Graphical Temporal Structured Programming for Interactive Music

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 argue about the specific issues that arise in such environments, and we present the current version of the i-score sequencer. It is an extensive software suite that bridges the gap between time-based, logic-based and flow-based interactive application authoring tools. This is done in a single cohesive graphical user interface, built upon a few simple and novel primitives that give to the composer the expressive power of structured programming, in a time line adapted to the notation of parameter-oriented interactive music.

1. INTRODUCTION

2. TEMPORAL MULTIMEDIA PROGRAMMING

[1][2] - voir mails envoyés à théo, myriam, etc. - langages visuels - références anciens articles i-score - protocoles ? - dire qu'il n'y a pas directement de support des données musicales.

3. SYNTAX

The syntax and graphical elements used in i-score as well as the execution semantics are for the most part introduced in [3, 4]. Multiple execution semantics for the same graphical formalism exist. They are based on Petri Nets, Time Automata and Reactive languages.

The novelty lies in the introduction of temporal loops, and of a computation model based on Javascript. These two features allow complete expressive power in the written scores.

3.1 Variables

Variables can be easily implemented by - Variables ? -¿ can be stocked in the tree -¿ no notion of scope - JS can be used for local variable concept

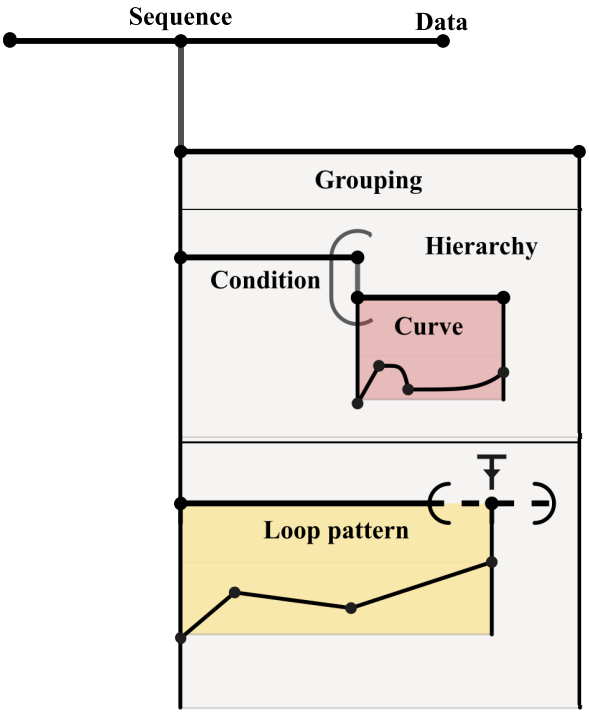


Figure 1. Screenshot of a part of i-score, showing major elements of the formalism.

3.2 Temporal structured programming

Structured programming is a programming paradigm which traces back to the 60's, and was conceived at a time where the use of `GOTO` instructions was prevalent in computing, which often led to code difficult to understand.

The structured programming theorem[5, 6] 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(Pred) then A else B`),
- Iterative (`while(Pred) do A`).

Additionnaly, the ability to perform computations is required in order to have a meaningful program.

In order to allow for interactive musical scores authoring, we simulate 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, which include scenarisation, looping, and multimedia processes, are run at each tick.

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. For instance : *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. For instance : *send a single random number to an OSC address*.

Some elements of the syntax are shown in fig. 1.

The scenario is an organisation of the elements of our formalism which allows for sequencing elements, conditional branching, and interactive triggering. The loop is another organization of the same elements, more restrictive, and with a different control flow.

The software communicates via OSC and maintains a tree of parameters able to mirror the object model of remote software built with Max/MSP, PureData, Unity3D or any OSC-compliant environment. Conditions, interactive triggers, and more generally all processes of i-score have unrestricted access to this tree, and can read its data, perform computations on it, and write it back to the remote software.

4. AUTHORING FEATURES

i-score provides multiple authoring features aiming to allow the composer to express himself. Besides, the software is based on a plug-in architecture which allows for easy extension of its capabilities.

- 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 form :

```
function(t) {
    var obj = new Object;
    obj["address"] = '/an/osc/address';
    obj["value"] = 42;
 return [ obj ]
}
```

  When writing an instantaneous process, the function to provide is similar, but does not take the time in argument. Functions writer are encouraged to write stateless functions, since it would allow for optimizations opportunities such as precomputing an array of values.

- Automation : the traditional DAW automation, which writes in a parameter over time. Currently provided are 1D (linear, power) and 3D (cubic spline) automations.

- Mapping : similar in appearance to the automation, this process will take an OSC address as input, apply a transfer function, and write 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 afterwards.

- Execution speed control : the temporal constraints all have a multiplicating coefficient for execution speed.

- Introspectability : i-score exposes the current score in the same tree that the remote devices. Multiple attributes can be queried, and to some extent modified during the execution of the score. For instance, the current position of the play head, the activation status of interactive triggers, and the execution speed of all the temporal constraints are controlable parameters. The elements are organized hierarchically according to their hierarchy in the scenario, and can also be controled from the network. Plug-ins can expose their own attributes in this tree.

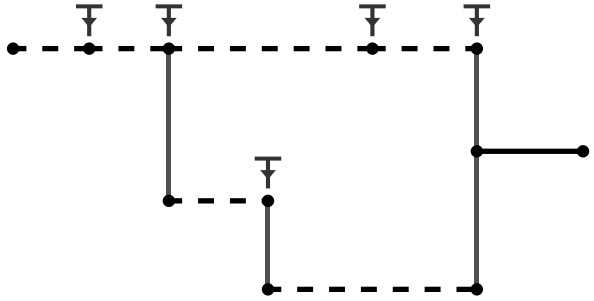


Figure 2. An example of event-driven score

- Interactive execution : during the authoring process, it is sometimes necessary to replay just a part of the score. i-score provides this capacity. However, it is necessary for the composer to choose beforehand the truth value of conditions he wants to test. There are two "debug" execution modes : one that will directly start playing from where the user points the mouse, and another that will try to merge all the previously sent values, to put the external environment in the same state that it would be if the score had been played normally up until this point. This breaks if there are physical processes that are not instantaneous, as would be the case for smoke machines for instance.

5. TEMPORAL DESIGN PATTERNS

In this section, we present various design patterns that can be used when one wants to build an interactive score with i-score. We will showcase event-driven scores, which will have a behaviour similar to a traditional computer program executing instruction after instruction at maximal speed, and an example in simulating the concept of function as it exists in the C language.

5.1 Event-driven design

Event-driven, or asynchronous design is a softwar 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.

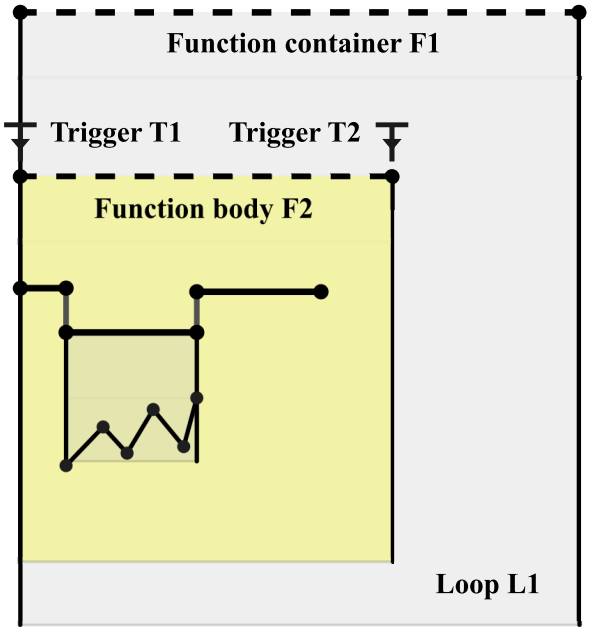
Instead of writing : `play A; play B; play C`, in event-driven programming, one would write :

```
when A is triggered
    play B
when B is triggered
    play C
play A
```

Since i-score supports interactive events, one can easily write such event chaining. An example is given in fig. 2.

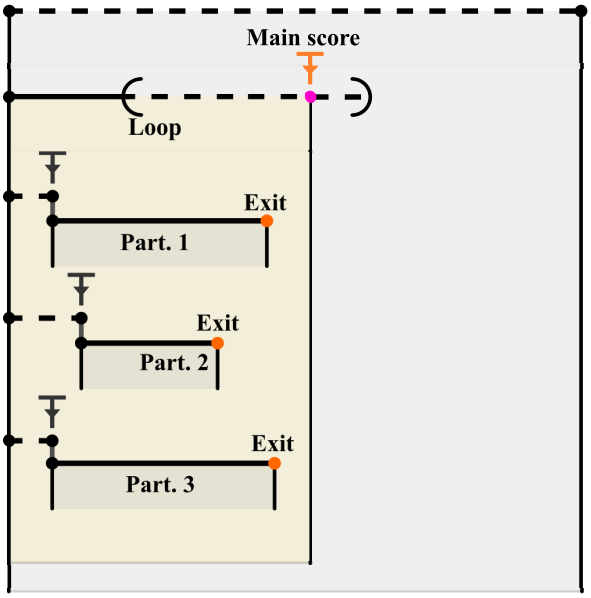
The advantage is that ordered operations can be written easily : there is no possibility for B to happen before A if there is a time constraint between A and B. However, the execution engine will introduce a delay of one tick between each call. The tick rate can be set to as low as one kilohertz.

5.2 Simulating functions



6. MUSICAL EXAMPLES

6.1 Klavierstück



6.2 Fibonacci recursive

7. CONCLUSION

- plug-in architecture - open-source - soon : audio and midi support (midi can be done today with osc wrappers but a piano-roll or sheet music view would be nice) - computation graph may be useful to plug the input of a box to an output without using the tree - other features : spatial, etc. - embeddable - problématique de déboguage, analyse statique

Acknowledgments

This work was funded by Blue Yeti and ANR.

8. REFERENCES

[1] P. Ackermann, "Direct manipulation of temporal structures in a multimedia application framework," in *Proceedings of the second ACM international conference on Multimedia*. ACM, 1994, pp. 51–58.

[2] 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.

[3] J.-M. Celerier, P. Baltazar, C. Bossut, N. Vuaille, J.-M. Couturier *et al.*, "OSSIA: Towards a unified interface for scoring time and interaction," in *Proceedings of the 2015 TENOR Conference*, 2015.

[4] P. Baltazar, T. de la Hogue, and M. Desainte-Catherine, "i-score, an Interactive Sequencer for the Intermedia Arts," pp. 1826–1829, 2014.

[5] 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.

[6] H. D. Mills, "Mathematical foundations for structured programming," 1972.