1 Introduction

Objectif: lier paradigme du séquenceur et paradigme du patcher

Donner une sémantique formelle à l'exécution de ces structures.

Ce qui manque: déclaration de variables locales pour nommage ? Actuellement on utilise des variables globales si on veut un nom.

=> comment est-ce qu'on peut combiner de la meilleure manière possible l'univers des patchers et des séquenceurs.

=> décrire requirements pour qu'un autre système de contraintes temporelles puisse fonctionner => au minimum : enable / disable => set date => set offset pour offset audio (p-ê pas nécessaire si on fait comme LAStream)

Pbq si on fait ça en fonctionnel: construction obligatoire en "top-down" ? on commence par les processus, puis les contraintes, etc. sinon on doit parcourir tout pour remplacer ce qu'on veut

- Langage: formalisation par ML ? utilisation par QML ?
- Tête d'un langage "ML" : quelles sont les opérations que l'on fait ?

2 Temporal model

We note: TC for the temporal conditions, IC for the instantaneous conditions, I for the intervals. chaining:

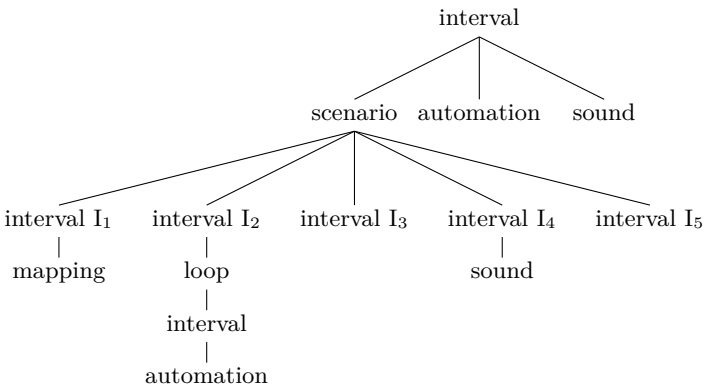

Figure 1: Hierarchical tree

2.1 Data types

process, interval, event, sync

2.1.1 Conditions and expressions

We first define the conditional operations we want to be able to express. We restrain ourselves to simple propositional logic operands: **and**, **or**, **not**.

Expressions operate on addresses and values of the device tree presented in chap. **??**, according to the grammar in **??**.

Formally, expressions are defined as a tree: Let **Comparator** be an identifier for standard value comparison operations: $<, \leq, >, \geq, =, \neq$ and **Operator** standard logical operators **and** & **or**.

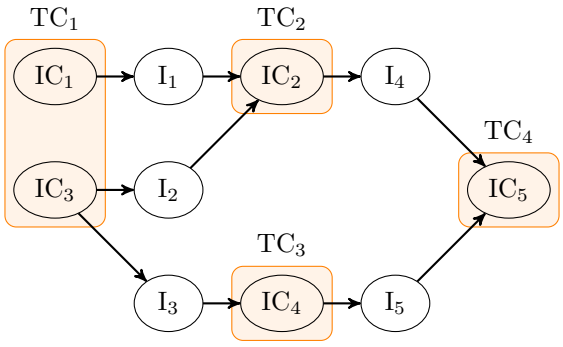

Figure 2: Temporal DAG

$$
\begin{aligned}
\textbf{Atom} &: (\text{Parameter} \mid \text{Value}) \times (\text{Parameter} \mid \text{Value}) \times \text{Comparator} \\
\textbf{Negation} &: \text{Expression} \\
\textbf{Composition} &: \text{Expression} \times \text{Expression} \times \text{Operator} \\
\textbf{Impulse} &: \text{Parameter} \times \text{Bool} \\
\textbf{Expression} &: \text{Atom} \mid \text{Negation} \mid \text{Composition} \mid \text{Impulse}
\end{aligned}
$$

Two operations are defined on expressions and the data types that compose them:

- **update** : Expression → Expression. Used to reset any internal state and query up-to-date values for the expressions. For instance, **update** on an **Atom** fetches if possible new values for the parameters, why may include network requests.
- **evaluate** : Expression → Bool. Performs the actual logical expression evaluation, according to the expected rules.

Precisely:

$$
\begin{cases}
\text{update}: & \text{Composition} \to \text{Composition} \\
& (e_1, e_2, o) \mapsto (\text{update } e_1, \text{update } e_2, o) \\
\text{update}: & \text{Negation} \to \text{Negation} \\
& (e_1) \mapsto (\text{update } e_1) \\
\text{update}: & \text{Atom} \to \text{Atom} \\
& \begin{cases}
(p_1, p_2) \mapsto (\text{pull } p_1, \text{pull } p_2) \\
(p_1, v_2) \mapsto (\text{pull } p_1, v_2) \\
\dots
\end{cases} \\
\text{update}: & \text{Impulse} \to \text{Impulse} \\
& (p, b) \mapsto (p, \text{false})
\end{cases}
$$

- An atom is a comparison between two parameters, a parameter and a value, or two values.
- Negations and compositions are the traditional predicate logic building blocks.
- We introduce a specific operator, “impulse”, which allows to decide whether a value was received.

2.1.2 Interval

We want to be able to express the passing of time, for a given duration. This duration may or may not be finite.

A duration is defined as a positive integer. An interval is at its core a set of durations: a min, an optional max, and the current position. The lack of max means infinity. An interval is said to be fixed when its min equals its max.

$$\textbf{Interval} = \text{Duration} \times \text{Optional<Duration>Duration}$$

The time scale is not specified by the system: for instance, when working with audio data it may be better to use the audio sample as a base unit of time. But many applications don't use the audio rate: when working purely with visuals it may be better to use the screen refresh rate as time base in order not to waste computer resources and energy.

2.1.3 Instantaneous condition

2.1.4 Temporal condition

Then, we want to be able to enable or disable events and intervals according to a condition, given in the expression language seen in **??**.

$$\textbf{Condition} = \text{Expression}$$

A condition has two associated operations: `evaluate` and `update`.

```
update expression -> expression

evaluate expression -> Bool
```

2.1.5 Process

2.1.6 Operations

```
add_process interval process -> interval

add_event sync
```

exécution :
interval:

```
tick: interval * t -> interval * state
```

processes:

```
state: process * t -> process * state
```

2.2 Temporal graph: scenario

2.2.1 Creational operations

```
state:

process_event:
```

2.2.2 Execution operations

```
add_interval sc itv sev eev
```

```
add_sync sc
```

2.3 Loop

Pbq: not introducing cycles in the temporal graph

3 Data model

3.1 Data types

3.2 Operations

3.3 Data graph

3.4 Data processes

3.4.1 Automation

3.4.2 Mapping

3.4.3 JavaScript

3.4.4 Piano Roll

3.4.5 Sound file

3.4.6 Sound input

3.4.7 Shader

3.4.8 3D model

gltf ?

4 Combined model

4.1

5 Proposed sequencer behaviour

UI: création automatique de liens implicites des enfants vers les parents => "cable créé par défaut" quand on rajoute un processus dont on marque l'entrée

=> pour toute contrainte, pour tout scénario, créer noeud qui fait le mixage => création d'objets récursivement, etc

- Problème des states dans scénario ? => states du scénario: comment interviennent-ils ? faire un scénario fantôme *

- Mettre l'accent sur la recréation de la sémantique de i-score à partir du graphe: => messages: actuellement "peu" typés ; rajouter type de l'unité ?

=> pbq du multicanal: pour l'instant non traitée, on ne gère que les cas mono / stereo pour le upmix / downmix Choix pour multicanal: faire comme jamoma avec objets tilde => sliders

et dispatching de canaux ? => cables: rubberband ? il faut mettre un rubberband dès qu'on a une entrée et une sortie qui n'ont pas la même vitesse relative. Dire que pour les automations ça interpole de manière naturelle avec le ralentissement et l'accélération (on sépare vitesse et granularité)

Exécution complète d'un tick:

- Copie des buffers audio - Exécution du tick temporel - Récupération des states

- Dire qu'on pourrait affiner en combinant plus précisément les "sous-ticks" temporels et de données pour que par exemple la production d'un état dans un scénario entraîne une condition dans un autre scénario

6 Applications

- Exemple article Myriam: micro-montage et sélection d'effets - Carrousel

7 Evaluation

- Recréer séquenceur traditionnel, patcher, et ableton live (vue session).

=> "third gen" audio sequencer. first gen: cubase, etc second gen: non-linear: ableton, bitwig third gen: i-score

8 Conclusion