusic	Y	Windows and MacOSX	NA	M1	Free of Charge (FOC) in I-MAESTRO Framework	All for usage in the I-MAESTRO if needed	Licensing Or distribution	All	To be defined According to applications

	alone application	(Free software) or with max/MSP (commercial software)									
IRCAM	Annotation tool		Y	SVG	NA	M1	Free of Charge (FOC) in I-MAESTRO Framework	All for usage in the I-MAESTRO if needed	Licensing Or distribution	All	To be defined According to applications
IRCAM	Audio signal extractors : Timbre, Tempo, Scales, Modes	MPEG7 XML code	Y	All	NA	M1	Free of Charge (FOC) in I-MAESTRO Framework	All for usage in the I-MAESTRO if needed	Licensing Or distribution	All	To be defined According to applications

Where:

- All: for accessing contractors, means also available for affiliate members, related subcontractors and partners associated with take-up actions.
- FOC: for licensing means Free Of Charge, that is on royalty free basis.
- I-MAESTRO Framework: means that the Source Code or the Binary code in discussion will be posted in the I-MAESTRO framework (it can be posted there for the project duration and/or after the project conclusion, see last columns).
- Integrated in I-MAESTRO Framework: means that the Pre-existing know-how and tools under discussion will be integrated into the I-MAESTRO model and tools and thus they will remain in the I-MAESTRO framework.
- IMF share, means I-MAESTRO Framework Share, the value and the licensing of this provided component will be estimated on the basis of the influence of the component itself into the I-MAESTRO Framework Share, which is the Cost of code/tools that will finish in the I-MAESTRO Framework.

Table of the Excluded Pre-existing know-how:

This table has to list the Pre existing know how and tools that are explicitly excluded from access and usage inside the I-MAESTRO project.

Contractor Provider	Pre existing know how tool, description
IRCAM	Pitch detection algorithm (code name : f(0))

22 Acronyms and Sources

22.1 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 organization 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 Organization is a standardization body that is recognized 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)
RTE	Run Time Environment
SCO	Sharable Content Object
SCORM ™	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.

22.2 Sources

- 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

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://exelearning.org/?q=about
- 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