Interactive Music Science Collaborative Activities

Team Teaching for STEAM Education

Deliverable 4.10

Final Version of Computational models for sound and music generation for virtual instruments

Date:	17/07/2018
Author(s):	Robert Piechaud (IRCAM), Quentin Lamérand (IRCAM)
Contributor(s):	
Quality Assuror(s):	Fotini Simistira (UNIFRI), Carlos Acosta (LEOPOLY)
Dissemination level:	PU
Work package	WP4 – Core enabling technologies modules of iMuSciCA
Version:	1.0
Keywords:	sound generation, virtual musical instrument
Description:	Final version of the computational models for sound and music generation for virtual instruments.

H2020-ICT-22-2016 Technologies for Learning and Skills
iMuSciCA (Interactive Music Science Collaborative Activities)
Project No. 731861
Project Runtime: January 2017 – June 2019

Executive Summary

In this deliverable we present the final version of the computational models for sound and music generation for virtual instruments. The virtual instrument models are based on physics and embodied in IRCAM's Modalys technology. As part of the iMuSciCA project, Modalys has been ported from C++ to the browser HTML5 context in order to function like any other workbench module. The result library is called modalys.js. First we present the Modalys iMuSciCA API, and then we will expose some standalone examples. Finally and briefly, we will talk about CPU performances.

The following urls will be come across in this document:
Root url:
https://s3amdev.ircam.fr/

Instruments test urls:
https://s3amdev.ircam.fr/pluckedstrings.html
https://s3amdev.ircam.fr/pluckedstrings_with_snail.html
https://s3amdev.ircam.fr/xylo.html
https://s3amdev.ircam.fr/plate.html
https://s3amdev.ircam.fr/simpledrum.html
https://s3amdev.ircam.fr/pluckedstrings2channels.html

Version Log			
Date	**Version No.**	**Author**	**Change**
27-06-2018	0.1	Robert Piechaud (IRCAM), Quentin Lamérand (IRCAM)	Initial content
28-06-2018	0.2	Robert Piechaud (IRCAM), Quentin Lamérand (IRCAM)	Ready for reviewing
12-07-2018	0.2	Fotini Simistira (UNIFRI), Carlos Acosta (LEOPOLY)	Review comments
13-07-2018	0.2	Fotini Simistira (UNIFRI)	Review comments
13-07-2018	0.3	Robert Piechaud (IRCAM), Quentin Lamérand (IRCAM)	Address reviewers comments
17-07-2018	1.0	Vassilis Katsouros (ATHENA)	Submission to the EU

Disclaimer

This publication has been produced with the assistance of the European Union. The content of this publication is the sole responsibility of iMuSciCA consortium and can in no way be taken to reflect the views of the European Union.
iMuSciCA is an H2020 project funded by the European Union.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

Abbreviation	Description
JS	Javascript
WASM	Web Assembly
asm.js	Strict subset of Javascript
API	Application Programming Interface
CPU	Central Processing Unit
WP	Work Package
PU	Public document
ATHENA	ATHENA RESEARCH AND INNOVATION CENTER IN INFORMATION COMMUNICATION & KNOWLEDGE TECHNOLOGIES
UCLL	UC LIMBURG
EA	ELLINOGERMANIKI AGOGI SCHOLI PANAGEA SAVVA AE
IRCAM	INSTITUT DE RECHERCHE ET DE COORDINATION ACOUSTIQUE MUSIQUE
LEOPOLY	3D FOR ALL SZAMITASTECHNIKAI FEJLESZTO KFT
CABRI	Cabrilog SAS
WIRIS	MATHS FOR MORE SL
UNIFRI	UNIVERSITE DE FRIBOURG

1. Introduction

The Modalys[1] virtual music instrument technology is embedded into **modalys.js**, a single-file javascript library. Modalys owns two types of sounding objects: i) primitives, such as strings, tubes, plates or membranes, and ii) 3D objects defined by their geometry expressed in a mesh in .obj format. In the second case, the mesh can either contains solids[2], or surface elements representing the neutral fiber[3] of the object.
modalys.js, through its message-based communication API, contains all that is required to instantiate, test and finally perform virtual musical instruments based on physical models.

2. Installation and technical requirements for modalys.js

2.1. Installation

modalys.js javascript library is included in the html code this way:

```
<script src="script/modalys.js"></script>
```

It defines a Modalys object that will need to be initialized in order to use it (cf. 3.1.2).

2.2 Requirements

modalys.js is built from the Modalys C++ code using emscripten[4] with heavy optimizations. It produced WebAssembly[5] code, and a javascript loader. This resulting javascript is wrapped then in an object that creates Web Audio nodes and an API to communicate with Modalys core functions.

3. Description and use of modalys.js

As a core sound library, modalys.js doesn't *display* anything (apart from log messages within the browser's developer console). As such, modalys.js works very much as a server, responding to client's request to test or perform virtual instruments.

[1] http://forumnet.ircam.fr/product/modalys-en/
[2] out of iMuSciCA's scope.
[3] or neutral axis: https://en.wikipedia.org/wiki/Neutral_axis
[4] https://en.wikipedia.org/wiki/Emscripten
[5] https://en.wikipedia.org/wiki/WebAssembly

3.1. modalys.js API

3.1.1. Communication context

iMuSciCA core modules communicate with each other through an asynchronous client-side messaging service: postal.js, which implements a channel/topic(.subtopic) paradigm.

- We use the modalys channel.
- Each module has a dedicated topic and subtopics.
- To 'talk' to the Modalys module, another module will publish on a topic from the modalys channel, named after the requested action (ex: play).
- Module can send notifications by publishing on the modalys > notification topic.
- A module will therefore subscribe to modalys > notification to receive Modalys' feedbacks.

3.1.2. Messages *to* Modalys

Upon initialization modalys.js subscribes to all topics of the "modalys" channel and dispatches data to methods of the Modalys object, whose names match the topics.
The Modalys object need to be initialized with an AudioContext and its output node connected to a destination. Here is a minimal sample code to setup Modalys.

```
Modalys.isready.then(function() {
  var audioContext = new (window.AudioContext || window.webkitAudioContext)();
  Modalys.init(audioContext);
  Modalys.output.connect(audioContext.destination);
});

var modalysChannel = postal.channel("modalys");
```

Here are the available topic of the "modalys" channel you can publish on :

- **try**: test the sound of an instrument being designed.
- **play**: real time performance of a virtual instrument.
- **updateParameter**: update some parameter (pluck position etc) in real time while playing.
- **pause**: put a virtual instrument in hold (play mode)
- **resume**: resume playing the instrument after hold.
- **stop**: stop a virtual instrument.
- **instrumentInfo** : ask materials list and presets for the specified "instrumentType"

3.1.3. Messages *from* Modalys

modalys.js publishes messages on the "notification" topic of the "modalys" channel.

3.1.4. Trying a virtual music instrument at design stage

It is a 3D Instrument Design ↔ modalys.js communication situation.
Below are the types of messages that should be sent upon instrument design stage, when testing how an instrument sounds.

3.1.4.1 Primitive-based instruments

Example for a 2-string instrument:

```
modalysChannel.publish("try",
  {
    "type": "pluckedString",
    "name": "MyBiChord",
    "content": [
      {
        "type": "chord",
        "name": "chordC",
        "content": [
          {
            "type": "number",
            "name": "length",
            "content": 1
          },
          {
            "type": "number",
            "name": "radius",
            "content": 0.0005
          },
          {
            "type": "number",
            "name": "tension",
            "content": 432
          },
          {
            "type": "string",
            "name": "material",
            "content": "steel"
          }
        ]
      },
      {
        "type": "chord",
        "name": "chordEb",
        "content": [
          {
            "type": "number",
            "name": "length",
            "content": 0.915
          },
          {
            "type": "number",
            "name": "radius",
            "content": 0.0005
          },
          {
```

```
            "type": "number",
            "name": "tension",
            "content": 510
          },
          {
            "type": "string",
            "name": "material",
            "content": "steel"
          }
        ]
      }
    ]
  }
);
```

In return, modalys should send these messages (on the “notification” topic) upon reception:

```
{
  "sessionId": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
  "status": "preparing"
}
```

```
{
  "sessionId": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
  "status": "processing"
}
```

```
{
  "sessionId": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
  "status": "ready"
}
```

And finally:

```
{
  "sessionId": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
  "status": "playing"
}
```

Then the process stops automatically when there is no more sound, resulting in this message:

```
{
  "sessionId": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
  "status": "over"
}
```

3.1.5. Playing a virtual music instrument

It is a Musical Performance ⇿ modalys.js communication situation.
Below are the main types of messages that are sent in performance situation.

3.1.5.1. Preparing for performance

When performance mode is initialized, the following message must be sent to Modalys to prepare for performance mode:

```
modalysChannel.publish("play",
  {
    "type": "pluckedString",
    "name": "MyBiChord",
    "content": [
      {
        "type": "chord",
        "name": "chordC",
        "content": [
          {
            "type": "number",
            "name": "length",
            "content": 1
          },
          {
            "type": "number",
            "name": "radius",
            "content": 0.0005
          },
          {
            "type": "number",
            "name": "tension",
            "content": 432
          },
          {
            "type": "string",
            "name": "material",
            "content": "steel"
          }
        ]
      },
      {
        "type": "chord",
        "name": "chordEb",
        "content": [
          {
            "type": "number",
            "name": "length",
            "content": 0.915
          },
          {
```

```
            "type": "number",
            "name": "radius",
            "content": 0.0005
          },
          {
            "type": "number",
            "name": "tension",
            "content": 510
          },
          {
            "type": "string",
            "name": "material",
            "content": "steel"
          }
        ]
      }
    ]
  }
);
```

Parts

Some instruments are made of one single object (ex: a membrane), others from multiple similar objects called “parts” (ex: a simplified violin made of4 strings, a xylophone made of 12 bars etc.). In some other iMuSciCA documents, parts are sometimes called “subobjects”.

Each part has an ID (unique among parts). This ID will be used for mapping the right gesture to the right part (i.e.: which string is being plucked in a guitar).
For each instrument, the part IDs are the names of the subobjects of a specific type for this instrument : “chord” for *pluckedString* / *guitarInstrument* and *bowedString*, “surface” for *hitBar* and *hitRectPlate*, “circleSurface” for *hitCircMembrane* and “squareSurface” for *hitRectMembrane*.
So in our example, we have two parts with IDs “chordC” and “chordEb”.

3.1.5.2. Notification when ready to play

As for the *try*, modalys.js sends “preparing”, “processing” and “ready” notifications when receiving a *play* request. Then a “description” notification is sent which includes a list of the instrument performance parameters:

```
{
  "sessionId": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
  "status": "description",
  "partsNumber": 2,
  "performanceParameters": [
    {
      "name": "pluckpoint",
      "type": "float",
      "default": 0.6,
      "min": 0.001,
      "max": 0.999
```

```
    },
    {
      "name": "position",
      "type": "float",
      "default": 0.1,
      "min": -0.5,
      "max": 0.5,
      "try": [[0, 0.5], [0.01, -0.5], [0.02, 0.5]]
    },
    {
      "name": "listeningpoint",
      "type": "float",
      "default": 0.3,
      "min": 0.001,
      "max": 0.999
    },
    {
      "name": "outputgain",
      "type": "float",
      "default": 2,
      "min": 0
    },
    {
      "name": "pitchbend",
      "type": "int",
      "default": 0,
      "min": -200,
      "max": 200
    }
  ]
}
```

A good practice is to save the sessionId when one of these notifications occurs, as you will need it later for all the operations performed in this "play" session. For example, when it's "ready":

```
var sessionId;
modalysChannel.subscribe("notification", function(data) {
  if (data.hasOwnProperty('status') && data.status == 'ready') {
    sessionId = data.sessionId;
  }
  console.log(data);
});
```

3.1.5.3. Performing

During the performance, gesture data are being sent to modalys.js in real time, and several parameters can be changed at once:

```
modalysChannel.publish("updateParameter", {
  "sessionId": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
  "parameters": [{
     "name": "plectrumPosition",
     "partId": "chordC",
     "value": 0.03,
     "when": 0.01
   }, {
     "name": "stringPluckPoint",
     "partId": "chordC",
     "value": 0.49
   }]
});
```

The **when** field is optional. It creates a linear interpolation between the current value and the new one, during the specified time (in sec.). If absent, the parameter change is instant.

3.1.5.4. Pausing, resuming or ending a performance

A session can be paused or resumed instantly ; the effect in modalys.js will be to disconnect/reconnect the web audio node attached to the virtual instrument. But the current state of this instrument is kept “frozen”. This following request must be sent:

```
modalysChannel.publish("pause [or resume]", {
  "sessionID": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
});
```

To end a performance session, send the following request:

```
modalysChannel.publish("stop", {
  "sessionID": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
});
```

And finally, once the instrument is actually terminated, modalys fires this:

```
{
  "sessionID": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
  "status": "over"
}
```

3.1.6. The server side eigenvalue solver for 3D objects

Although not used in the iMuSciCA project, the ability to create sounding objects from 3D mesh is one of Modalys’ particularities. This is, however, the most demanding mathematical aspects of the core engine.

The static mode computation increases exponentially upon the mesh’s granularity, with a generalized eigenvalue problem under the hood. For that reason, this very CPU-intensive initial

phase is processed server-side by the **eigenvalueproblemsolver** command line (based on Lapack[6] and Superlu[7]) with asynchronous client requests (AJAX) .

[6] http://www.netlib.org/lapack/
[7] http://crd-legacy.lbl.gov/~xiaoye/SuperLU/