Graphical Temporal Structured Programming for Interactive Music

Jean-Michaël Celerier
LaBRI, Blue Yeti
Univ. Bordeaux, LaBRI, UMR 5800,
F-33400 Talence, France.
Blue Yeti, F-17110 France.
jcelerie@labri.fr

Myriam Desainte-Catherine
LaBRI, CNRS
Univ. Bordeaux, LaBRI, UMR 5800,
F-33400 Talence, France.
CNRS, LaBRI, UMR 5800,
F-33400 Talence, France.
INRIA, F-33400 Talence, France.
myriam@labri.fr

Jean-Michel Couturier
Blue Yeti, F-17110 France.
jmc@blueyeti.fr

ABSTRACT

The development and authoring of interactive music or applications, such as user interfaces for arts & exhibitions has traditionally been done with tools that pertain to two broad metaphors. Cue-based environments work by making groups of parameters and sending them to remote devices, while more interactive applications are generally written in generic art-oriented programming environments, such as Max/MSP, Processing or openFrameworks. In this paper, we present the current version of the i-score sequencer. It is an extensive graphical software that bridges the gap between time-based, logic-based and flow-based interactive application authoring tools. Built upon a few simple and novel primitives that give to the composer the expressive power of structured programming, i-score provides a time line adapted to the notation of parameter-oriented interactive music, and allows temporal scripting using JavaScript. We present the usage of these primitives, as well as an i-score example of work inspired from music based on polyvalent structure.

1 Introduction

This paper outlines the new capabilities in the current iteration of i-score, a free and open-source interactive scoring sequencer. It is targeted towards the composition of scores with an interactivity component, that is, scores meant to be performed while maintaining an ordering or structure of the work either at the micro or macro levels. It is not restricted to musical composition but can control any kind of multi-media work.

We first expose briefly the main ideas behind interactive scores, and explain how i-score can be used as a language of the structured programming language family, targeted towards temporal compositions, in a visual time-line interface.

In previous research[1] interactive triggers were exhibited as a tool for a musician to interact with the computer following a pre-established score. Here, we show that with the introduction of loops, and the capacity to perform computations on variables in a score, interactive triggers can be used as a powerful flow control tool, which allows to express event-driven constructs, and build a notion similar to traditional programming languages procedures.

We conclude by exhibiting an i-score example of a musical work inspired by polyvalent structure music, that can be used by composers as a starting point to work with the environment. This example contains relatively few elements, which shows the practical expressiveness of the language.

2 Existing works

The sequencer metaphor is well-known amongst audio engineers and music composers. It is generally composed of tracks, which contains audio or MIDI clips, applied effects and parameter automations.

In multiple cases, it has been shown that it was possible to write more generalist multimedia time-line based sequencers, without the need to restrict oneself to audio data types. The MET++ framework[2] is an object-oriented framework tailored to build such multimedia applications. A common approach, also used in previous version of i-score, is to use constraint programming to represent relations between temporal objects[3, 1, 4]. This is inspired from Allen's relationship between temporal objects. In [5], Hirzalla shows how conditionality can be introduced between multimedia elements in a time-line to produce different outcomes.

Other approaches for interactive music are generally not based on the time-line metaphor, but more on interaction-centric applications written in patchers such Max/MSP or PureData, with an added possibility of scoring using cues. Cues are a set of parameters that are to be applied all at once, to put the application or hardware in a new state. For instance, in a single cue, the volume of a synthesizer may be fixed at the maximum value, and the lights would be shut off. However, the temporal order is then not apparent from the visual representation of the program, unless the composer takes care of maintaining it in his patch. When using text-based programming environments, such as Processing or OpenFrameworks, this may not be possible if concurrent processes must occur (e.g. a sound plays while the lights fade-in).

Antescofo[6] is another approach for real-time interactive scoring targeted towards live score following.

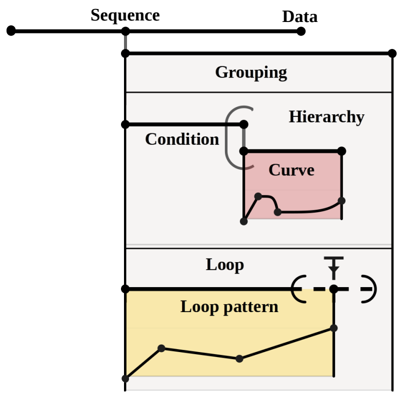

Figure 1. Screen-shot of a part of i-score, showing major elements of the formalism. The *time constraint* is the full or dashed horizontal line, the *states* are the black dots, the *time nodes* are the vertical bars, and a *time event* is shown at the right of the "Condition" text. Interactive *triggers* are black **T**'s with a downwards arrow. There are five *time processes* (capitalized): a *scenario* which is the hierarchical root of the score, another *scenario*, in the box "Hierarchy", an *automation* on a remote parameter in the "Curve" box, a *loop* in the box containing the *loop* pattern, and another *automation* that will be looped.

The syntax and graphical elements used in i-score as well as the execution semantics are for the most part introduced in [7, 8]. Multiple execution semantics for the same graphical formalism have been developed. They are based on Petri Nets[9], Time Automata[10], reactive languages[11] and temporal concurrent constraint programming[12].

The novelty of our approach lies in the introduction of graphical temporal loops, and of a computation model based on JavaScript that can be used at any point in the score. These two features, when combined, provide more expressive power to the i-score visual language, which allows for more dynamic scores.

3 Temporal structured programming

Structured programming is a paradigm which traces back to the 1960's, and was conceived at a time where the use of `GOTO` instructions was prevalent, which often led to code difficult to understand.

The structured programming theorem[13, 14] states that any computable function can be computed without the use of `GOTO` instructions, if instead the following operations are available:

- Sequence (`A` followed by `B`),
- Conditional (`if(P) then A else B`),
- Iterative (`while(P) do A`).

Where `P` is a boolean predicate, and `A`, `B` are basic blocks. Additionally, the ability to perform computations is required in order to have a meaningful program.

In order to allow for interactive musical scores authoring, we introduce these concepts in the time-line paradigm. A virtual machine ticks a timer and makes the time flow in the score graph. During this time, multiple processes, including hierarchical scoring, looping, and multimedia processes, are computed.

Processes can be of two kinds: temporal or instantaneous. Temporal processes can be explained as functions of time that the composer wants to run between two points in time: *do a volume fade-in from t=10s to t=25s*. Instantaneous processes are functions that will be run at a single point in time: *play a random note*.

3.1 Scenario

The *scenario* is a particular setup (fig. 1) of the elements of the i-score model: *time constraint* (a span of time which contains temporal processes), *time node* (which will synchronize the ending of *time constraints* with the happening of an external event such as a note being played), *time event* (a condition that will cause its following *time constraints* to start if it is true), and *state* (which contains data to send and instantaneous processes). Time flows from left to right as in most traditional sequencers. Due to the presence of interactivity, the various possibilities of execution of the score cannot be shown. The presence of a single interaction point makes the state space enormous. Hence dashes are shown when the actual execution time is not known beforehand. For instance: *play a D minor chord until a dancer moves on stage*.

In the context of a *scenario*, as shown in [7], these primitives allow for sequencing elements, conditional branching, and interactive triggering, but are not enough for looping. The user interface allows for all the common and expected operations when editing a *scenario*: displacement, scaling, creation, deletion, copy-paste...

3.2 Loop

The loop is another setup of these elements, more restrictive, and with a different execution algorithm: it is composed exclusively of two *time nodes*, two *time events*, two *states*, and a *time constraint* in-between, which we name loop pattern. When the second *time node* gets executed, the time flow is reverted to before the execution of the first *time node*. This means that if the composer adds an interactive trigger on any of these *time nodes*, the *loop* will be able to play patterns of different durations at each loop cycle, with a different outcome. This is strictly more general than loops in more traditional sequencers such as Avid Pro Tools, Steinberg Cubase or Apple Logic, when looping consists in a strict duplication of the audio or MIDI data.

As for the *scenario*, the *loop* is implemented as a temporal process that will be inserted under an existing *time constraint*. Hence the *loop*'s overall duration will be limited by its parent constraint's duration.

3.3 Communication

The software communicates via the OSC [1] protocol, and Minuit: an OSC-based RPC [2] and discovery protocol. It maintains a tree of parameters able to mirror the object model of remote software built with Max/MSP, PureData, or any OSC-compliant environment. Conditions, interactive triggers, and all processes of i-score have access to this tree. They can read its data, perform computations on it, and send it back to the remote software or hardware. In the course of this paper, "device tree" refers to this tree.

Musical information can currently be expressed with the help of either a remote software or hardware device able to convert OSC messages to either MIDI or audio.

[1] Open Sound Control
[2] Remote Procedure Call

3.4 Variables

Variables in i-score are based on the device tree, which acts like a global memory. The composer can create new variables graphically in the tree, and select their types. The variables are statically typed:

- A variable's type must be chosen between integer, boolean, floating point, impulse, string, character, or tuple.
- C-like implicit conversion can take place: an integer and a floating point number will be able to be compared.

While no mechanism allows dynamic allocation of new variables during the execution of a score, it could be implemented by providing a special address that would act like a virtual `malloc` operation on the i-score tree. Finally, there is no scoping: any process can access to any variable at any point in time. Variables should be considered elements of the object model of the local software. However, when local variables are necessary as intermediary steps for complex computations, it may be better to use the embedded scripting facilities to encapsulate them.

4 Authoring features

i-score provides multiple authoring features aiming to give a fair amount of freedom to the composer. Besides, the software is based on a plug-in architecture allowing for easy extension of its capabilities. Here is a list of the most prominent features of i-score:

- JavaScript support: one can use JavaScript scripts both as temporal and instantaneous processes. When writing a temporal process, the composer has to provide a function of the following form:

```
function(t) { return [ {
      address : '/an/address',
      value : 42 * t
    } ]; }
```

  This function will get called at each tick with $t \in [0; \infty[$, 1 being the default date set by the author. It creates a linear ramp on the `/an/address` parameter, going from zero to 42 during the duration of the parent *time constraint* of this process. If the process lasts longer than its expected duration due to interactive triggering, `t` will become greater than 1. The messages returned will be applied to the local state of the tree and sent remotely. When writing an instantaneous process, the function to provide is similar, but does not take any arguments. A global object provides an API to query the current state of other parameters in the device tree. This allows for arbitrarily complex mappings between parameters.
- Automation: the traditional DAW [3] automation, which writes in a parameter over time. Currently provided are 1D (linear, power) and 3D (cubic spline) automations.
- Mapping: this process takes an OSC address as input, applies a transfer function drawn graphically, and writes the output to another address.
- Recording: one can record either automations which will be able to apply automatic interpolation for numeric parameters, or record any kind of input message to replay it as it happened afterwards.
- Execution speed control: the temporal constraints all have a multiplicative factor for execution speed.
- Introspection: i-score exposes the current score in the same tree that the remote devices. It follows the hierarchical model defined by the *scenario* and the *loop*. Attributes can be queried, and to some extent modified during the execution of the score : status of interactive triggers, execution speed of temporal constraints, etc. The elements are organized hierarchically according to their hierarchy in the *scenario*, and can also be remote-controlled from the network. Plug-ins can expose their own attributes in this tree.
- Interactive execution: during the authoring process, it may be necessary to play from a specific point in the score. The state of the score, tree, and external devices is computed as if a full play-through had happened up to this point.

[3] Digital Audio Workstation

5 Temporal design patterns

In this section, we present two design patterns that can be used for writing an interactive score. We will first showcase event-driven scores, which can be used to get a behaviour similar to a traditional computer program executing instructions in sequence without delay, or common network communication tasks. Then, we will present an example of the concept of procedure in a time-oriented model.

5.1 Event-driven design

Event-driven, or asynchronous design is a software design paradigm that is centered on the notion of asynchronous communication between different parts of the software. This is commonly used when doing networked operations or user interfaces, since they have to be responsive.

In textual event-driven programming, one would write a software using callbacks, futures or reactive programming patterns[15].

With interactive triggers, one can easily write such event chaining. An example is given in fig. 2. The advantage is

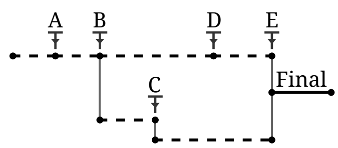

Figure 2. An example of event-driven score: if all the interactive trigger's conditions are set to true, they will trigger at each tick one after the other. Else, standard network behaviour is to be expected.

that ordered operations are easily written: B cannot happen before A if there is a *time constraint* between A and B. However, the execution engine will introduce a delay of one tick between each call. The tick frequency can be set to as high as one kilo-hertz. Synchronization is trivial: here, the last *time constraint* **Final**, will only be executed after all the incoming branches were executed. This allows to write a score such as: *start section B five seconds after musician 1 and 3 have stopped playing*. There is no practical limit to the amount of branches that can be synchronized in this way.

5.2 Simulating procedures

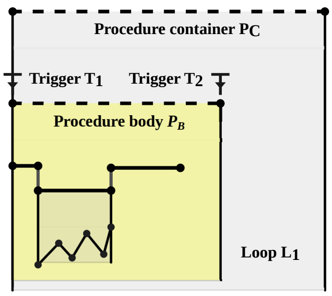

Figure 3. Implementation of a procedure in i-score.

The notion of procedure is common in imperative programming languages. It consists in an abstraction around a behaviour that can be called by name easily. It can be extended to the notion of function which takes arguments and returns values. However, it causes the visual flow to loose coherence: the definition of the procedure is given at a point in the score, and its usage occurs elsewhere.

Fig. 3 gives a procedure P able to be recalled at any point in time. There is a restriction due to the temporal nature of the system: it can only be called when it is not already running. This is due to the single-threaded nature of the execution engine: there is only a single playhead for the entirety of the score, hence a single temporal process cannot run at the same time at two different points.

The procedure is built as follows:

- A *time constraint*, P_C in the root *scenario* will end on an interactive triggering set with infinite duration.
- This *time constraint* contains a *loop* L_1. The procedure is named `p` in the local tree. The interactive triggers T_1, T_2 at the beginning and end of the pattern *time constraint* are set as follows:
  - T_1: `/p/call true`.
  - T_2: `/p/call true`.

  A *state* triggered by T_1 should set the message: `/p/call false`. This causes the procedure not to loop indefinitely: it will have to be triggered manually again.
- The *loop*'s pattern P_B contains the actual procedure data, that is, the process that the composer wants to be able to call from any point in his score.

The execution of this process will then overlay itself with what is currently playing when at another point of the score, the message `/p/call true` is sent. Once the procedure's execution is finished, it enters a waiting state until it is called again. This behavior is adapted to interactive arts: generally, one will want to start multiple concurrent processes (one to manage the sound, one to manage videos, one to manage lights...) at a single point in time; this method allows to implement this.

6 Musical example: polyvalent structure

In this example (fig. 4), we present a work that is similar in structure to Karlheinz Stockhausen's *Klavierstück XI* (1956), or John Cage's *Two* (1987). The complete work contains variables in the tree and a temporal score. The tree is defined in fig 5.

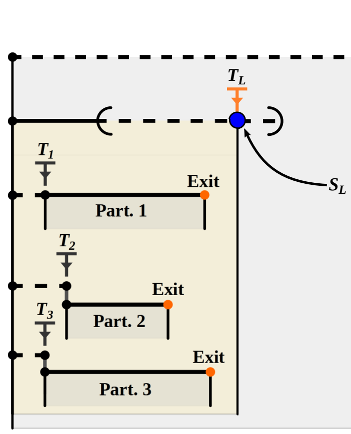

Figure 4. An example of polyvalent score in i-score

Address	Type	Initial value
`/part/next`	integer	chosen by the composer
`/part/1/count`	integer	0
`/part/2/count`	integer	0
`/part/3/count`	integer	0
`/exit`	boolean	false

Figure 5. Tree used for the polyvalent score

`/part/next` is an address of integral type, with a default value chosen by the composer between 1, 2, 3: it will be the first played part. The score is as follows: there are multiple musical parts containing recordings of MIDI notes: **Part. 1, 2, 3**. These parts are contained in a *scenario*, itself contained in a *loop* that will run indefinitely. At the end of each part, there is an orange *state* that will write a message "true" to a variable `/exit`. The pattern of the *loop* ends on an orange interactive trigger, T_L. The *loop* itself is inside a *time constraint* ended by an interactive trigger, T_E. Finally, the parts are started by interactive triggers $T_{\{1,2,3\}}$.

The conditions in the triggers are as follows:

- $T_{\{1,2,3\}}$ `/part/next == {1, 2, 3}`
- T_L `/exit == true`
- T_E $\bigvee_{i \in 1..3}$ `/part/i/count > 2`

The software contains graphical editors to set conditions easily. Finally, the blue *state* under T_L contains a JavaScript function that will draw a random number between 1 and 3, increment the count of the relevant `/part`, and write the drawn part in `/part/next` :

```
function() {
  var n = Math.round(Math.random()*2)+1;
  var root = 'local:/part/'
  return [ {
    address : root + 'next',
    value   : n
  }, {
    address : root + n,
    value : iscore.value(root + n) + 1
  } ];
}
```

If any count becomes greater than two, then the trigger T_E will stop the execution: the score has ended. Else, a new loop iteration is started, and either T_1, T_2 or T_3 will start instantaneously.

Hence we show how a somewhat complex score logic can be implemented with few syntax elements.

Another alternative, instead of putting MIDI data in the score, which makes it entirely automatic and non-interactive, would be to control a screen that displays the part that is to be played. A musician would then interpret the part in real-time.

7 Conclusion

We presented in this paper the current evolutions of the i-score model and software, which introduces the ability to write interactive and variable loops in a time-line, and the usage of JavaScript to perform arbitrary computations on the state of the local and external data controlled by i-score.

Currently, the JavaScript scripts have to be written in code, even if it is in a generally visual user interface. But given enough testing and user evaluation, it could be possible to have pre-built script presets that could be embedded in the score for the tasks that are the most common when writing a score.

Additionally, we aim to introduce audio and MIDI capabilities in i-score, so that it will be able to work independently of other sequencers. For instance, should it play a sequence of three sounds separated by silence, it would be difficult for the composer if he had to load the songs in an environment such as Ableton Live, and work with them remotely from the other time-line of i-score.

This would also allow for more control on the synchronization of sounds: if they are controlled by network, the latency can cause audio clips that are meant to be synchronized in a sample-accurate manner to be separated by a few milliseconds, it is enough to prevent the usage in some musical contexts.

Acknowledgments

This work is supported by an ANRT CIFRE convention with the company Blue Yeti under funding 1181-2014.

8 References

[1] A. Allombert, G. Assayag, M. Desainte-Catherine, C. Rueda *et al.*, “Concurrent constraints models for interactive scores,” in *Proc. Sound and Music Computing 2006*, 2006.

[2] P. Ackermann, “Direct Manipulation of Temporal Structures in a Multimedia Application Framework,” in *Proceedings of the Second ACM International Conference on Multimedia*, ser. MULTIMEDIA ’94. New York, NY, USA: ACM, 1994, pp. 51–58.

[3] J. Song, G. Ramalingam, R. Miller, and B.-K. Yi, “Interactive authoring of multimedia documents in a constraint-based authoring system,” *Multimedia Systems*, vol. 7, no. 5, pp. 424–437, 1999.

[4] M. Toro-Bermúdez, M. Desainte-Catherine *et al.*, “Concurrent constraints conditional-branching timed interactive scores,” in *Proc. Sound and Music Computing 2010*. Citeseer, 2010.

[5] N. Hirzalla, B. Falchuk, and A. Karmouch, “A Temporal Model for Interactive Multimedia Scenarios,” *IEEE Multimedia*, vol. 2, no. 3, pp. 24–31, 1995.

[6] A. Cont, “ANTESCOFO: Anticipatory Synchronization and Control of Interactive Parameters in Computer Music.” in *International Computer Music Conference (ICMC)*, Belfast, Ireland, Aug. 2008, pp. 33–40.

[7] J.-M. Celerier, P. Baltazar, C. Bossut, N. Vuaille, J.-M. Couturier, and M. Desainte-Catherine, “OSSIA: Towards a unified interface for scoring time and interaction,” in *Proceedings of the 2015 TENOR Conference*, Paris, France, May 2015.

[8] P. Baltazar, T. de la Hogue, and M. Desainte-Catherine, “i-score, an Interactive Sequencer for the Intermedia Arts,” in *Proceedings of the 2014 Joint ICMC-SMC Conference*, Athens, Greece, 2014, pp. 1826–1829.

[9] A. Allombert, G. Assayag, M. Desainte-Catherine *et al.*, “A system of interactive scores based on Petri nets,” in *Proc. Sound and Music Computing 2007*, 2007, pp. 158–165.

[10] J. Arias, M. Desainte-Catherine, and C. Rueda, “Exploiting Parallelism in FPGAs for the Real-Time Interpretation of Interactive Multimedia Scores,” in *Proc. Journées de l’Informatique Musicale 2015*, 2015.

[11] J. Arias, M. Desainte-Catherine, S. Salvati, and C. Rueda, “Executing Hierarchical Interactive Scores in ReactiveML,” *Proc. Journées de l’Informatique Musicale 2014*, pp. 25–34, 2014.

[12] M. Toro, M. Desainte-Catherine, P. Baltazar *et al.*, “A model for interactive scores with temporal constraints and conditional branching,” *Proc. Journées de l’Informatique Musicale 2010*, pp. 31–38, 2010.

[13] C. Böhm and G. Jacopini, “Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules,” *Commun. ACM*, vol. 9, no. 5, pp. 366–371, May 1966.

[14] H. D. Mills, *Mathematical foundations for structured programming*, 1972.

[15] K. Kambona, E. G. Boix, and W. De Meuter, “An evaluation of reactive programming and promises for structuring collaborative web applications,” in *Proceedings of the 7th Workshop on Dynamic Languages and Applications*. ACM, 2013, p. 3.