Composing and Performing Interactive Music using the HipHop.js language

Bertrand Petit
Inria/Cirm
Sophia Antipolis, Nice, France
bertrand.petit@inria.fr

Manuel Serrano
Inria
Sophia Antipolis, France
manuel.serrano@inria.fr

ABSTRACT

Skini is a platform for composing and producing live performances with the audience participating using connected devices (smartphones, tablets, PC, etc.). The music composer creates beforehand musical elements such as melodic patterns, sound patterns, instruments, group of instruments, and a dynamic *score* that governs the way these basic elements behave according to events produced by the audience. The platform allows to control the musical quality of the work, even if during the concert or the performance, the audience interacts with the system giving birth to an original music creation.

Skini scores are expressed in terms of *constraints on events* that control *which* musical elements are accessible to the audience and *when* they are available. Constraints may be instantaneous, for instance one constraint may disable violins while trumpets are playing. They may also be temporal, for instance, one constraint may prevent the piano to play more than 30 consecutive seconds.

The Skini platform is implemented in Hop.js [10], for the general infrastructure and most of the user interfaces, and in HipHop.js [13], a reactive synchronous DSL, for implementing the music scores. The HipHop.js constructs, which consist of temporal operators such as parallel executions, sequences, awaits, synchronization points, and preemption, form the core implementation language for expressing Skini musical constraints.

This paper presents the Skini platform and It reports about live performances and an educational project. Some musical pieces created with Skini can be found at:

`https://soundcloud.com/user-651713160`

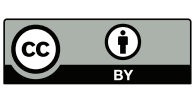
Author Keywords

Interactive Music, Web programming, Reactive synchronous programming

CCS Concepts

•Human-centered computing → Interaction design process and methods; Collaborative content creation; •Applied computing → Sound and music computing;

1. INTRODUCTION



NIME'19, June 3-6, 2019, Federal University of Rio Grande do Sul, Porto Alegre, Brazil.

In the 60's, the philosopher Umberto Eco, and musicians such as K. Stockhausen, K. Penderecki, and L. Berio wondered about the relationships between composers, musicians, and the audience [3]. In particular, Eco has shown that following the evolution of physics from Copernicus to Einstein, the perception of the world has evolved from being static to being more dynamic. This has impacted 20th century art where some musicians have tried to express this dynamics through works where the performer *and* the audience had a concrete impact on the musical result. Eco called these endeavors *open work* or *moving work*, in the sense that the final musical result is not strictly known in advance.

Recent technology improvements, mainly broad network coverage and device availability, have opened up new opportunities for interactive music, which has become an active field [1, 8, 14]. A tremendous effort has been devoted to improve the technical aspects of systems allowing individual interaction, see Swarmed [5] for instance. These technological challenges have attracted most attention, so frameworks dedicated to music composition of interactive performances following the *moving work*, proposing a clear composition scheme and the way to make it work, are rare. Skini is such a framework. It is meant for composing and executing interactive performances.

This paper present Skini with an organization that is as follows. Section 2 presents the related work. Section 3 presents the Skini key principles and concepts for creating and performing musical pieces. Section 4 focuses on musical interactive composition and orchestration. Section 5 presents the Skini software architecture and gives a taste of HipHop.js score programming. Section 6 presents two actual musical projects already executed with Skini. Section 7 presents directions for future work.

2. RELATED WORK

Taylor *et al.* overview the domain of collaborative music [12]. They present several developments aim to break the unidirectional communication, well established in musical performances, by involving the public in the process of musical creation. Three main directions emerge: social behavior, technology, and impact of interaction. We think that there is a fourth direction that is still not clearly addressed: composition issues for interactive music. Here are some examples.

Social behavior: In [1] the authors studied *Computer Mediated Musical Collaboration* by exploring how to use smartphone mediation in musical contexts. They report on the activities of the Smartphone ensemble at the University of Caldas. This study focuses on the design of the graphical interface and social behavior of an audience. In the same vein, [9] focuses on the level of interaction necessary for good public participation, based on the experiences of the Stan-

ford Mobile Orchestra. The results of these experiments are valuable inputs for Skini.

Technology: In [8], the authors introduce a way to distribute mobile applications using mobile ad-hoc network. This study mostly focuses on telecommunications technology. Similarly, [6] proposes a method of communication by using DTMF (Dual Tone Multiple Frequency) with high frequency sound. Skini uses web technologies, hence it is network support agnostic.

Impact of interaction: Opened band [11] proposes a system based on multi-user chat where text entries are translated into sounds. Similarly, [4] presents two sound installations performed at the Peninsula Arts Contemporary Music Festival 2016, at the University of Plymouth, focusing on the experimentation of collaborative music using voting system, movement tracking and indoor positioning. More recently, the Cosima project at IRCAM [7] explores the relationship between the body, media and space through new interfaces and collaborative creation tools. Cosima is a good example of using smartphones and Web technologies to control and to generate sound using smartphone speakers.

MassMobile [14] and Open Symphony [11] are architectures close to Skini. They present mass audience participation tools based on client-server systems dedicated to the production of live shows intended for a medium and large audience. In Open Symphony, improvisations by performers are controlled by audience members who specify collectively playing modes by a voting system. MassMobile is connected to Max/MSP and can be deployed in a variety of performance contexts as far Max/MSP is used. This platform is a complete interactive solution, and does not implement a particular composition process as skini proposes.

Musical Composition issues: More in line with our musical composition issues, and especially regarding how to achieve an engaging experience for the audience members, Jieun Oh and Ge Wang in 2011 [9] described the challenges a composer faces when the audience participates in a music performance, such as controlling uncertainty factor, and defining roles between the audience and the composer/performers.

3. SKINI BASIC CONCEPTS

Skini organizes the music in short musical sequences called patterns. Playing the music consists in selecting some of these patterns. This is the role of the audience during the show that must select patterns from a set of availabilities that depend on previous audience selections and on the musical path described in the composer score. In order to help the participants select patterns they like, they can first listen to them using their smartphones, and then *activate* the ones they like. To ensure the musicality of the work, the composer controls which patterns are proposed to the audience by creating coherent groups. The art of managing the availability of the groups of pattern is the *orchestration*, in reference to the common use of this word in orchestral music.

3.1 Patterns

A pattern can be any part of a score, a MIDI sequence, or a pre-recorded sound. For the sake of interaction, patterns are generally short (few seconds), as longer patterns would monopolize the instruments, lengthen the waiting time of sequences to be played by one instrument, and thus reduce the interaction with the audience. When activated, a pattern can be played as is, or it can be changed in real time depending on how the composer imagines the show's behavior. For instance, a MIDI sequence pattern can be transposed to another tonality. Moreover, we are currently using an interface where patterns can also be created by the audience using a dedicated designer tool.

As patterns are activated by the audience, the composer cannot controlled in advance the result of the interpretation of his score. He must manage the uncertainty introduced by the interaction by thinking about the work in terms of *group of pattern's architecture*, in the sense of managing the available groups of patterns according to the aesthetic project.

Using pattern activation to create original music is not new. Twentieth century classical composers such as Stockhausen (*Klavierstück XI*), Henri Pousseur (*Scambi*), and Boulez (*Troisième Sonate pour piano*) have used similar concepts. The originality of Skini is the delegation to the audience for the activation. Since it is no longer the musicians on stage that play the music, the composer is forced to give up on the traditional way of creating music. The interaction with the audience must be inherent to the creativity process. This gives a new dimension to U. Eco's openness.

3.2 Orchestration

In the common musical language, orchestration is the temporal description of the instruments actually played. The instruments are arranged in groups: violins, violas, cellos, double basses, horns, percussion, etc. The composer chooses the groups for each specific part of his work. In Skini, the orchestration consists in grouping patterns and defining constraints in between them. The orchestration governs the interaction with the audience but the audience reacts to what it is listening. This is an *open work* according to U. Eco's definition.

The composer defines the orchestration evolution according to various constraints and parameters such as time, and audience behavior. The audience behavior is defined by the activated pattern combinations. At a given time, the orchestration establishes the status of the groups of pattern (available or not available) for groups of people in the audience. Such a system, reacting to events (activation and time) to reach a defined state of the orchestration, is modeled by automata. The HipHop.js language has been precisely created for automata programming. We will see how it fits Skini architecture.

3.3 Interaction

Skini distinguishes three different roles people in the audience may play.

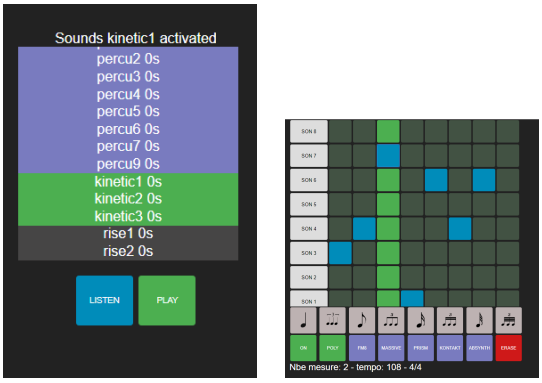


Figure 1: From left to right, (i) the Actor interface, (ii) the Skini sequencer.

1. Actors: they are those among the audience that activate patterns, generally using the web browser of their smartphone (Figure 1(i)). By selecting patterns,

actors generate events which are handled by the orchestration automaton.

2. Designers: they create patterns during the show using the Skini sequencer. It runs in a web browser (Figure 1(ii)) and is also expected to be accessed via smartphones or tablets. Its interface mimics those of traditional MIDI launchpads. It allows designers to create loops synchronized with all the other patterns.

3. Conductors: are those who give their opinion or propose a global evolution of the performance. Their web interface is not shown here.

The software component that controls the orchestration is the *automaton*. It executes the rules and satisfies the constraints defined by the composer. In other words, the automaton *runs* the score. The complexity of the score all depends on the composer will. It can implement complex automatic strategies based on the behavior of the audience or it can be very lightweight and delegates most of the control to a controller that manually supervises the show.

4. COMPOSITION PROCESS

Whatever the kind of music to be created, the steps to be followed by the composer are the same: 1) design all the patterns, 2) create the initial orchestration, 3) simulate the audience behavior, and 4) modify the orchestration and repeat step 3 until the result is nice enough.

4.1 Pattern Design

The only constraint Skini imposes on patterns is their duration: all the patterns available simultaneously must have the exact same duration. This guarantees the rhythmic coherence of the piece. On the other hand, the *sound* of a pattern is totally free. For instance, a pattern can be a fragment of a melodic sentence. It can use a basic rhythm, and the composer can build the others on variations of this rhythm. It can be part of a family of patterns created by transposing an initial one. A pattern can also be a complex synthesized sound, or a sound sample, etc. Figures 2, 3, and 4 show three representions of the same simple pattern which can be played in three different ways: Figure 2 by a synthesizer, Figure 3 by a musician, Figure 4 by a sound file player.

Figure 2: Example of a simple MIDI pattern

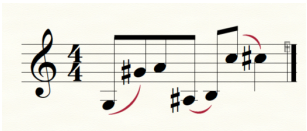


Figure 3: Example of a simple pattern score

The main difficulty for the composer is to imagine patterns adapted to all, or most, of the orchestral scenarios that will be created in the second step.

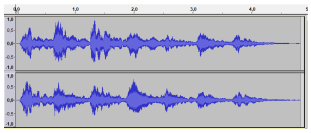


Figure 4: Example of a pattern sound

4.2 Orchestration Design

The composer designs the evolution of the orchestration that must anticipate audience interactions. For that, the orchestration must accommodate different paths through the events generated by the acting audience. How to design this orchestration and how to represent it are open questions. For the sake of the example, Figure 5 shows an early representation on the paper we used for one of our first piece. This representation is close to what is usually used for traditional musical scores. On the left (Y axis) are the groups of pattern. On the X axis is a timing view of the orchestration. We use the following symbols to represent the activation of groups of pattern:

- ▼ denotes a forced group activation, without waiting for an action by the audience;
- ▲ is denotes a forced group deactivation;
- • means that a group can be activated if one or several previous events linked by arrows occur;
- × means that a group can be deactivated if, one or several previous events linked by arrows, occur.

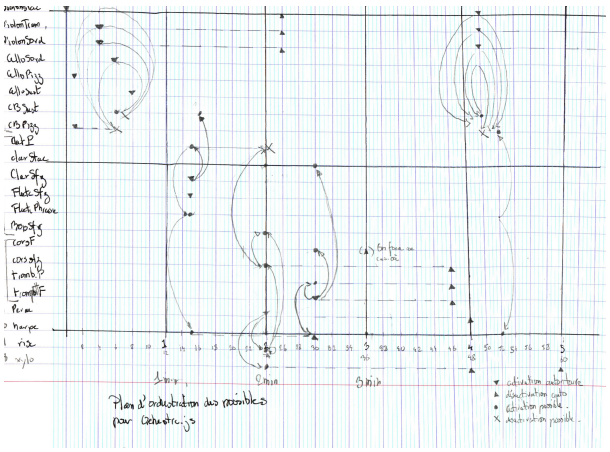


Figure 5: An orchestration on paper

Additionally, arrows are labeled with numbers to count the number of events and horizontal dash lines show the availability period for a group of pattern.

This representation is imprecise. It lacks many details that only the actual HipHop.js program can express. However, in our experience, this sort of graphical representations have proved themselves to be particularly useful.

4.3 Simulation

It is difficult to imagine how the implementation of the orchestration will actually work, but it is basically the art of composing music to deal with this uncertainty and openness. This is why patterns and orchestration need simulation tools. We have created one for Skini. It behaves like an audience that would select patterns randomly. The simulator is controlled by three parameters: the minimum response time of people in the audience, the maximum response time, and the waiting time for a pattern to be played. Although simple, these three parameters are sufficient to simulate realistic rehearsals, at least, sufficient for the com-

poser to figure out the flavor and shape of the music he is composing.

5. SKINI ARCHITECTURE

Skini relies on a web architecture (Figure 6) close to the one used by the Open Symphony [11]. The core system is implemented in Hop.js, a multi-tier JavaScript programming language. The orchestration is implemented in HipHop.js that implements the core synchronous reactive primitives of Esterel [2]. In this section we present the main part of the system architecture, and we introduce the orchestration programming.

5.1 System

The individual Skini components are conceptually distributed over several computers but of course, a single computer to run them all if the complexity of the performance is not too CPU demanding. The components are:

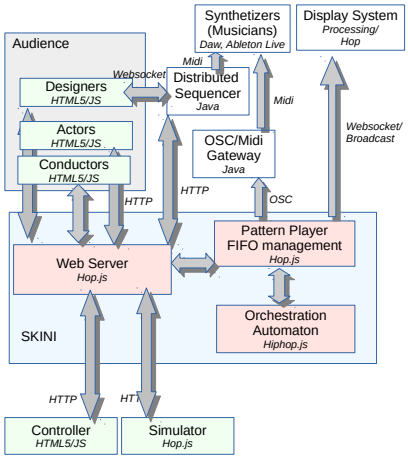


Figure 6: Logical view of Skini.

Web Server: it is in charge of all the interactions with the audience and it controls all the other components of the platform. The orchestration automaton runs inside the web server.
Actor and Controller: they are dynamic HTML5 clients connected to the Hop server, communicating using HTTP and websockets.
Simulator: this is a Hop.js client using the same protocols as the actor. It behaves like an audience by generating automatically pattern activation. Parameters allow to dimension the size of the simulated audience.
Pattern Player: the Pattern Player manages the requests coming from the audience. It uses waiting queues that accumulate the patterns that cannot be played instantaneously when all the synthesizers or musicians are busy.
Orchestration Automaton: this is the software component that runs the composer score. It reacts to audience events and selects which patterns are available.
Synthesizers: this is the component in charge of playing the actual music. It communicates with the other components using Open Sound Control (OSC) commands. A Digital Audio Workstation (DAW) such as Reaper or Ableton Live runs the synthesizers.
Designer Interface and Distributed Sequencer: The Designer component is an HTML5 client, which communicates with the Hop.js server and the Distributed Sequencer using HTTP and websockets. The Distributed Sequencer plays the sequences of the Designers and is synchronized with the Pattern Player. The Designer client display is synchronized with the Distributed Sequencer.
Display System: it displays various information to the users, for instance, who is selecting what in the audience. It communicates with the rest of the application using HTTP protocols.

5.2 HipHop.js Flavor for Orchestration

We motivate the usage of the HipHop.js language for implementing orchestrations with a simple example. We show the implementation of an orchestration that initially proposes to the audience cellos, and violins playing staccato patterns. After five ticks (where ticks are defined by the music tempo), percussion patterns are made available and then, after the first percussion pattern has been played, the cellos can play five patterns and stop. In parallel violins should stop after 4 cello events or after 10 ticks. Figure 7 shows the time events of that orchestration.

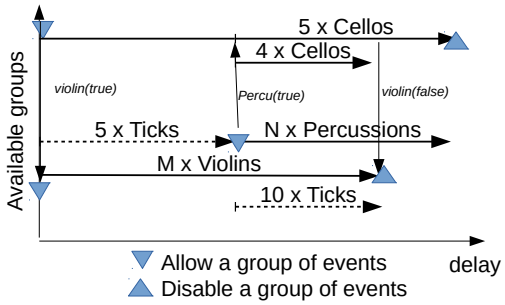


Figure 7: Simple example.

This example is sufficiently complex to illustrate the main benefit of HipHop.js. We left to the reader the exercise to implement it using traditional sequential programming languages.

A HipHop.js execution is split in a succession of *reactions* that are triggered by external events associated with user interactions. The purpose of a reaction is to *select* the next expression Hop.js will execute. An HipHop.js program is organized as a list of modules that are loaded into a *reactive machine*. A module specifies the signals it can receive and emit. The module implementing our example is as follows:

```
hiphop module orchestration(
    in Tick, out Perc, in PercPlaying, out Cello,
    in CelloPlaying, out ViolinStacatto) {
   emit ViolinStacatto(true);
   emit Cello(true);
   await count(5, Tick.now);
   emit Perc(true);
   await (PercPlaying.now);
   fork {
      await count(5, CelloPlaying.now);
      emit Cello(false);
   } par {
      await count(4, CelloPlaying.now)
            || count(10, Tick.now);
      emit ViolinStacatto(false);
   }
}
```

The module `orchestration` first emits two output signals `ViolinStacatto` and `Cello` with a value `true`. Then, it waits for five ticks before emitting the signal `Perc` with a value `true`. These emissions will be received by the Hop.js web

server that will modify the status of the orchestration accordingly. The value `true` means that the patterns in a group of pattern (here Cello, Perc and ViolinStacatto) can be activated by the audience. These emissions will also update the web interface of all connected participants in the audience. A value `false` does the opposite. It deactivates a group of pattern. Activating a pattern in the audience generates a *playing* signals corresponding to the group the pattern belongs to. For instance, we someone selects a percussion, the signal `PercPlaying` is emitted. It enables the HipHop.js to proceed by executing the `fork/par` constructs that follows.

The `fork/par` control structure runs its branches in parallel and waits for all of them to complete. The branches can communicate and synchronize with each others by broadcasting signals that are instantaneously delivered. During a reaction, all branches of all parallel constructs receive the exact same information. The HipHop.js execution is synchronous and deterministic.

HipHop.js can block on complex predicates. For instance, in the second branch of the `fork/par` construct, it waits for the first condition of 4 `CelloPlaying` events or 10 `Tick`. HipHop.js imposes no constraint on conditional expressions. They can mix arbitrary temporal expressions and JavaScript expressions.

This example, although simple, illustrates the benefit of HipHop.js. The constraints of the musical orchestration are mapped directly into constructs of the language without extra bookkeeping or encoding techniques. This greatly alleviate the composer/programmer task as it dramatically reduces the distance between the representation of his score and its actual executable implementation.

6. MUSICAL PERFORMANCES

Skini has already been used in different contexts. We present two experiences here. The first one is a show performed in 2017 during the *MANCA festival* of contemporary music in Nice, France. The second one is the *Fabrique à Musique*, a project funded by the SACEM (Music Composer Society).

6.1 Golem, MANCA 2017

The Golem show is based on the legend of Golem in Prague who was created in the 16th century in order to protect the Jewish community. On stage it involved a calligrapher and an actress/singer. The calame (pencil) of the calligrapher was used as a percussion producing sounds which were processed in real-time. The show lasted 45 min, along with two interactive scenes of about five minutes. There is a short video of the show available at: `https://www.youtube.com/watch?v=MuZfpSgsIPo`.

We ran the show twice, with two different categories of spectators. On 8 December 2017, with two classes of young pupils, and on 9 December 2017 with a more classical panel of 150 spectators. Before the show, some instructions were given in order to explain the audience how to access the private WiFi network, how to get connected to the server, and the main features of the interface. The role of the audience in the story was explained and emphasized.

6.1.1 Golem: 8 December 2017

There were 42 pupils of 12 years old, highly motivated by the idea of using a smartphone during a live show. Though the smartphone is an important device for them, generally it is forbidden in schools. The idea of using smartphones certainly contributed to their active participation both to the show, and also to the "question and answer" session that followed. According to the system logs, 42 users tried to access the system, that is, all the pupils. This was not surprising as the pupils were informed and prepared by their teachers. During this show we had two sessions of interactions of 4 and 5 minutes each. This generated 4890 events (2400 in session 1; 2490 in session 2), 689 patterns were activated by the pupils (228 in sessions 1; 461 in session 2) which corresponds to 1.3 pattern activation per second.

6.1.2 Golem: 9 December 2017

There were 150 people, mostly the MANCA festival public who were aged between 30 and 70 years, with the mean age of around 40-50 years, and thus very different than the previous day. During both sessions, around 90 people accessed the system. That is 40 % of the spectators did not interact, or did not succeed in using the application for several reasons, such as handset incompatibility or misunderstanding of the process. During this show, we had two interactive sessions of 3 and 6 minutes each. This generated 4099 events (2171 in session 1; 1928 in session 2), 754 patterns were activated by the public, which makes roughly a mean of 1.4 pattern activations per second.

6.2 Take Away from Golem Shows

The first observation is that the audience actively participated in the show. This is demonstrated by the level of activity during all the interactive sessions. This shows that the concept of the interaction has been well received by different public. Unsurprisingly we notice that the pupils were in proportion much more connected than the standard public with a much better rate of connection (100% against 60%). Surprisingly, we found that the distribution of activity among the public is very similar. This means that the proportions of active, low active, or inactive people did not depend on the type of audience. We expected a more stable average activity per individual with the young population than with the public. For both populations, the diversity of behaviors we have observed led us to vary the type of interface by developing the *designer* interface in particular.

6.3 La Fabrique à Musique 2018/2019

The Skini platform and a composer were selected by the SACEM for the creation of a show by a class of 23 young pupils (12 years old). The goal was to create a complete musical work which was presented officially in a concert hall of the Nice Conservatory of Music. (See in French at `https://www.youtube.com/watch?v=Zp0eUvIFqvk`)
This project is much more than a performance. It is a complete educational approach. The Skini platform is not only used for playing music created by a single composer, but also for creating collaborative music. This means that the pupils have to first understand what is a pattern and an orchestration. They had to learn how to manipulate the *designer* interfaces of figure 1 using tablets. The creation of the patterns was done in small groups of 6 pupils maximum, in five sessions. The orchestration was done by the composer and validated by the class. The show took place with the pupils activating the musical work in real time.

7. NEXT STEPS

The simulator used during the composition process can be used as an *automatic music generator* by merely recording a simulation session. The extracted score is complete and can be played as is by musicians. However, the simulator is currently overly simple and the music it generates is just good enough for testing but not as a final product. The evolution of the simulator into a realistic generator is a research topic in itself that we plan to explore.

We usually run our own dedicated WiFi network for the performances, but we have also ran conclusive experiments with plain internet connections. Skini requires a very limited bandwidth as the amount of data exchanged in the communication is low. We plan to implement Skini on the Cloud as this will allow performances to be run in public contexts such as in the streets, public places, museums and stations.

Currently, Skini synchronizes the patterns created by the composer and those created by the designers, but patterns created in real-time can only be played by the designers. We would like to remove that constraint in order to let patterns created by some members being integrated in real time into the current musical piece and played by other participants. This would make it possible to achieve performances where the musical piece would be entirely created and manipulated by the audience.

In our experiments the motivation for the audience to participate in the music was high. In Golem, probably because the audience actions were part of the story telling and because the audience knew that it had something to do during certain periods of the show. For the other performances, we suppose that the novelty of the system was enough to attract attention. This interest might not last too long... In the current implementation, the actor interface is very simple. This is a conscious design decision, but we believe that this interface, being very simple, is not engaging enough because it is too disconnected from the music being played. Thus, the actors influence is not intuitive enough to feel the real impact on the musical narration. We are investing new ways to display the actions of each participants in order to provide a graphical feedback that could complement the impact on the music being played. This will certainly be possible by finding different ways of staging the score, by displaying its evolution in real time using graphical representations on a large screen. Another possibility comes with the pattern model which offers the opportunity to link non musical information to the patterns. In the first version of Skini, the simple actor received some code names for the patterns, and used it to listen to the patterns and activate them. We imagine that we can bring new ideas at this level. Instead of giving code names to the audience we can give sentences, with real meaning, and use this sentences to generate a text in real time.

8. CONCLUSION

Skini is a platform for interactive performance compositions. It takes into account an uncommon dimension in the composition process: the audience behavior. By contrast with classical music, where scores are used to define the musical discourse according to time, and sometimes according to the decision of the performers, in an interactive score the composer has to think about the audience behavior, the type of interactions he wants to occur according to the aesthetic of the project.

Skini is the result of the search for a solution that combines the constraints of interaction, and those of a musical discourse using scores. Skini offers a solution that is based on a few basic concepts, but whose musical implementation has proven to be complex. Indeed, we have chosen to implement our performances based on groups of musical patterns whose access is offered to the audience according to different criteria essentially related to the behavior of this audience.

Following the composition and performances of various musical pieces using Skini, we have demonstrated that our basic concepts allow to produce music pieces where listeners feel involved in the production of a coherent musical structure.

In addition, the first attempts to use Skini in music education are promising. Skini gives the opportunity to progress step by step, in a group and in interaction with a composer or teacher while following a simple methodology. This allowed to discuss with schoolchildren various musical subjects ranging from the design of simple musical phrases to the design of a complete musical discourse, in order to produce a real musical performance.

9. REFERENCES

[1] J. J. Arango and D. M. M. Giraldo. The Smartphone Ensemble. Exploring mobile computer mediation in collaborative musical performance. *Proceedings of the International Conference on New Interfaces for Musical Expression*, 16:61–64, 2016.

[2] G. Berry. The Esterel v5 Language Primer. 1997.

[3] U. Eco. *The Open Work*. 1989.

[4] M. Gimenes, P. Largeron, and E. Miranda. Frontiers: Expanding Musical Imagination With Audience Participation. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, volume 16, pages 350–354, 2016.

[5] A. Hindle. Swarmed: Captive portals, mobile devices, and audience participation in multi-user music performance. *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 174–179, 2013.

[6] M. Hirabayashi and K. Eshima. Sense of Space: The Audience Participation Music Performance with High-Frequency Sound ID. *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 58–60, 2015.

[7] J.-P. Lambert, S. Robaszkiewicz, and N. Schnell. Synchronisation for Distributed Audio Rendering over Heterogeneous Devices, in HTML5. *Proceedings of the 2nd Web Audio Conference (WAC-2016)*, pages 6–11, 2016.

[8] S. W. Lee, G. Essl, and Z. M. Mao. Distributing Mobile Music Applications for Audience Participation Using Mobile Ad-hoc Network (MANET). *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 533–536, 2014.

[9] J. Oh and G. Wang. Audience-participation techniques based on social mobile computing. *Proceedings of the International Computer Music Conference*, 2(August):665–672, 2011.

[10] M. Serrano and V. Prunet. A glimpse of Hopjs. *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming - ICFP 2016*, 2016.

[11] A. Stolfi et al. Open band: Audience Creative Participation Using Web Audio Synthesis. *Proceedings of Web Audio Conference*, 2017.

[12] B. Taylor. A History of the Audience as a Speaker Array. di(4):481–486, 2017.

[13] C. Vidal, G. Berry, and M. Serrano. Hiphop.js: a language to orchestrate web applications. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing, SAC 2018, Pau, France, April 09-13, 2018*, pages 2193–2195, 2018.

[14] N. Weitzner et al. massMobile – an Audience Participation Framework. *NIME 2012 Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 92–95, 2012.