The Mixing Algebra of Musical Elements in µO

Stéphane Rollandin
hepta@zogotounga.net

draft - 14 November 2013

Abstract

The `MusicalElement` hierarchy implements the main classes for reification of musical structures such as notes, phrases, rhythms, csound scores, and more. We discuss in this paper the unifying algebra allowing all these objects to be composed.

Notation

In the following, the printed evaluation of a Smalltalk expression is represented following a ▶ symbol. When a graphic representation is available (a screenshot of a µO editor in most cases), it is displayed after a ▶. All code is written in `Consolas font`.

1. Temporal structure of a `MusicalElement`

A musical element is always anchored in a local temporal frame extending from time 0 to an arbitrary duration. Relatively to that frame, the musical element always occupy a definite position, starting at a given arbitrary time and ending at another arbitrary time.

In the following we will use diagrams to illustrate the temporal structure of a musical element. Below is a musical element with a shorter duration than its end time:

```
  start time            end time
     |                     |
     [   MUSICAL ELEMENT   ]
------------
|          |
0        duration
```

Below is another element with a negative starting time, and a duration extending its end time:

```
start time         end time
   |                  |
   [ MUSICAL ELEMENT ]
       ------------------------
       |                      |
       0                   duration
```

Start time, end time and duration can be negative. These values are meant to localize the musical data with respect to its local time 0.

1.1. Delay

The elementary operator `#delay:` moves the musical element in its local frame.

Let's consider the simple element below:

```
[  music 1 ]
------------
0          d1
```

It is simple in the sense that it fully occupies its local temporal frame: its start time is 0 and its end time is equal to the frame duration.

The effect of `#delay:` (depending on weither its argument is positive or negative) can be

```
     [  music 1 ]
-----------------
0              d1+|delay|
```

or

```
[  music 1 ]
   ---------
   0      d1-|delay|
```

The delay effect is an offset of the musical element within its temporal frame, along with the modification of that frame duration.

If the element is not simple, as in

```
  [  music 2 ]
---------
0       d2
```

then the effect of `#delay:` can be

```
        [  music 2 ]
---------------
0            d2
```

or

```
    [  music 2 ]
    ---
    0  d2
```

or even, resulting in a negative duration,

```
[  music 2 ]
      ----
      d2   0
```

2.1. Scaling

The operator `#scale:` stretch a musical element relatively to the origin of its temporal frame, as illustrated by the following diagrams :

The element

```
    [  music ]
----------------
0            duration
```

scaled to twice its duration becomes

```
        [      music       ]
--------------------------------
0                          duration
```

while the element

```
     [  music ]
-----
d    0
```

becomes

```
          [       music      ]
----------
d         0
```

2. Mixing operators

Two musical elements can always be mixed : their temporal frames are merged[1].

We will illustrate the various mixing operators with elements `A`,`B`,`C`,`D` as below:

```
[  music A ]
------------
0          dA

[    music B   ]
----------------
0              dB

   [  music C ]
---------------------------
0                         dC

     [  music D ]
-----------------
0               dD
```

1 This fundamental operation either results in a single musical element similar to the initial ones, such as the merging of `MusicalNote`s into a `MusicalPhrase`, or, when the mixed elements are not of the same nature, in a `CompositeMix`, where the initial musical elements are simply referenced at their respective positions.

2.1. Plain mixing

The mixing operator is |.

`A|B` is equal to `B|A` and looks like

```
[  music A ]
[    music B   ]
----------------
0              dB
```

`C|D` is also equal to `D|C`

```
   [  music C ]
     [  music D ]
---------------------------
0                         dC
```

2.2. Concatenation

The concatenation operator is `+` (also expressed as `,`).

The diagram for `A+B` (or `A,B`) is

```
[  music A ][    music B   ]
----------------------------
0                        dA+dB
```

while `B+A` (or `B,A`) looks like

```
[    music B   ][  music A ]
----------------------------
0                        dA+dB
```

`C,D` is

```
   [  music C ]                  [  music D ]
--------------------------------------------
0                                      dC+dD
```

and `D,C` looks like

```
     [  music D ]   [  music C ]
--------------------------------------------
0                                      dD+dC
```

2.3. Backward mixing

Backward mixing is mixing after having aligned the second element duration with the first element duration. The backward mixing operator is `//`

Contrary to `|`, `//` is not commutative: `A//B` is not the same as `B//A`.

`A//B` is

```
    [  music A ]
[    music B   ]
  --------------
  0            dA
```

while B//A is:

```
    [  music A ]
[    music B   ]
---------------
0              dB
```

C//D results in

```
   [  music C ]
              [  music D ]
--------------------------
0                        dC
```

while D//C is

```
   [  music C ]
              [  music D ]
         ----------------
         0               dC
```

2.4. Synchronization

The synchronising operator & scales its second argument to the duration of the first one.

A&B is A mixed with B scaled to A duration:

```
[  music A ]
[  music B ]
-----------
0          dA
```

while C&D is C mixed to D scaled to C duration:

```
   [  music C ]
        [      music D    ]
--------------------------
0                        dC


[  music A ][  music B ]
------------------------
0          dA          2*dA
```

C&D looks like

```
   [  music C ]
        [      music D    ]
--------------------------
0                        dC
```

2.5. Beat concatenation

The beat concatenation operator ,, concatenate its second argument scaled to the duration of the first, to the first.

So A,,B is

```
[  music A ][  music B ]
------------------------
0                      2*dA
```

B,,A is

```
[    music B    ][    music A    ]
---------------------------------
0                            2* dB
```

C,,D is

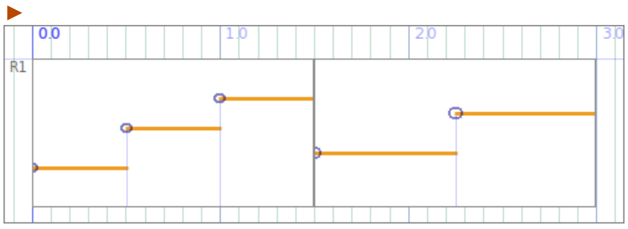
```
   [  music C ]                 [    music D    ]
-------------------------------------------------
0                                             2*dC
```

D,,C is

```
   [  music D ] [  music C ]
---------------------------------
0                           2*dD
```

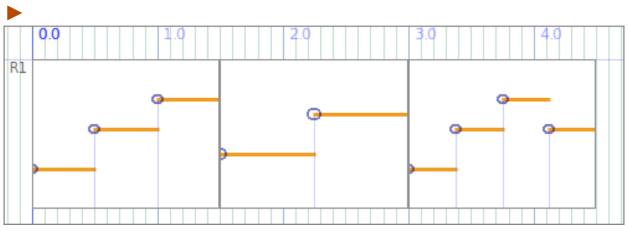
Beat concatenation is special in the sense that it always return a CompositeMix[2].

```
'c,e,g' kphrase ,, 'e,g' kphrase
```



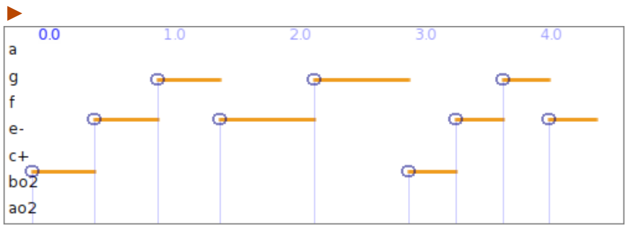
This is so that beat concatenations can be chained:

```
'c,e,g'  kphrase  ,,  'e,g'  kphrase  ,,  'c,e,g,e'
kphrase
```



The ,,, operator does beat concatenation and reduce the result:

```
'c,e,g'  kphrase  ,,  'e,g'  kphrase  ,,,  'c,e,g,e'
kphrase
```



2 See "Usages of CompositeMix in muO" for more

3. Solid duration

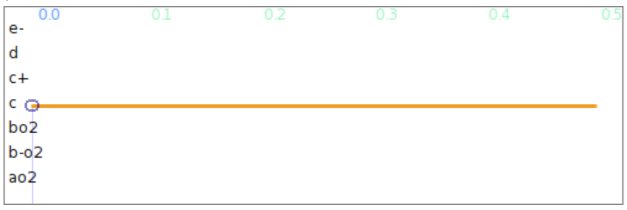
Since the duration of a musical element can be different from its end time, it has to be maintained separately: it is stored in each element as a value called "solid duration".

When a musical element has a non-`nil` solid duration, its duration is the solid duration. When the solid duration is `nil`, its duration is its end time.

Let's see an example of the usefulness of the notion of solid duration, with the implementation of articulation for musical notes.

We consider a plain C note:

```
note := 'c' knote
▶
```

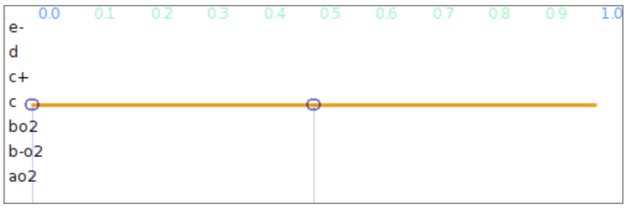


By default, it does not have a solid duration:

```
note duration
▶ 0.5
note solidDuration
▶ nil
note endTime
▶ 0.5
```

If we repeat this note by concatenation, we get:

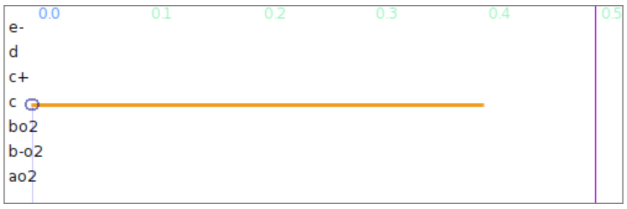
```
note, note
▶ 'c,c'
▶
```



The second note has been appended at the end of the first one: this is a legato articulation.

The `#duration:` selector enables setting the solid duration. Let's use it to change the structure of the note:

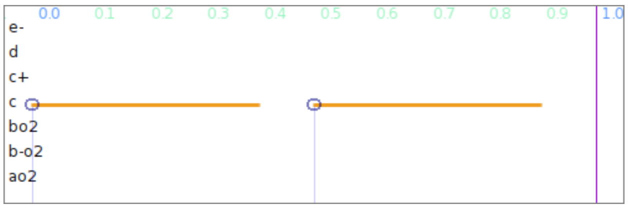
```
note duration: 0.5; length: 0.4.
▶
```



```
note duration
▶ 0.5
note solidDuration
▶ 0.5
note endTime
▶ 0.4
```

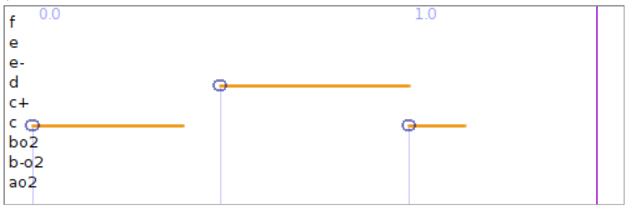
Because the note duration is now longer than its end time, its articulation is a staccato:

```
note, note
▶ 'cd77,ct96,l192'
▶
```



The `#staccato:` and `#legato` selectors are based on this principle. The first one takes a numeric argument giving the ratio of a musical element duration to be taken as its actual length, while the second identifies the two.

```
('c' knote staccato: 0.8), 'd' knote legato, ('c' knote staccato: 0.3)
▶
```


f
e
e-
d
c+
c
bo2
b-o2
ao2