OSSIA: towards a unified interface for scoring time and interaction

Jean-Michaël Celerier
Blue Yeti
jeanmichael@blueyeti.fr

Pascal Baltazar
L'Arboretum
pascal@baltazars.org

Clément Bossut
LaBRI
bossut.clement@gmail.com

Nicolas Vuaille
LaBRI
nicolas.vuaille@gmail.com

Jean-Michel Couturier
Blue Yeti
jmc@blueyeti.fr

Myriam Desainte-Catherine
LaBRI
myriam@labri.fr

ABSTRACT

The theory of interactive scores addresses the writing and execution of temporal constraints between musical objects, with the ability to describe the use of interactivity in the scores. In this paper, a notation for the use of conditional branching in interactive scores will be introduced. It is based on a high level formalism for the authoring of interactive scores developed during the course of the OSSIA research project. This formalism is meant to be at the same time easily manipulated by composers, and translatable to multiple formal methods used in interactive scores like Petri nets and timed automaton. An application programming interface that allows the interactive scores to be embedded in other software and the authoring software, I-SCORE, will be presented.

1. INTRODUCTION

This article will focus on a novel approach to represent and execute conditional branching in interactive scores. Interactive scores, as presented in [**?**], allow a composer to write musical scores in a hierarchical fashion and introduce interactivity by setting interaction points. This enables different executions of the same score to be performed, while maintaining a global consistency by the use of constraints on either the values of the controlled parameters, or the time when they must occur. This is notably achieved in the current version of the I-SCORE [1] software, presented in [**?**].

Previously, interactive scores did not offer the possibility to make elaborate choices in case of multiple distinct events occurring at the same time; the work presented here removes this limitation. Here, we will initially present the use cases for conditional branching, as well as several existing works of art which involve conditions. Then, we will introduce new graphical and formal semantics, researched during the course of the OSSIA project. Their goal is to allow composers to easily make use of conditional branching during the authoring of interactive scores. We will show the compliance with previous research on the same field, which allows for strong verification capabilities. We will conclude by presenting the software implementation of these formalisms in the upcoming version 0.3 of I-SCORE, which will be able to edit and play such scenarios in a collaborative way.

[1] http://i-score.org/

2. A CASE FOR CONDITIONAL INTERACTIVE SCORES

Even before the advent of computing, there was already a need to write scores containing informations of transport : in western sheet music, manifestations of this are the *D. S. Al Coda*, *D. S. Al Fine*, *Da Capo*, and repetition sign. There is however no choice left at the interpretation.

A case with more freedom for the performer is the *fermata*, which allows for the duration of a musical note to be chosen during the interpretation of the musical piece : the score moves from purely static to interactive, since there can be multiple interpretations of the lengths written in the sheet.

There is also the different case of improvisational parts where each musician has the freedom of his own choice during a few bars – or even a whole piece. In our case, the choices might involve multiple people at the same time (for instance multiple dancers each with his position mapped and used as a parameter), and lead to completely different results.

2.1 Conditional works of art

Some of the most interesting cases happen in more recent times, with the advent of composers trying to push the boundaries of the composition techniques. John Cage's *Two* (1987), is a suite of phrases augmented with flexible timing : "Each part has ten time brackets, nine which are flexible with respect to beginning and ending, and one, the eight, which is fixed. No sound is to be repeated within a bracket.". The brackets are of the form : $2'00'' \leftrightarrow 2'45''$ and are indicated at the top of each sequence.

Branching scores can be found in Boulez's *Third sonata for Piano* (195557) or in Boucourechliev's *Archipels* (1967-70) where the interpreter is left to decide which paths to follow at several points of bifurcation along the score. This principle is pushed even further in the polyvalent forms found in Stockhausen's *Klavierstücke XI* (1957) where different parts can be linked to each other to create a unique combi-

nation at each interpretation. Some of these compositions have already been implemented in computers, however it was generally done in a case-by-case basis, for instance using specific Max/MSP patches that are only suitable for a single composition. The use of patches to record and preserve complex interactive musical pieces is described in [**?**].

The scripting of interactive pieces can also be extended towards full audio-visual experiences, in the case of artistic installations, exhibitions and experimental video games. Multiple case studies of interactive installations involving conditional constraints (*Concert Prolongé*, *Mariona*, *The Priest*, *Le promeneur écoutant*) were conducted in the OSSIA project. *Concert Prolongé* (i.e. extended concert) offers an individual listening experience, controllable on a touchscreen where the user can choose between different "virtual rooms" and listen to a different musical piece in each room, while continuously moving his virtual listening point – thus making him aware of the (generally unnoticed) importance of the room acoustics in the listening experience. *Mariona* [**?**, section 7.5.3] is an interactive pedagogic installation relying on automatic choices made by the computer, in response to the users behaviours. This installation relies on a hierarchical scenarization, in order to coordinate its several competing subroutines. *The Priest* is an interactive system where a mapping occurs between the position of a person in a room, and the gaze of a virtual priest. *Le promeneur écoutant* [2] (i.e. the wandering listener) is a stand-alone interactive sound installation designed as a video game with different levels of exploration, mainly by auditory means.

In closing, interactive applications for exhibitions offer various situations in which conditional constraints are required, from touchscreen applications to full-fledged interactive installations. Several projects have been studied in the scope of OSSIA, in order to make the creation of new complex interactive applications more efficient by using the tools that are developed in this research project.

2.2 Existing notations for conditional and interactive scores

We chose to compare the existing notations in a scale that goes from purely textual like most programming environments, to purely graphic like traditional sheet music. Similarily, there are multiple ways to define interactivity and, consequently, multiple definitions of what is an interactive score.

The programmatic environments generally take a preexisting programming language, like LISP, and extend it with constructs useful for the description of music. This is the case with for instance Abjad [**?**], based on Python and Lilypond, a famous music typesetting software based on a TEX-like syntax. There are also programming languages more axed towards interpretation and execution of a given score, which can take the form of the program itself. This is the case with Csound and CommonMusic [**?**]. In general, programming languages of this kind offer a tremendous amount of flexibility in term of flow-control. However, they require additional knowledge for the composer to write scores with it.

[2] http://goo.gl/et4yPd

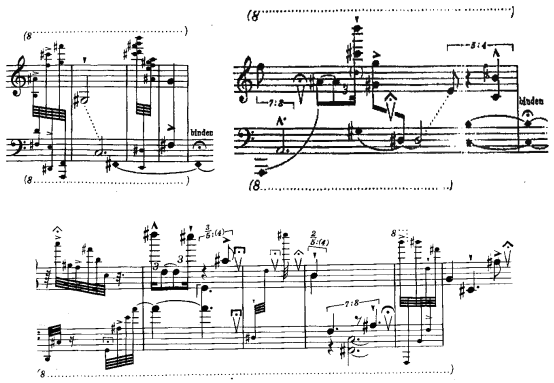


Figure 1: Fragments B2, C1, C3 of *Klavierstücke XI*, Karlheinz Stockhausen. [5]

The purely graphic environments allow compositions of scores without the need to type commands, and are much closer to traditional scores. For instance, multiple Max/MSP externals, Bach for Max/MSP [**?**], note~ [3], rs.delos [4] and MaxScore [**?**] allow to write notes in a piano roll, timeline, or sheet music from within Max. But they are geared towards traditional, linear music-making, even if one could build a non-linear interactive song by combining multiple instances, or sending messages to the externals from the outside.

Finally, there is a whole class of paradigms that sit between the two, with the well-known "patcher"-like languages: PureData, Max/MSP, OpenMusic [**?**], PWGL [**?**]. These software work in term of data-flow : the patch represents an invariant computation which processes control and/or audio data. In each case, it is possible to work purely graphically, and flow control is generally implemented as a block that acts on this data (`[expr]` in Pd/-Max or `[conditional]` and `[omif]` in OpenMusic, for instance). These software all allow to use a textual programming language to extend the capabilites or express some ideas more easily.

2.3 Goals

The examples presented in section 2.1 allow us to devise the specification for the kind of conditional capabilities that we want to allow composers to express in our notation system. Most examples studied here operate at a macroscopic level : the choices of the performer generally concerns sections, but at the phrase level, these are often traditional scores, as can be seen from an excerpt from *Klavierstücke XI* in fig. 1. However, the case of a single note which would last longer depending on a given condition can also happen.

The main problem is that there is generally no specific symbol to indicate the conditional execution; instead, the

[3] `http://www.noteformax.net`

[4] `http://arts.lu/roby/index.php/site/maxmsp/rs_delos`

[5] ©Copyright 1957 by Universal Edition (London) Ltd., London/UE 12654. Retrieved from [**?**]

explanation is part of the description of the musical piece. Hence, we have to devise a graphical notation simple enough and yet able to convey easily these different levels of conditions.

These conditions operate on a span of time, which can range from instantaneous, like in the Stockhausen piece, where the performer has to choose his next phrase at the end of the one he is currently playing, to indeterminate, in the case of a perpetual artistic installation waiting for the next visitor. A single symbol might then not be enough to convey in a readable fashion the whole meaning, and multiple symbols would be necessary to explain the articulation of the time in the musical piece.

Finally, an important requirement is to be able to study formal properties on the written score. The presence of conditional expressions means that there is some kind of flow control in the song. Like in a traditional computer software, we want to be able to verify that some properties will remain true for the score : for instance, again in the case of *Klavierstücke XI*, we would like to be able to specify : *at a given time, there cannot be two overlapping fragments*, and be informed if there might be a possible execution of the score that would lead to this case. This has practical implications especially when working with hardware, which can have hard requirements on the input data. This means that the notation will have to be grounded with solid formal semantics.

2.4 Formal semantics

The current work is based on previous work at the LaBRI by Jaime Arias, Mauricio Toro and Antoine Allombert, that attempt both to formalize the composition semantics and to provide ways for real-time performance of interactive scores. Our work is threefold: finding formal semantics adapted to complex conditional constraints; studying their execution; devising a consistent and simple graphical representation, in order to make the creation of interactive conditional scores as intuitive as possible.

The four following interactive scores formalisms were researched, in order to give a solid foundation, and enforce strong provability properties:

2.4.1 Petri nets

One of the most prominent ideas in the research, on which the current implementation of I-SCORE is based, is the use of Petri nets in order to model interactive scores, by focusing on agogic variations.

The followed methodology was to define basic nets for each Allen relationship [**?**], and then to apply a transformation algorithm, described in [**?**, section 9.2].

When a score is played, it is compiled by I-SCORE 0.2 (shown in fig. 2) into a Hierarchical Time Stream Petri Net (HTSPN) which is in turn executed using its well-known semantics.

Coloured Petri nets were also used to model complex data processing in interactive scores [**?**], in order to allow the description and execution of sound processes to occur directly in the score.

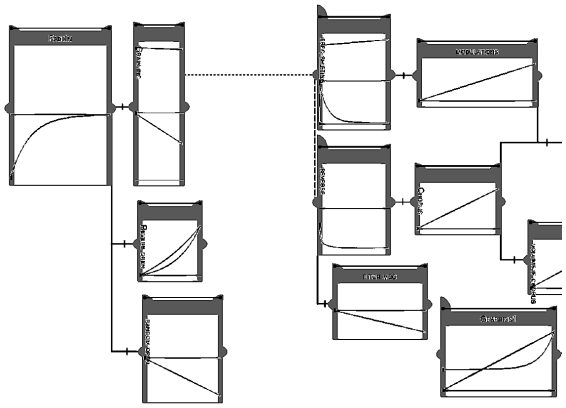


Figure 2: i-score 0.2. The model is based on a succession of boxes (containing curves and other boxes), and relationships between these boxes. A box can have trigger points at the beginning and the end; this introduces an unnecessary graphical coupling between the data and the conditions.

2.4.2 Temporal Concurrent Constraint Programming

Since the interactive scores can be expressed in terms of constraints (A *is after* B), one of the recurrent ideas for their formalisation was to use Non-deterministic Temporal Concurrent Constraint Programming (NTCC), since it allows constraint solving. This approach was studied by Antoine Allombert [**?**] and Mauricio Toro [**?**, **?**].

However, there are multiple problems, notably the impossibility to compute easily the duration of a rigid constraint, and the exponential growth of the computation time of constraint solving, which led to some latency in the implementation, making real-time operations impossible.

2.4.3 Reactive programming

Due to the static nature of models involving Petri nets and temporal constraints, a domain-specific language, REACTIVEIS [**?**], was conceived in order to give dynamic properties to interactive scores. An operational semantic is defined using the synchronous paradigm, to allow both static and dynamic analysis of the interactive scores. This also allows composers to easily describe parts of their score that cannot be efficiently represented in a visual manner.

2.4.4 Timed Automata

The current focus of the research is put upon the investigation of models for the formal semantics of conditional constraints in interactive scores.

This has been achieved using the extendend timed automata of UPPAAL. Timed Automata allow to describe both logical and temporal properties of interactive scores. Moreover, the shared variables provided by UPPAAL allows to model the conditionals. They are also used for hardware synthesis, in order to target Field-Programmable Gate Arrays (FPGAs) [**?**]. Real-time execution semantics is implemented with this method.

The problem of the implementation of loops is however still unresolved : it makes static analysis on the score harder, since we hurt the reachability problem.

3. THE OSSIA PARADIGM

3.1 Presentation

OSSIA (Open Scenario System for Interactive Applications) is a research project, presented in [**?**] and funded by the french agency for research (ANR). Its goal is to devise methods and tools to write and execute interactive scenarios. The two main objectives are to provide a formalisation for interactive scenarisation and seamless interoperability with the existing software and hardware. This paper will focus on the interactive scoring part, the interoperability being provided by the Jamoma Modular framework [**?**], which allows the use of multiple protocols, such as OSC or MIDI.

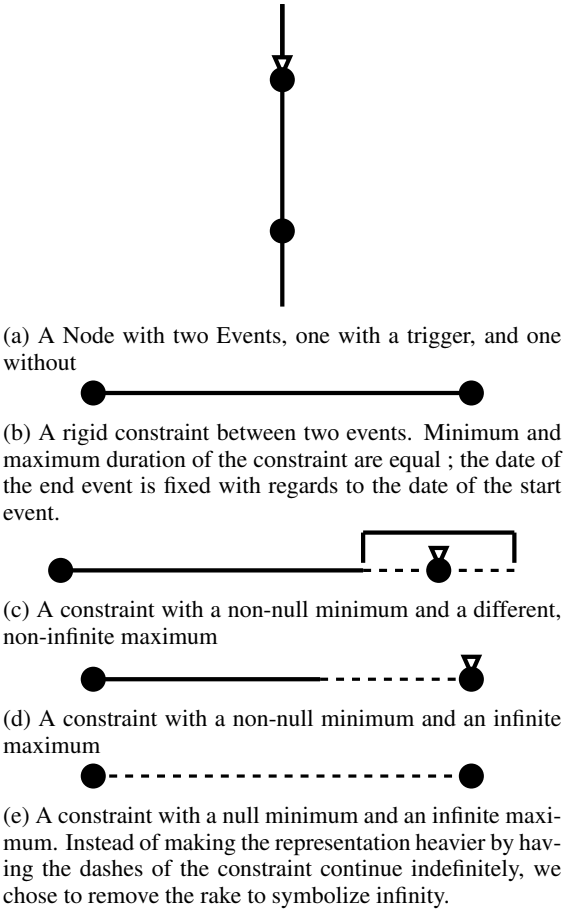
When comparing with the previous approaches for interactive scores (Acousmoscribe, Virage, i-score 0.2), the OSSIA project tries to follow a "users first" philosophy : the research work is shared and discussed with artists, developers, and scenographers from the musical and theater fields, and their use case serve as a basis for the focus of the research. They are in turn asked to try the software and discuss about the implementation.

For instance, in the previous studies of interactive scores, a mapping had to be done between the theoretical foundation (Petri nets, temporal constraints...) and the domain objects with which the composer had to interact. This has led to mismatches between the representation and the execution [**?**] of the score. The most prominent problem was the inability to express cleanly multiple synchronized conditions, and to route the time flow according to these conditions. The formalism also did not allow for boxes directly following each other in a continuous manner, and always required the existence of a relationship between them. Instead, in the OSSIA project, we tried to conceive high-level concepts that would allow a composer to easily write an interactive score, build a software over these concepts, and then implement them on the basis of the formalisms presented in part. 2.4.

The main concepts of interactive scores can be grouped in two categories: temporal elements and contents. The temporal elements (scenarios, instantaneous events, temporal constraints, conditional branching and hierarchy) allow to create the temporal and logical structure of the scenario, and the contents (states and processes) allow to give actual control over several kind of processes.

3.2 Temporal elements

In order to allow the composer to write interactive conditional scores, it is necessary to provide temporal constraints, to allow at least a partial ordering between the different parts of the score. This is done using four base elements : Node, Event, Constraint and Scenario. A *Node* (Time Node) represents a single point in time. An *Event* describes an instantaneous action. A *Constraint* describes the span of time between two given Events. Finally, the *Scenario* structures the other elements and checks that the temporal constraints are valid and meaningful.



(a) A Node with two Events, one with a trigger, and one without

(b) A rigid constraint between two events. Minimum and maximum duration of the constraint are equal ; the date of the end event is fixed with regards to the date of the start event.

(c) A constraint with a non-null minimum and a different, non-infinite maximum

(d) A constraint with a non-null minimum and an infinite maximum

(e) A constraint with a null minimum and an infinite maximum. Instead of making the representation heavier by having the dashes of the constraint continue indefinitely, we chose to remove the rake to symbolize infinity.

Figure 3: The OSSIA Graphical Formalism

3.2.1 Scenario

A Scenario is defined as an union of directed acyclic graphs. The vertices are Events and the edges are Constraints. The direction is the flow of time. It allows to organize the other base elements in time.

Scenarios follow these basic rules:

- A Scenario begins with a Node
- There can be multiple Events explicitly synchronized by a single Node
- A Constraint is always started by an Event and finished by another, distinct Event

Events and Constraints are chained sequentially. Multiple Constraints can span from a single Event and finish on a single Event, as shown in fig. 7. The operational semantics of these cases will be described later. This allows different processes to start and/or stop in a synchronized manner.

3.2.2 Events and Nodes

An Event allows to describe precisely a part of what will happen in a specific, instantaneous point in the execution of the score. It is the basic element, to which are further attached Constraints.

Events can be explicitly synchronized using Nodes. This means that when an Event is triggered, all the other Events on the same Node are also evaluated and instantaneously

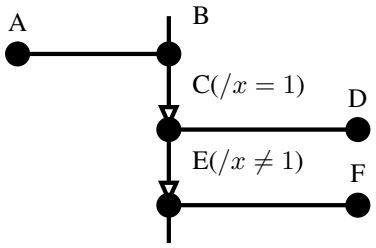


Figure 4: Implementation of temporal branching. A, B, C, D, E, F are Events. B, C, E are on the same Node. C contains the condition $/x = 1$ and E contains $!(/x = 1)$ in order to have an *if - then - else* mechanism. C and E are evaluated when the constraint between A and B has ended.

triggered (or discarded if their Condition is not met, see section 3.3.1).

3.2.3 Constraints

A Constraint represents a span of time. Due to the interactive nature of the proposed paradigm, the span can change at execution time, like a *fermata*. When the author wants to allow a Constraint to have a variable duration, he renders it flexible. This means that the end of the Constraint depends on the Condition of its final Event.

A Constraint can be activated or deactivated: if it is deactivated, it will not count for the determination of the execution span of its end event.

The graphical representation of a Constraint can change according to its minimum and maximum duration. The minimum m's range is $[0; +\infty]$, and the maximum M's range is $[m; +\infty] \setminus \{0\}$. In the user interface (introduced in section 4), the duration is directly linked to the horizontal size and is visible on a ruler.

3.2.4 Graphical formalism

The graphical formalism for these elements is presented in fig. 3.

The Node is a vertical line. An Event is a dot on a Node. If there is a trigger on the Event, a small arrow indicates it. The colour of the arrow can change at run-time to indicate the current state of the trigger.

The Constraint is an horizontal line that represents a span of time, like a timeline. If the constraint is flexible, the flexible part is indicated by dashes and a rake. When there is no maximum to the constraint, there is no rake.

3.3 Operational semantics

3.3.1 Conditions

Each Event carries a condition of execution, and a maximum range of time for its evaluation. The effective range of time for execution is computed by constraint-solving algorithms. For the Event to enter its execution range, all the Constraints that finish on this Event must be between their minimal and maximal duration.

An Event is executed as soon as its condition evaluates to True in its execution range. As a result, the Constraints that

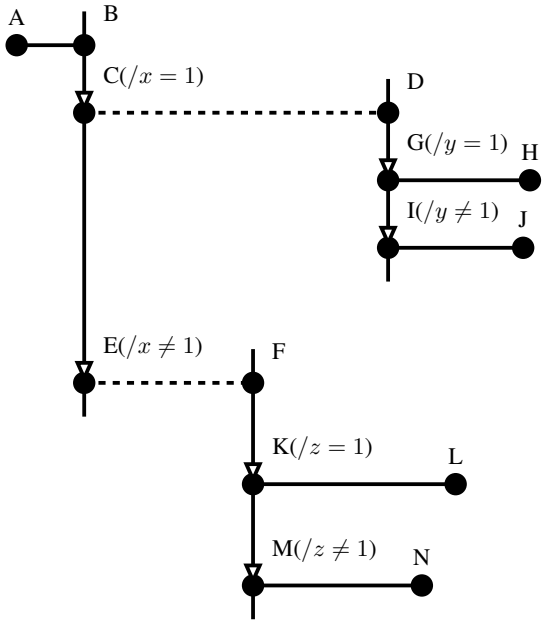


Figure 5: Nested *if - then - else* using flexible constraints

follow this Event are started, and the messages that might be stored in the Event are sent. Otherwise, the Event is discarded and all the following Constraints are deactivated.

There is usually a default condition which is "all the constraints that explicitly finish on the event have ended". This default condition can be replaced or extended. For instance, there can be checks on the arrival of a specific network message, or checks on a remote or local address's value with a specific expression, with the following syntax:

- For parameters that can have a value, there can be comparisons between the values. For instance:

```
/some/parameter > 35 &&
(   /other/parameter != "a string"
 || /last/parameter == true)
```

- Value-less parameters (akin to *bangs* in PureData) can also be used as triggers for the evaluation of expressions. In this mode, logical operators have a different meaning. For instance:
  `/some/bang && !/another/bang`
  will trigger if :
  - `/some/bang` is received, and
  - `/another/bang` is not received within the synchronization interval.

- This is not to be confused with the comparison with boolean values :

```
/a/val == true &&
/another/val == false
```

  which will trigger when the parameters will both be set (not necessarily at the same time) to the required values.

Higher-level operations, like a mouse click on a GUI can then be translated in conditions on Events, in order to bring rich interaction capabilities to the software dedicated to the execution of the scores.

3.3.2 Conditional branching

Branching occurs when, at a single point in the score, two different executions can happen, which leads the scenario to distinct states. For example, the classic *if - then - else* construct can be implemented by having two Events with opposite conditions, as shown in fig. 4. It is also possible to have other cases : for instance, there could be a set of conditions that would lead to either both constraint, or no constraint executed. It is also possible to have multiple constraints with the same condition. Figure 5 presents a method to instantaneously nest conditions, using a flexible constraint with a minimal duration of zero. These patterns can be used as is. However, the authoring software should provide graphical ways to simplify these common cases, for instance by not showing the duplicated conditions. This is not yet done in the current development version of i-score, which allows its users to author scores using the raw formalism presented here.

Convergence occurs when we want to synchronize parts of a scenario that branched previously. The constraint solver ensures that the durations are coherent during the authoring of the score. The execution date of the Node for which a convergence happens will necessarily be after the minimum of each converging Constraint.

3.3.3 Execution of a Node

In this part, we will first present a high-level algorithm that explains the general order of execution of a Node. Then, we will study a particular Node and see how the algorithm translates into a Petri net that can be executed for this particular Node.

As we said before in section 3.2.2, Events are attached to a Node. They are ordered and will be evaluated one after another according to the algorithm given in fig. 6.

The software guarantees that at a high level, if the composer sets an order, the messages will be sent in this order. The only delay will be the one induced by the ordering of instructions in the CPU.

However, it is necessary to be aware that the protocols used underneath, like UDP which is commonly used in OSC protocol implementations, might not always have such strong guarantees. For this reason, and also because it is not possible to expect network messages to arrive exactly at the same time, the author can specify a synchronization time on a Node: it can wait for a brief time after the triggering of another Event in the same Node, in order to let some time for messages to be received and change the result of conditional choices. However we don't have yet a way to represent this graphically; the idea of a bolder time node was proposed. The synchronization time can also express the will to synchronize events at a lower rate, like two people clapping hands at the same time for example.

Another possibility for the ordering would be to run the Event whose condition did trigger the time node first, and then run the others. This should be a choice left at the discretion of the score writer.

```
Require: We enter the evaluation range for a Node n
  if all(n.eventList(), Event::emptyCondition) then
      repeat
          wait()
      until n.DefaultDate
      for Event e in n.eventList() do
          e.run()
      end for
  else
      repeat
          if any(n.eventList(), Event::isReady) then
              nodeWasRun ← true
              for Event e in TimeNode.eventList() do
                  if validate(e.condition()) then
                      e.run()
                  else
                      e.disable()
                  end if
              end for
          end if
      until nodeWasRun
  end if
```

Figure 6: Execution algorithm for a Node

Figure 7 presents the different branching and converging cases that may occur, all mixed in a single Node.

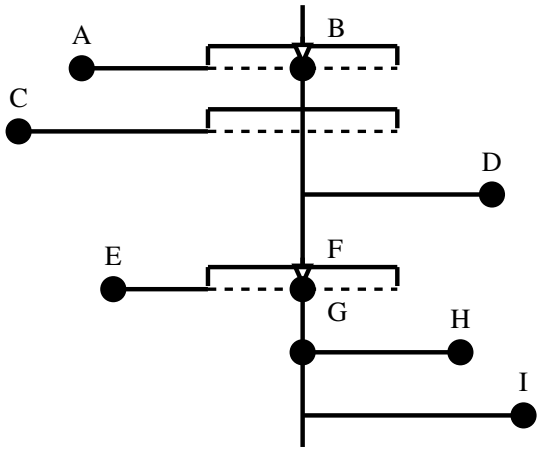


Figure 7: A complete example of Node in the OSSIA graphical formalism. Two constraints converge on the Event B, and three constraints branch from the Node that synchronizes B, F, G. The durations are already processed by the constraint solver.

3.4 Contents

An Event may contain a State, which contains messages, and can itself hierarchically contain other States. At a conceptual level, for the composer, a State generally represents a change of state in a remote or local device, i.e. a discontinuity.

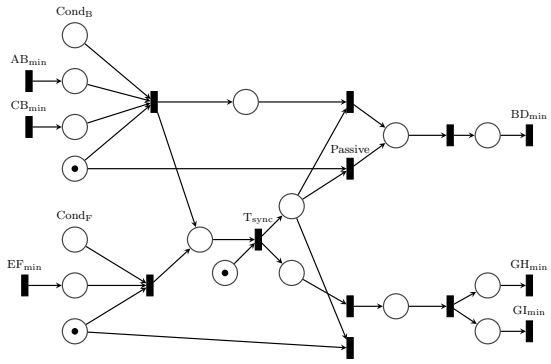


Figure 8: Petri net for the Node of fig. 7.

The previous Constraints's minimum duration have to elapse in $\mathrm{AB_{min}}$, $\mathrm{CB_{min}}$, $\mathrm{EF_{min}}$. When a Condition is validated, a token is put in the respective place ($\mathrm{Cond_B}$ or $\mathrm{Cond_F}$); this triggers the evaluation of the whole Node, unless it was already triggered. The central transition $\mathrm{T_{sync}}$ introduces a small synchronization delay to allow other conditions to validate if necessary. When an Event's condition is not validated, a passive token is instead sent to the following Constraints, which will not trigger the execution of any process or state, and will simply allow the scenario to keep going after a failed condition. This is achieved by the Passive transition.

A Constraint also acts as a container: during its execution, several Processes can be executed in parallel.

The two main Processes are:

- The Curve, which allows to send interpolated data in any protocol available to the software.
- The Scenario, which allows hierarchy to happen seamlessly: a constraint can contain a scenario, which can in turn contain other Constraints.

There can be two possible executions for processes: they can do something on each tick of a scheduler; or they can send start and stop signals and behave however they want in-between.

Processes can share data with the Events at the beginning and the end of the parent Constraint, by putting them in specific States.

The API provides ways for somebody to implement his own processes and use them afterwards in scores.

3.5 Usage as an API

One of the goals of this high-level paradigm is to allow at the same time, a simple mapping with graphical elements, clear semantics of execution, and simple translations with the proven semantics that were studied before, like Petri nets and Timed Automata. In this way, a composer could describe his score which could then be translated into a variety of formats that allow for static and dynamic analysis.

This is achieved by writing a C++ API, that allows for multiple implementations and the use generic programming. It is accessible in `https://github.com/OSSIA/API`. It consitutes a kind of domain-specific language, with the elements that were talked about earlier.

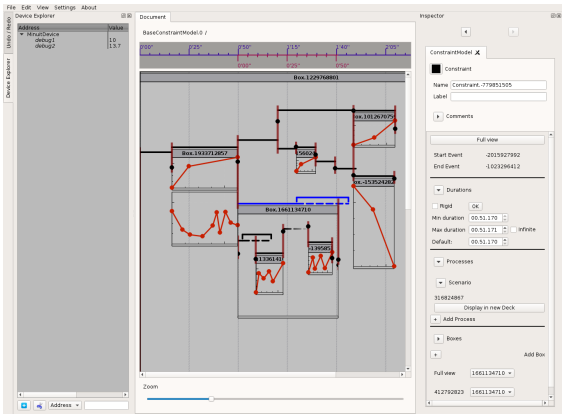


Figure 9: An example of score in i-score 0.3. Not all the graphical features presented here are already implemented.

3.6 Implementation in terms of Petri nets

In order to maintain cohesiveness with the previous works on the field, we chose to represent the temporal logic of our formalism in terms of Petri nets. The span of time of the constraints is represented as a transition, and states are emitted when a token enters a state-containing place. These places are not represented here because there can be multiple cases according to the requirements of the writer of the score: it would for instance be possible to send the last state either after the default duration of the constraint, or as soon or late as the condition is validated. It is also possible to add places and transitions in order to have a specific behaviour occurs when the maximum duration of a constraint elapses without triggering.

Figure 8 represents the translation of the Node shown in fig. 7 into Petri nets. We refer to Constraints by the name of their start and end Events.

4. I-SCORE: TEMPORAL AND LOGICAL USER INTERFACES

The API talked about in section 3.5 is being used as the basis of different projects tied to the interactive scores. The dependency graph is shown in fig. 10.

The different sub-projects are:

- I-SCORE, a graphical editor and player for the interactive scores. It solves the problem of the display, edition and interaction with simultaneous elements of the interactive scores. Its current development version (0.3) is available at (`https://github.com/OSSIA/`
- J.SCORE and I-SCORE-CMD: two players for the scores produced in i-score. The first is an external for Max, the second is a standalone command-line executable. Their current version is available in the repository of the Jamoma project (`https://github.com/Jamoma/Jamo`

I-SCORE also exposes its own dynamic device in order to provide some kind of external control at run-time. For example, the conditions could be changed prior to their evaluation, in order to set them in advance at true or false according to events that might have occurred previously during the execution of the score.

The current version (0.2) of I-SCORE, shown in fig. 2, relies on the idea that relations are used to separate boxes. At the time of writing, the upcoming version (0.3, shown in fig. 9) is able to create constraints, events, and nodes using the graphical formalism that was presented in this article, and can play and export these scores so that they can be played on the other software of the suite. It also allows a primitive form of collaborative edition of scores on a local network, and the authoring and execution of distributed scores is currently being studied. Distributed scores would allow for instance to have a part of a score run on a computer, and another part run on another computer.

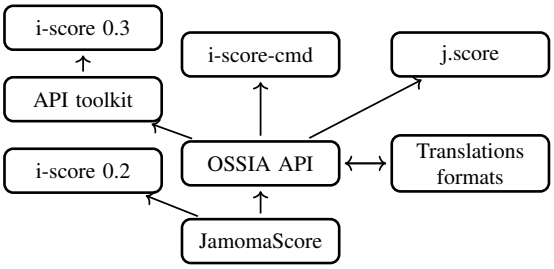


Figure 10: Diagram of the different components of the OSSIA project

5. CONCLUSION

We presented in this article a new interactive score semantic that allows the conception and execution of conditional scores. This semantic is thought of as a mapping into well-known formal models, such as Petri nets, Timed Automata, and Reactive languages: it is meant to be easily understandable and usable for the composer.

However, in order to achieve more expressive power, we still need to find a way to implement loops. Two approaches are currently being studied: one using a Loop process, and another using a concept of `goto`; once one is chosen, we will try to find a relevant graphical element to present it.

Furthermore, there could be some interest in the specification and implementation of variables, which could alleviate the need for an adjacent software like Max/MSP to perform complex logical computations. This would maybe pave the way towards a time-oriented Turing complete programming language, with a simple graphical representation which would allow composers to write complex scores in an understandable way. Another track is the implementation of an audio engine, for instance by embedding FaUST[6], in order to be able to produce sound directly from i-score. The relevant parameters would then be exposed and controlled within i-score.

The next step for the graphical formalism is to make usability studies in order to find the most convincing interactions in the authoring software for the composers.

Acknowledgements

The authors would like to thank Théo de la Hogue as well as Jaime Arias for their contributions and insightful comments during the writing of this article. The OSSIA project was funded by the french agency for research (ANR) under the reference ANR-12-CORD-0024 for the 2012-2015 period.

[6] `http://faust.grame.fr/`