i-score, an Interactive Sequencer for the Intermedia Arts

Pascal Baltazar
L'Arboretum
pascal@baltazars.org

Théo de la Hogue
GMEA
theod@gmea.net

Myriam Desainte-Catherine
LaBRI
myriam@labri.fr

ABSTRACT

The i-score intermedia sequencer allows to design complex interactive scenarios by coordinating heterogeneous and distributed media systems. Based on software frameworks issued by several long-term research iniatives, this application does not produce any media of its own, but rather controls other environments parameters, by creating snapshots and automations, and organizing them in time in a multi-linear way. In this article, we describe the main features of the software, in order to guide the workshop participants towards actual creation of interactive scenarios by controlling their favorite software environment.

1. INTRODUCTION

Recent technological developments in real-time media computing, as well as aesthetic evolutions of the contemporary arts converge in the generalization of distributed technical setups for media-management. Even though, some of these setups include systems for scripting scenarios in time, it is very often complicated to coordinate these heterogeneous systems with the ease and flexibility that the artistic process requires. Furthermore, while the temporal scripting in the context of fixed-time media (such as in DAWs) has now come to a mature state with well-defined and stable user-interaction paradigms, the introduction of interactivity renders these paradigms almost inoperative. Time-scripting in interactive works (such as performances or interactive installations) is typically managed with cues, as points of synchronisation throughout a scenario, with very few possibilities for designing evolutions of expressive parameters in time, compared to what automations in fixed-time media softwares do. The Ableton Live[1] or Qlab[2] applications offer such cue management with some capabilities for automation editing. More experimental sequencers such as Duration[3] and Vezèr[4] allow even more complex automation capabilities. However, cues and automations are managed in these softwares as a linear list of events to be successively triggered, without further temporal organization. Also, even if they are capable of controlling remote applications and devices through the OpenSoundControl protocol, the manual management of OSC addresses is often a tedious process involving a lot of typing.

The *i-score* interactive sequencer proposes solutions to these problems by offering an organized way to structure events in time, while keeping degrees of liberty for interactivity. It is also built upon the concept of networked devices, presented as a visual interactive tree structure in order to intuitively control them in the scenario.

The workshop will focus on hands-on practice: participants will use i-score with their favorite multimedia environment and will be guided, following the article's structure, towards the actual creation of interactive scenarios.

[1] https://www.ableton.com
[2] http://figure53.com/qlab/
[3] http://www.duration.cc
[4] http://www.vezerapp.hu

2. PROJECT HISTORY

i-score[5] is the result of several research initiatives along the last fifteen years. The paradigm of using temporal relations in order to keep temporal consistency in a scenario was introduced in the *Boxes*[1] spectral composition software. Further implementations of this paradigm were carried on in *Boxes 2*, *Acousmoscribe*[2] and *Iscore* [3], an implementation for the OpenMusic composition environment. Interactivity was introduced during the Virage research project, under the concept of trigger points, and it was implemented in the *libIscore* library. This library was used as an engine for both *Acousmoscribe 2* and *Virage sequencer*[4], the latter addressing specific problems of the performing arts, in particular those relating to the concept of flexible time, as pointed out in [5]. As both of these softwares used similar user-interaction concepts and were based on the same engine (*libIscore*), the decision was taken to merge both development efforts into one and only software. This was achieved by injecting concepts and designs implemented in the *Virage* sequencer into the *Acousmoscribe 2* code, in order to create the *i-score* sequencer. Many efforts on user-friendliness and ergonomics were made in the process, in order to turn proof-of-concept prototypes into actually usable professional software. Further development of the system was carried on in the frame of the OSSIA[6] project. In the first place, the *libIscore* library was refactored on the basis of the *Jamoma framework*[6] in order to create the *Score* library, which will be further described in Section 4. *i-score* was then refactored

[5] http://i-score.org/
[6] OSSIA is a collaborative research project financed by the French National Agency for Research. It aims at the formalization of logico-temporal constraints for hierarchical, non-linear and multi-user interactive scenarios, for the contexts of video-games and museal installations

upon this library, and extended, along with *Score*, in order to deal with non-linear temporal structures through conditionnal branching. This effort produced the version 0.2 of *i-score* which is discussed in this article and that will be used for the proposed workshop.

3. USER INTERFACE

The *i-score* sequencer is an open source [7] standalone application developed in C++ with the Qt [8] cross-platform framework. It is currently available for the Mac, Linux and Android platforms. As mentionned above, and further described in section 4, *i-score* is based on the *Score* and *Modular* frameworks, which provide the majority of the services required for temporal and content management.

In contrast with most sequencers, *i-score* does not produce any media of its own, but rather controls parameters of remote applications, by creating snapshots and automations, and organizing them in time in a multi-linear way. Its design is led with the emphasis on several concerns:

Speed and responsiveness in the edition of scenarios: In order to keep the creative workflow as fluid as possible, the interface design allows the user to create states or evolutions (sometimes composed of dozens of parameters) in a matter of seconds. The user can do so by creating a temporal object, associating a set of parameters to it and creating snapshots or interpolations of theirs states in a few clicks, without having to type or manage any text-based content.

Evolutivity is then another important matter, as the user must be able to precisely and thoroughly refine its scenarios, in particular by precisely designing automations of parameters. Temporal structure must also be refineable in order to accomodate with the evolutions of the scenario over time, in particular if it interacts with human performers.

Clarity and intuitivity are key features in the overall design of the interface, and are kept in mind whenever implementing new features, in order to keep the whole user interface consistant and its look-and-feel as clear and minimalist as possible.

3.1 General presentation

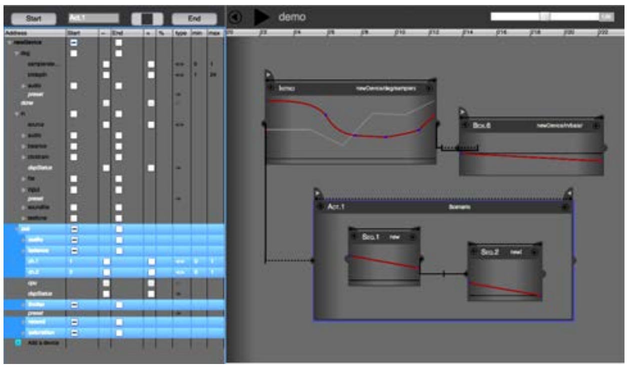
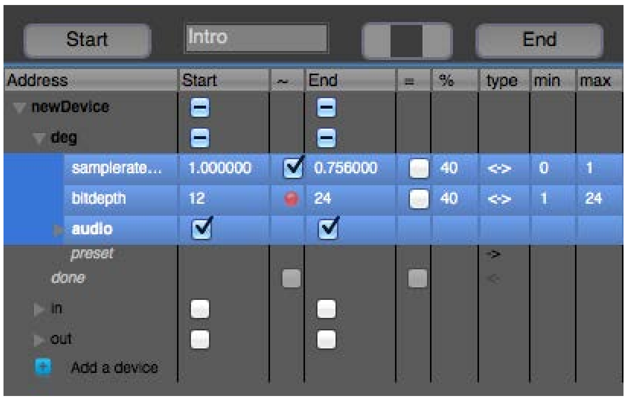


Figure 1. Overview of the i-score sequencer user interface.

i-score's user interface is organized in two sections: a tree-based interface on the left, for managing multiple remote devices' parameters; and a timeline-based interface on the right, allowing to arrange temporal objects in time.

3.2 Content management

As mentioned above, *i-score* controls other environments' parameters. In order for the user to do so, the first step is to declare such an environment, or device in i-score's parlance, by clicking on the [+] button labeled "Add a device" at the bottom of the inspector. A pop-up window will then open, where network parameters of the device can be set up, such as IP, port and protocol. The device, in Minuit, or its descriptive file in OSC, will then be scanned, in order to display its structure as a tree in the inspector.



Address	Start	~	End	=	%	type	min	max
newDevice	⊟		⊟					
deg	⊟		⊟					
samplerate...	1.000000	✓	0.756000	☐	40	<>	0	1
bitdepth	12	●	24	☐	40	<>	1	24
audio	✓		✓					
preset						->		
done		☐		☐		<-		
in	☐		☐					
out	☐		☐					
Add a device								

Figure 2. The devices inspector

The example provided here uses the Jamoma 0.6 for Max patcher described in [7], connected through the Minuit protocol [9]. The elements displayed in the inspector, notably the selection of nodes and the values associated with them, are contextual to the currently selected box. For instance Figure 2 displays the actual content of the "Intro" box of Figure 1. The lines highlighted in blue are those whose parameters are contained in the box's events and/or automations. For instance, the start event of the box contains the values `1.` for the samplerateRatio parameter, and `12` for the bitdepth parameter. Other parameter values under the audio node are also contained in the start and end events (as the checkbox indicates), but they are hidden by the folding of the structure under this node. Parameter values can be entered manually by typing them in the appropriate cells. They can also be queried programmatically from the remote device, by selecting a set of nodes and pressing the Start or End buttons on the top of the inspector. When nodes with descendants are selected, the algorithm will recursively query all descendants for values, even when these are hidden from the inspector view by folding them out.

When looking at Figure 1, we can notice that the "Intro" box contains two breakpoint functions, reflected in the inspector by the checkboxes in the column between the start and end values. These automations can be created by sev-

[7] available at https://github.com/i-score/i-score
[8] http://qt-project.org/

[9] Minuit is a query system based on OSC. Actual connection between i-score and Jamoma is achieved through an interface patcher available on https://github.com/Minuit/minuit

eral ways: When differing numerical values of a parameter are assigned to the start and end events of a box, a linear interpolation is automatically created in the interval, which the user can disable by unchecking the automation checkbox. An automation can be created by checking this same checkbox, even if no value is present in the box's events: these will then be automatically filled with the minimum and maximum values of the parameter, displayed in the rightmost column of the inspector. Finally, automations can be recorded live by listening for a remote parameter. This function is enabled during the edition by Command (for Mac) or Control (for Linux) clicking on the checkbox. Then, during the execution, *i-score* will listen for the changes in the selected value(s) and it will record them as an automation. Once created, the automations can be edited in the boxes, by choosing their parameter address in the top-right menu. Points in the breakpoint functions can be moved, added or deleted, and curve coefficients can be edited by shift-clicking on the last point of a segment and dragging up or down. Functions can also be free-hand drawn in the box by holding the Command/Ctrl key.

3.3 Temporal arrangement of objects

The arrangement section is based on the timeline paradigm, as in most media-editing softwares. Time is represented from left to right on the horizontal axis. The vertical axis has no predetermined meaning.

Temporal objects (also called *boxes*) can be placed on the timeline, moved and resized. They contain a starting event and an ending event, in which parameter states can be stored. They can also contain temporal processes, such as automations and sub-scenarios.

By default, when creating a box, it will be played at the date which is represented on the timeline. It is then possible to create fixed-time scenarios by simply drawing boxes on the timeline. Execution can be launched by pressing the play button on the top and stopped by pressing the stop button that then replaces it, or alternatively toggled by repeatedly pressing the space bar. Execution can be started from any date by clicking on the time ruler on the top and pressing play. In this case, all previous content of the scenario will be compiled and dumped just before actually starting the execution, in order to start from a similar state of the system as if the scenario had been played normally from its beginning. During execution, the global speed (or clock-rate) of the scenario can be changed on the fly, by dragging the slider on the top, or by sending a numerical value to its associated OSC address.

It is also possible to detach the execution of a particular box from the fixed flow of time, thus rendering it interactive, by adding *trigger points* at its beginning or end. This is easily done by clicking on the small oval quarter on the top corners of the box, which will create triangles representing the trigger points. At execution time, these trigger points, and thus the execution of the associated boxes, can be triggered manually by clicking onto them, or by hitting the right arrow or one of the number keys. Available trigger points will be associated with number keys based on their respective dates: the earlier one will be triggerable with the 0 key, the next one with the 1 key, and so on. Trigger points can also be triggered remotely by listening to a remote or local address and/or evaluating expressions based on values of remote or local parameters.

Relative positions of boxes can be organized in time and maintained by temporal relations. This is useful for fixed-time scenarios (i.e. without trigger points) in order to maintain the temporal consistency of a scenario while moving parts of it or when inserting new objects inbetween existing ones. In the absence of trigger points, relations are considered rigid, and represent a constant duration, that can be modified by the user during the edition by click-dragging them horizontally. At execution time, these rigid relations' durations will always be respected.

When the end of a relation is connected to an interactive event (i.e. associated with a trigger point), it becomes flexible. This is represented by a discontinuous line. Even if it is represented with a certain length on the timeline, such a flexible relation has no predetermined duration, as its end will be interactively triggered during the execution. It is however possible to restrict this duration to a bound interval. This is achieved by setting minimum and/or a maximum bounds to the relation. The minimum bound is always accessible for edition: it is placed by default at the very beginning of the relation and can be moved by dragging it to the desired duration. During execution, this minimum bound will prevent the trigger point to be triggered before the chosen duration. In order to make the maximum bound appear, the user needs to double-click on the relation, and is then able to drag it to the desired duration. At execution time, conversely to the minimum bound, if the trigger point is not yet triggered when the maximum duration is elapsed, it will then be automatically triggered.

Alternatively to the temporal relations, events can be linked by logical relations. By pressing the alt key while creating a relation, the user is able to create a conditionnal relation between two or more events, which will also create a trigger point on each of the related events. At execution-time, all expressions of the conditionnaly-related events will be evaluated simultaneously. Those being true will be triggered, the others will be disposed of, as well as their successors, thus creating diverging branches in the scenario.

4. A SHAREABLE ENGINE

The *i-score* engine is based on two C++ libraries: *Score*[10] and *Modular*[11], which respectively offer services for temporal organization and devices management.

4.1 Score

The *Score* library is organized in four main classes : *TimeEvent*, *TimeCondition*, *TimeProcess* and *TimeContainer*. The *TimeEvent* class has an indicative date, a status (waiting, pending, happened, disposed) and can refer to states to recall when it is actived. The *TimeCondition* class takes care of i-score's trigger points with an event table classified by *Expressions*, which are evaluated in order to activate

[10] https://github.com/OSSIA/Score

[11] https://github.com/jamoma/JamomaCore/tree/master/Modular

each event or dispose of them. The *TimeProcess* class defines an interface to compile the actions that are performed on the start, during execution and at the end of any temporal content Currently, there are two *TimeProcess* plug-ins available: *Interval* and *Automation*. The development of a mapper and a generator plug-ins is planned and it is also possible to third-parties to create dedicated plug-ins for media management. The *TimeContainer* class inherits from *TimeProcess*. It defines an interface to add, delete and manage actions on *TimeEvents*, *TimeConditions* and *Time-Processes* list. For the time being, the only *TimeContainer* plug-in is *Scenario*. The *Scenario* plug-in implements a constraint manager to keep precedence relations consistant between events during edition (based on the GeCode library) and compiles into a Petri net in order to check execution validity.

4.2 Modular

The *Modular* library helps to develop Model-View-Controler oriented applications, in order to represent their services as a tree structure and to expose them remotely. *i-score* uses this library to manage communication protocols via a plug-in interface, which considers that any protocol can be reduced to some basic operations like listen, get, set or explore. For the time being two plug-in are available for the OSC and Minuit protocols.

5. PERSPECTIVES AND FUTURE WORK

The use of *i-score* still raises many real-life problems and the integration of the new *Score* library also introduces some new ways of managing time.

First of all, i-score's current interface is based on a duration management approach where each box has its own start and end and where boxes have to be preceded or followed by relations. This workflow is problematic when a user simply wants to edit a set of cue events and create transition between them. With the use of *Score* it is now possible to edit single events, while intervals and an automation are now seen as specific processes that can share their events. This new architecture also allows to have several processes attached to the same start and end events. Since Scenario is also a process, the hierachy feature is now fully functionnal.

To accomodate with these various changes, a team of developers is currently developing a new version of the graphical interface from scratch, called *i-score* 0.3. This new interface will introduce several graphical conveniences, allow to navigate between hierarchical levels and to vertically pile different processes of the same box as storeys, in order to help editing them in parallel. A track system will also be created to split the timeline and explorer interfaces vertically in order to break down large sets of parameters into smaller parts, thus increasing readability.

Finally, as the *Score* library can now be compiled for the linux environnement, it will be possible to embed it into units like the BeagleBoard or Raspberry Pi. This leverages possibilities for the edition of distributed scenarios inside the same *i-score* interface and for the synchronization of their execution. This also allows the development of distributed setups of many media players, which can be useful for many applications in the museographic domain.

6. CONCLUSIONS

The development of the *i-score* sequencer has emerged from joint concerns and long-term collaborations between artists and researchers. The application has been used in a professional production context, and now requires to increase its user base in order to accomodate with more use cases and situations. Feedback and comments from the workshop participants will then be a valuable input for future development and research around this project.

Acknowledgments

Development of i-score has been supported by LaBRI, l'Arboretum and GMEA, with funding from the 2012-2015 OSSIA ANR-12-CORD-0024 research project. The authors would like to thank developpers Nicolas Hincker, Jaime Chao and Clément Bossut for investing a lot of energy in the development and for providing some content for this article, and Jaime Arias Almeida for proof-reading.

7. REFERENCES

[1] A. Beurivè and M. Desainte-catherine, "Representing musical hierarchies with constraints," in *Proceedings of CP'01, Musical Constraints Workshop, Paphos, Cyprus*, 2001.

[2] M. Desainte-Catherine and J.-L. D. Santo, "L'acousmoscribe, un éditeur de partitions acousmatiques," in *Proceedings of Electroacoustic Music Studies Buenos 09 Aires*, 2009.

[3] A. Allombert, M. Desainte-Catherine, and G. Assayag, "Iscore: A system for writing interaction," in *Proceedings of the 3rd International Conference on Digital Interactive Media in Entertainment and Arts*, 2008.

[4] A. Allombert, R. Marczak, M. Desainte-Catherine, P. Baltazar, and L. Garnier, "Virage : Designing an interactive intermedia sequencer from users requirements and theoretical background,," in *Proc. of the International Computer Music Conference*, 2010.

[5] P. Baltazar, A. Allombert, R. Marczak, J.-M. Couturier, M. Roy, A. Sèdes, and M. Desainte-Catherine, "Virage : Une réflexion pluridisciplinaire autour du temps dans la création numérique," in *Actes des 14 d'Informatique Musicale, Grenoble*, 2009.

[6] T. De La Hogue, J. Rabin, and L. Garnier, "Jamoma Modular: une librairie C++ dediee au developpement d'applications modulaires pour la creation," in *Proc. of the 17es Journées d'Informatique Musicale*, Saint-Etienne, France, 2011.

[7] T. Lossius, T. de la Hogue, P. Baltazar, T. Place, N. Wolek, and J. Rabin, "Model-view-controller separation in max and jamoma," in *Submitted to the joint ICMC/SMC conference, Athens, Greece,*, 2014.