A WEB BASED ENVIRONMENT EMBEDDING SIGNAL PROCESSING IN MUSICAL SCORES

Dominique Fober **Yann Orlarey** **Stéphane Letz** **Romain Michon**
Grame-CNCM
fober@grame.fr

ABSTRACT

We present an online environment for the design of musical scores, also allowing for the embedding of signal processors and hence the publication of electronic works. This environment is part of the INScore project. Its latest version has been transcribed to WebAssembly/Javascript to provide in a web browser the same features as in its native counterpart: the diversity of music representations supported by INScore, the interaction capabilities and all the dynamic aspects of the score.

After some historical elements about distributed musical scores, we will provide some reminders about the INScore project and its associated description language. We will then describe the architecture of the system and the choices made for its portability to the Web. Then, we will present the extensions specific to the Javascript version and in particular the support of signal processing objects. Finally, we will show how INScore's communication system has been extended to allow online musical score control from a native version of INScore, paving the way for real-time performance on the web.

1. INTRODUCTION

The deployment of music notation tools on the Internet has been investigated since the late 1990s. The Guido Note Server [1], designed as a client-server architecture and based on the Guido Music Description Language [2] (GMN) is an example of such systems. It was followed by a large number of applications offering online music editing services in a design modelled on traditional score editors (e.g., MuseScore [1]), enhanced by sharing services. In this area, we can mention Noteflight, [2] Scorio, [3] , or also in the line of description languages associated to compilers, LilyBin [4] or the GuidoEditor, [5] the latter having the particularity to embed the compiler in a web page. All these systems are based on a traditional approach of musical notation and do not deal with problems related to network performances.

It is more recently and often thanks to the impulse of composers that distributed score systems have emerged. Quintet.net [3] – an interactive Internet performance environment enabling up to five performers to play music in real-time over the Internet under the control of a *conductor* – is among the first performance-oriented notation systems. The Decibel Score Player [4] is another approach to distributed musical score, based on a purely graphical notation music (as opposed to symbolic notation). It allows for the synchronization of the scores of a performing ensemble. However, these systems are implemented as native applications (Quintet.net is based on Max/MSP and the Decibel Score Player is a standalone application for the iPad) and are therefore potentially not suited to be distributed on the web.

Facing similar problems, SmartVox [5] uses a standard browser to distribute and synchronize musical scores, which are also accompanied by audio signals. In the same line but with a focus on improvised music, John, the semi-conductor [6] is another web-based approach to music notation. It is more recently that Drawsocket [7] appears, a platform for generating synchronized, browser-based scores across an array of networked devices. Firmly rooted in web technologies (i.e., SVG, CSS, HTML and Javascript), it provides an API to develop networked scores.

INScore [8] (presented in section 2), is an environment for designing dynamic, interactive and augmented musical scores, built as a message-based system and controlled by OSC messages. It is naturally oriented towards network communication and is also open to web uses [9], in particular due to web server objects (`http` or `websocket`) that can be embedded in a score, and by providing a basic Web API allowing us to interact with the score from a browser. However, this approach is inherent to the native application and constrains its use as a client/server architecture, limiting both the ability to interact with the score and its dynamical aspects. We have therefore developed a Javascript version of the INScore environment, in the form of a library that can be integrated into a web page as a standalone engine. Taking advantage of the modular architecture of the Web, in particular thanks to the Node Package Manager (NPM), this implementation makes it possible to embed the Faust compiler [10] [11] and thus to provide signal processing objects within the score. This type of extension was never considered for the native version be-

[1] MuseScore `https://musescore.com/`
[2] Noteflight `https://www.noteflight.com/`
[3] Scorio `https://www.scorio.com/`
[4] LilyBin `http://lilybin.com/`
[5] GuidoEditor `https://guidoeditor.grame.fr/`

cause we considered that collaboration of specialized applications was a flexible model to implement. It is indeed possible to use INScore with Max/MSP as well as with SuperCollider or any other audio application, as long as it can communicate via OSC. The equivalent model of the Web lies rather in the aggregation of components.

Finally, a simple extension of the existing communication scheme has been designed to allow a web score to be controlled from the native version of INScore.

The next section is a quick review of INScore's approach to music representation. The following sections will detail the more technical aspects of the implementation for the web, the integration of Faust objects and the extension of the communication scheme, before concluding with the new perspectives offered by this environment and future works.

2. INSCORE ENVIRONMENT

INScore is an environment for the design of augmented, dynamic, and interactive musical scores [12]. It is the result of numerous research works dealing in particular with the extension of music notation to arbitrary graphical objects, time synchronization in the graphic space [13], dynamic and interactive scores [14], performance representation [15], and the extension of the score to network dimensions [16]. The design of a score is based on a specific scripting language and therefore also addresses the field of programming languages for the description of music [17].

2.1 Extended Scores

INScore allows you to extend symbolic notation practices, or even replace it, with arbitrary graphical objects: images, text, vector graphics, and videos. All these objects, including symbolic notation, have the same status as musical objects and an identical temporal dimension (i.e., date, duration and tempo). This homogeneity makes it possible to synchronize them in arbitrary combinations.

2.2 Representing the Time of Heterogeneous Objects

INScore takes advantage of the homogenous temporal dimension of the score objects to provide what we call *time synchronisation in the graphical space*, making it possible to represent the temporal relationships between objects using a synchronization mechanism. If we imagine that each pixel of an object carries a date (computed from the date and duration of the object) the synchronization system potentially makes it possible to graphically align all the pixels of two or more objects carrying the same date. The design of a cursor positioned at the current date of a score is achieved with a simple synchronization command. But above all, it becomes possible to reason in the temporal space and therefore in a metaphor close to musical thought, the rendering engine automatically *translate* the temporal dimensions in the graphic space.

2.3 Dynamic and Interactive Scores

The notion of *tempo* is an integral part of the temporal dimension of objects. By default, this tempo is set to zero: the object is motionless in time. When the tempo value is not zero, the object then moves in time at the speed specified by its tempo. Associated with the synchronization system, the use of the tempo allows you to create dynamic scores, whose form and content can evolve autonomously in time.

An interaction system complements these dynamic aspects. The time of a score, conceived as musical time, relative to a tempo, can also be *event based*, i.e. relative to asynchronuous events, which can be programmed in an arbitrary way. Among these events are the classic user interfaces events (such as mouse clicks, for example) but also *temporal* events, whose occurrence depends on the flow of time. Each object of the score is therefore capable of monitoring arbitrary events, including in the time domain: each event is associated with a set of messages that will be triggered at each occurrence of the event. These messages, expressed in INScore's scripting language, potentially allow the re-programming of all or part of the musical score.

2.4 The Network Dimensions of the Musical Score

INScore was originally designed to be driven by Open Sound Control (OSC) messages. It is therefore particularly suited to networks and the exchange of messages between INScore scores are native features. A simple *message forwarding* system, allows a score to control a set of other scores distributed over a local network, or to build a distributed music system on a client/server model [18].

As mentioned before, a score can also embed a web server [9], making it available from the Internet and providing control from a browser. On the other hand, the objects of a score can refer to resources distributed over the Internet, similarly to a browser that can aggregate content from different websites.

2.5 INScore Scripting Language

A score is described in a specific scripting language consisting of a textual form of OSC messages, extended with variables, control primitives, as well as symbolic score composition primitives. The following script is used to concisely describe the score in Figure 1:

```
/ITL/scene/title set txt "This is my first score !";
/ITL/scene/title scale 3;
/ITL/scene/title y -0.6;
/ITL/scene/title fontFamily Zapfino;
/ITL/scene/frame set rect 1.5 0.5;
/ITL/scene/frame color 230 230 230;
/ITL/scene/score set gmn '[ \meter<"4/4"> \key<-1> a f
g c c g a f ]';
/ITL/scene/score scale 0.6;
```

In fact, the above script mixes two languages: the INScore language whose general form is '/address parameters...', and the Guido language [2] which is used to specify the content of the `score` element.

This is my first score !

Figure 1. A simple score, described in a few lines.

3. INSCORE WEB ARCHITECTURE

INScore is based on a Model View Controller [MVC] architecture: the Model is an abstract description of the musical score, it includes all the properties of the elements which are organized in a tree, in a strictly similar way to their OSC address. The View is a graphical representation of the Model. The controller takes input messages, decodes them to modify the Model and when necessary, activates the refresh of the View at regular time intervals (every 10ms by default). This architecture was used to differentiate the method of handling the Model and the View in the Web implementation.

3.1 INScore Model as a WebAssembly Library

Mozilla developers have started the Emscripten compiler project [19] on the Internet using the LLVM technology. It initially allowed for the generation, from C/C++ sources, of a statically-compilable and garbage-collection-free typed subset of Javascript named asm.js. This first approach has demonstrated that near-native-code-performances could be achieved on the Web. Asm.js has been followed by WebAssembly[6] [WASM], a new efficient low-level programming language for in-browser client-side scripting, faster than the previous approach.

The existing INScore Model, developed in C++, was compiled with Emscripten to produce a WASM library. As a result, the native and the Web versions share the "main" of the code, which greatly minimises the maintenance of both platforms.

3.2 INScore View as DOM Based Javascript Library

INScore View has been developed using Typescript,[7] a language which builds on JavaScript, by adding static type definitions, allowing the TypeScript compiler to validate that code is working correctly. It is compiled as a Javascript library.

The View implementation is entirely based on the Document Object Model [DOM] as defined by the W3C.[8] It creates HTML elements on the fly and makes an extensive use of SVG. Most of the score objects properties are translated into style attributes (as defined by CSS).

[6] WebAssembly `https://webassembly.org/`
[7] Typescript `https://www.typescriptlang.org/`
[8] DOM Specification

3.3 INScore Controller

The controller lies in both the WASM and Javascript libraries as shown in Figure 2. Actually, the only input of the INScore engine are text messages (unlike the native version which also accepts OSC messages). These messages can result from user actions or come from the network (see section 5.3). They are first parsed and then passed on to the objects of the Model.

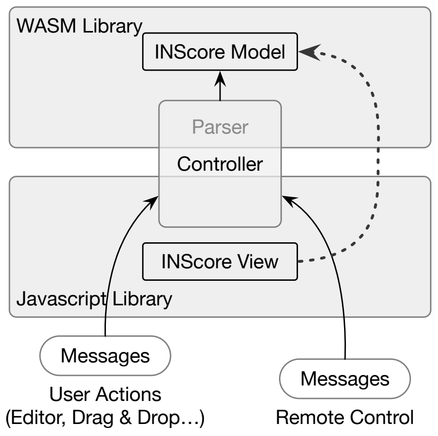

Figure 2. INScore Controller design. Input is collected from user action or received from the web to be passed to the WASM part of the controller. On changes, an update of the View is triggered and the View query the Model to synchronize.

4. SIGNAL PROCESSING EXTENSION

INScore Web can optionally embed the Faust compiler to provide signal processing objects within the score. Faust [10] is a functional, synchronous, domain-specific programming language working at the sample level, designed for real-time audio signal processing and synthesis. Faust programs can be efficiently compiled to a variety of target programming languages, from C++ to WebAssembly.

The Faust compiler is available as a WASM library [20] available as a NPM package[9] including a Javascript library providing a high level API to transform DSP code into a Web AudioNode.[10]

The type of a Faust object is `faust` and its `set` method (see Figure 3) takes DSP code as an argument. It is graphically represented by a browsable block diagram. Faust audio nodes can be instantiated as monophonic or polyphonic nodes, thus the `set` method takes an optional number of voices as illustrated below. When present (and even if equal to 1), a polyphonic Faust audio node is created.

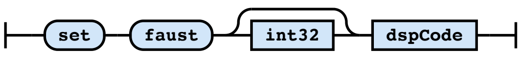

Figure 3. `set` method of Faust objects.

The following code creates a monophonic object named

[9] Faust NPM package
[10] The Web Audio API

karplus using the Faust physical modeling library, that is a ready-to-use, MIDI-enabled Karplus-Strong string with built-in UI.

```
/ITL/scene/karplus set faust
        'import("stdfaust.lib");
         process = pm.ks_ui_MIDI';
```

4.1 Faust Objects Methods

Faust objects carry all the properties common to INScore objects, including their temporal dimension. Their specific methods are the following:

- play: start or stop sound processing. Takes a boolean value ([01]) as an argument.

The next methods are only supported by polyphonic objects:

- keyOn, keyOff: similar to MIDI key on/off messages. Takes a MIDI channel, a pitch, and a velocity as arguments.
- allNotesOff: similar to MIDI all notes off message.

Faust objects support also specific query (get) methods, corresponding to read-only properties:

- in, out: gives the number of input and output signals of the Faust object.
- paths: gives the interface to the Faust object UI (as defined by the DSP code).

Paths returned by the path query are used internally to dynamically generate the address space of the Faust object (see section 4.2), providing control over the Faust node parameters.

4.2 Faust Objects Address Space

Faust provides user interface primitives allowing for an abstract description of a user interface within the Faust code. This description is independent from any GUI toolkits/frameworks and it's the *architecture files'* [21] responsibility to instantiate this abstract description. In INScore, this abstract description is instantiated in the address space of the Faust object. Let's consider the following query addressed to the karplus object as defined by the example in section 4.

```
/ITL/scene/karplus get paths;
```

The INScore engine returns a list of UI elements as follows:

```
/karplus/params/freq hslider freq 440.0 50.0 1000.0
0.01;
/karplus/params/bend hslider bend 0.0 -2.0 2.0 0.01;
/karplus/params/damping hslider damping 0.01 0.0 1.0
0.01;
etc.
```

where the general form of an element is a sequence of

address: the address of a Faust audio node parameter.

type: the type of the UI element (ignored by INScore).

name: the name of the UI element (ignored by INScore).

default: the default value of the parameter.

min: the minimum value of the parameter.

max: the maximum value of the parameter.

step: the step of values (ignored by INScore).

The address field is used to expand the address space of the Faust object so that for the object
/ITL/scene/karplus
the addresses
/ITL/scene/karplus/karplus/params/freq
/ITL/scene/karplus/karplus/params/bend
etc.
become valid addresses taking a float value as an argument, that is passed to the Faust audio node to set the corresponding parameter value.

5. WEB COMMUNICATION

5.1 Server Side

INScore provides a *fowarding* mechanism [16] that can be used to distribute scores over a local network. The general form of the forward message is illustrated in Figure 4. It can be addressed to the application or to the scene level. It takes a list of destinations as argument or can be used with no argument to stop forwarding. A destination host is specified similarly to a url, by IP number or by host name, followed by a port number. A filtering mechanism is also provided to select the messages to be forwarded. This mechanism was basically designed to transmit OSC messages on UPD sockets. It has been extended for the native INScore application, to support different protocols, namely Websockets and HTTP. The new form of the forward message is illustrated in Figure 5.

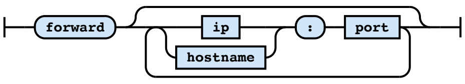

Figure 4. The forward message.

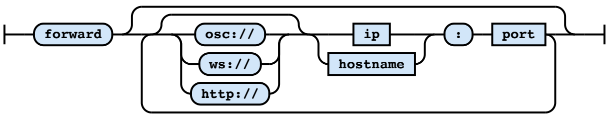

Figure 5. The extended forward message.

osc:// ws:// and http:// refer respectively to the OSC protocol, to Websockets and to HTTP. For compatibility reasons, the original form is preserved and implies the OSC protocol.

The OSC protocol runs over UDP and thus is connectionless, this is not the case for Websockets and HTTP that run over TCP. Currently and whether for Websockets or HTTP, the INScore server ignores the host name and accepts all incoming connections. This approach may be revised in the future to select authorized hosts.

5.2 Client Side

The OSC protocol is transparent on the client side: INScore has been natively designed to communicate via OSC. For Websockets and HTTP, an explicit connection must be initiated by the client and to this end, we have introduced the `connect` message whose form is similar to the `forward` message (see Figure 6). Used without argument, `connect` removes all the existing connections.

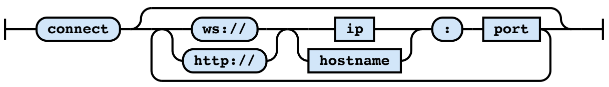

Figure 6. The `connect` message.

On client side, HTTP support is implemented over HTML Server-Sent-Events [SSE] API, a one way messaging system designed to allow a web page to get updates from a server. Websockets connections are bidirectional, but only communication from the server to the client is used for the time being.

Messages transmitted by the server are textual OSC messages that are parsed by the client upon receipt, in the same way as any input script.

5.3 Communication Scheme Overview

Figure 7 illustrates the overall communication scheme of INScore. Its extension to new protocols allows us - starting from a native version of INScore which then acts as a server - to control in parallel musical scores distributed over a local network and/or over the Internet.

The server receives as an input messages from user actions (e.g., drag & drop of scripts, interaction with the score, etc.) or generated by design according to the time flow of the score objects. Provided they are not filtered, these messages are automatically transmitted to all connected clients, making it possible to replicate and/or control a musical score on a whole set of targets.

6. CONCLUSIONS

The Web deployment of the INScore engine is an ongoing project started in 2017 in parallel to the native version development. The design issues encountered in the past were solved by the evolution of languages and compilation technologies, allowing us to support various platforms from the same source code.

The web version of INScore opens new perspectives, particularly relevant in the context of the current health crisis, where the web has taken a central place. The distribution of musical works involving electronic parts has never been straightforward. Being able to publish such works on the web, to make them available without prior installation, ready to be played from home, may constitue an essential

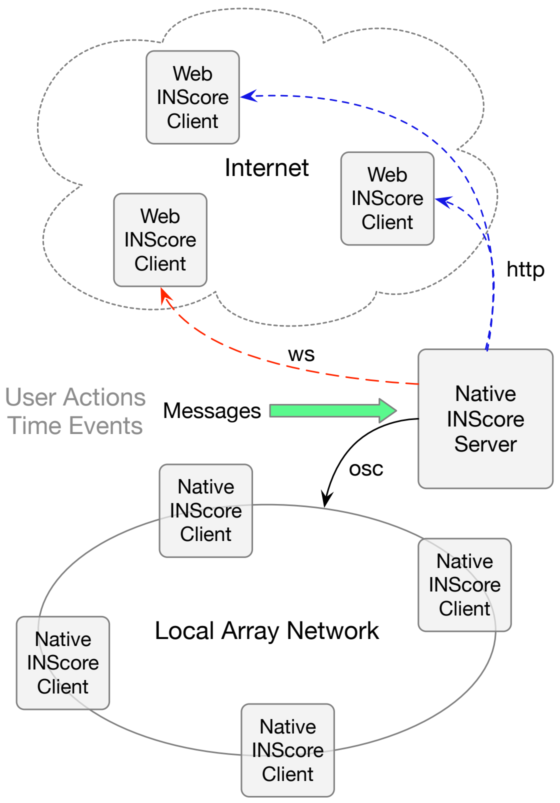

Figure 7. INScore communication scheme.

tool for the dissemination of contemporary creation. Encoding of existing musical works in INScore and Faust is underway with this in perspective.

Remote control could also open up new prospects, both from an educational point of view and for the performance of the music: a composer will be able to perform his piece from home, interacting dynamically with the performance, while a set of connected listeners will be able to follow this performance, having at their disposal both the representation of the work and its sound rendering. Distributed performance was already possible on a local network, the extended communication scheme brings it to the Internet.

The presented work will be available as a library on NPM. An INScore editor is online at

`https://inscoreweb.grame.fr/`

7. REFERENCES

[1] K. Renz and H. Hoos, “A Web-based Approach to Music Notation Using GUIDO,” in *Proceedings of the International Computer Music Conference*. ICMA, 1998, pp. 455–458.

[2] H. Hoos, K. Hamel, K. Renz, and J. Kilian, “The GUIDO Music Notation Format - a Novel Approach for Adequately Representing Score-level Music.” in *Proceedings of the International Computer Music Conference*. ICMA, 1998, pp. 451–454.

[3] G. Hajdu, “Quintet.net: An environment for composing and performing music on the internet,” *Leonardo*,

vol. 38, no. 1, pp. 23–30, 2005. [Online]. Available: https://doi.org/10.1162/leon.2005.38.1.23

[4] C. Hope, L. Vickery, A. Wyatt, and S. James, "The DECIBEL Scoreplayer - A Digital Tool for Reading Graphic Notation," in *Proceedings of the First International Conference on Technologies for Music Notation and Representation*. Institut de Recherche en Musicologie, May 2015, pp. 58–69. [Online]. Available: https://doi.org/10.5281/zenodo.1289610

[5] J. Bell and B. Matuszewski, "SMARTVOX - A Web-Based Distributed Media Player as Notation Tool For Choral Practices," in *TENOR 2017*, Coruña, Spain, May 2017. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01660184

[6] V. Goudard, "John, the semi-conductor : a tool for comprovisation," in *International Conference on Technologies for Music Notation and Representation (TENOR'18)*, ser. Proceedings of the 4th International Conference on Technologies for Music Notation and Representation, S. Bhagwati and J. Bresson, Eds., Montréal, Canada, May 2018. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01923258

[7] R. Gottfried and G. Hajdu, "Drawsocket: A Browser Based System for Networked Score Display"," in *Proceedings of the International Conference on Technologies for Music Notation and Representation*. Monash University, Jul. 2019, pp. 15–25. [Online]. Available: https://doi.org/10.5281/zenodo.3373369

[8] D. Fober, Y. Orlarey, and S. Letz, "INScore - An Environment for the Design of Live Music Scores," in *Linux Audio Conference*, Stanford, United States, 2012, pp. 47–54. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02158817

[9] D. Fober, G. Gouilloux, Y. Orlarey, and S. Letz, "Distributing Music Scores to Mobile Platforms and to the Internet using INScore," in *12th Sound and Music Computing Conference (SMC15)*, Maynooth, Ireland, Jul. 2015, pp. 229–233. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01348511

[10] Y. Orlarey, D. Fober, and S. Letz, "FAUST : an Efficient Functional Approach to DSP Programming," in *NEW COMPUTATIONAL PARADIGMS FOR COMPUTER MUSIC*, E. D. FRANCE, Ed., 2009, pp. 65–96. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02159014

[11] S. Ren, S. Letz, Y. Orlarey, R. Michon, D. Fober, M. Buffa, and J. Lebrun, "Using Faust DSL to Develop Custom, Sample Accurate DSP Code and Audio Plugins for the Web Browser," *Journal of the Audio Engineering Society*, vol. 68, no. 10, Nov. 2020. [Online]. Available: https://hal.inria.fr/hal-03087763

[12] D. Fober, Y. Orlarey, and S. Letz, "INScore - An Environment for the Design of Live Music Scores," in *Linux Audio Conference*, Stanford, United States, 2012, pp. 47–54. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02158817

[13] D. Fober, C. Daudin, Y. Orlarey, and S. Letz, "Time Synchronization in Graphic Domain - A new paradigm for Augmented Music Scores," in *International Computer Music Conference*, ICMA, Ed., New York, United States, 2010, pp. 458–461. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02158957

[14] D. Fober, S. Letz, Y. Orlarey, and F. Bevilacqua, "Programming Interactive Music Scores with INScore," in *Sound and Music Computing*, Stockholm, Sweden, Jul. 2013, pp. 185–190. [Online]. Available: https://hal.archives-ouvertes.fr/hal-00851956

[15] D. Fober, F. Bevilacqua, and R. Assous, "Segments and Mapping for Scores and Signal Representations," GRAME, Technical Report, 2012. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02158968

[16] D. Fober, G. Gouilloux, Y. Orlarey, and S. Letz, "Distributing Music Scores to Mobile Platforms and to the Internet using INScore," in *12th Sound and Music Computing Conference (SMC15)*, Maynooth, Ireland, Jul. 2015, pp. 229–233. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01348511

[17] D. Fober, Y. Orlarey, S. Letz, and R. Michon, "A Tree Based Language for Music Score Description." in *International Symposium on Computer Music Multidisciplinary Research*, Marseille, France, Oct. 2019. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02368958

[18] S. Zagorac and P. Alessandrini, "ZScore: A Distributed System For Integrated Mixed Music Composition and Performance," in *Proceedings of the International Conference on Technologies for Music Notation and Representation*. Concordia University, May 2018, pp. 62–70. [Online]. Available: https://doi.org/10.5281/zenodo.1289685

[19] A. Zakai, "Emscripten: An llvm-to-javascript compiler," in *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*, ser. OOPSLA '11. New York, NY, USA: Association for Computing Machinery, 2011, pp. 301–312. [Online]. Available: https://doi.org/10.1145/2048147.2048224

[20] S. Letz, Y. Orlarey, and D. Fober, "FAUST Domain Specific Audio DSP Language Compiled to WebAssembly," in *The Web Conference*. Lyon, France: International World Wide Web Conferences Steering Committee, 2018, pp. 701–709. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02158925

[21] D. Fober, Y. Orlarey, and S. Letz, "FAUST Architectures Design and OSC Support." in *International Conference on Digital Audio Effects*, IRCAM, Ed., Paris, France, 2011, pp. 231–216. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02158816