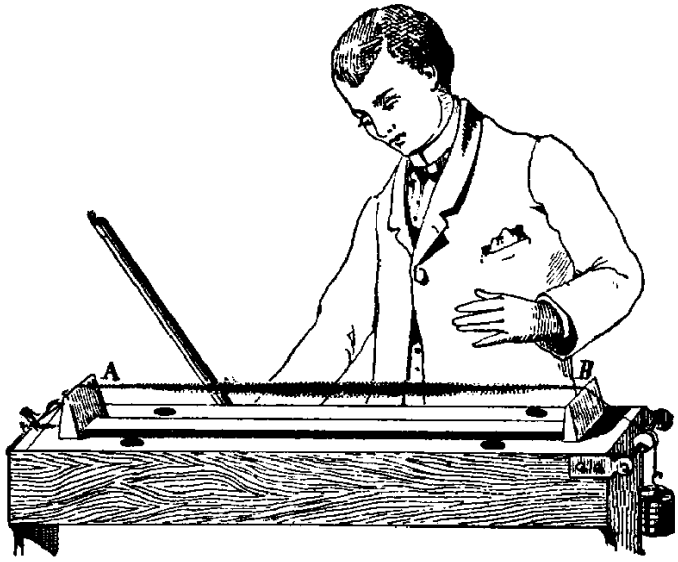
Director Musices 2.5

User Manual



Windows and Macintosh

Version: 2003-05-20

Introduction

Director Musices is a program that allows you to change the performance of a music score. It contains a set of rules that changes the duration, sound level etc of the notes. These rules mimic performance principles used by real musicians. The input quantity parameters given to the rules can drastically change a performance according to the intentions of the meta-performer (that is, you). This includes the modeling of different emotional characters such as “happy” or “sad”. The rules are a result from a long-term research project at the Royal Institute of Technology, Stockholm and have been mainly developed by Lars Frydén, Johan Sundberg and myself. For further information see our web-site below and the references in the end. The program has been developed by me and Vittorio Colombo (PC). If you have any questions, comments or found bugs we would be glad to hear from you and we hope you will enjoy using Director Musices.

Anders Friberg

andersf@speech.kth.se

Correspondence about music performance including Director Musices:

musperf@speech.kth.se

Music performance pages:

http://www.speech.kth.se/music/performance/

Basic operation

Loading Music

Choose "Open music" on the menu "music" (file menu on PC). Select a melody in the folder "Scores" (with the extension .mus). A track window is opened where the track variables such as midi channel, mid volume can be changed. Observe that all operations on the music such as rule application and playing are only done on those tracks that are marked Active in the track window.

Playing

Select "Play" on the “Play” menu. Technically, it first generates a list that is sorted by time order before the actual playing. "PlayLast" plays the last list generated from "Play". Currently in the PC version a midifile is generated and the mediaplayer in the system is activated.

Applying Rules

Select "Open default rule palette ..." on the "Rules" menu and push “Play performed”. This will apply all the rules listed in the window with quantities specified by the scrollbars. It is recommended to listen to each rule in isolation using exaggerate, medium and negative quantities. In this way, it is possible to get an understanding of how each rule affect the performance. When several rules change the same parameter, the total change is the sum of changes from each rule. The combination of rules acting on the same parameter is additive. That is, if two rules (acting on the same note) increase the sound level by 1 dB, the resulting increase is 2 dB.

It is also possible to load other rule palettes (extension .pal) and define your own. Several rule palettes can be open at the same time for easy comparison of different performances.

When a rule affect more than one performance parameter, additional scaling of these can be done using the rule input parameters in the text window to the right of the rule name. For example,

:amp 0

means that the effect on the sound level will be zero.

:dur 2

means that the effect on the duration (interonset) will be twice the amount normally given by the rule at the specified quantity setting.

Menu Commands

File menu (Music menu on Mac)

`Open Score...`

The main function to open a score file (default extension ".mus"). Initialize also the performance variables. A track window is created.

`Open Performance...`

A load function used for files with all performance variables already set (default extension ".per").

`Open MIDI file...`

Reads a type 1 midifile (default extension ".mid"). Will not read much extra midi information such as program change etc. The tracks will be numbered according to the order in the midifile. It is recommended to use quantized midi files coming from a score editor. If a score editor is used, try to make sure that the note durations are the same as printed on screen. For example, the Encore program by default makes the play duration of each note to be 90% of it note value. This introduces in Director Musices an extra rest after each note, which in turn makes the rule application unpredictable. However, it is possible to open almost any midi file. The durations are not quantized and an attempt to approximate a note value is done. The rules use in most cases only the duration as input and do not rely on note values. These are needed only for the score display. If there are several independent voices in one track, they are converted to a single voice, with the exception that simultaneous notes with the same duration are allowed. Note velocities are currently not read.

`Save Score As...`

Saves the score and track variables of the current music. i.e. not the performance variables (default extension ".mus").

`Save Performance...`

Saves all variables of the current music. i.e. also the performance variables (default extension ".per"). This is the same as the internal representation of the performance at the time of saving. It can be used to save different kinds of pre-computed performances. "Open Performance..." must be used to open the file.

`Save Performance MIDI file 0 As...`

Will save the current music in a midifile type 0 with the same midi-codes as when it is played (default extension ".mid"). Note that changing synth type in the music window will affect the midi-velocity scaling.

`Save Performance MIDI file 1 As...`

Will save the current music in a midifile type 1 with the same midi-codes as when it is played (default extension ".mid"). Note that changing synth type in the music window will affect the midi-velocity scaling.

`Exit (Quit on File menu on Mac)`

Exits the Director Musices application. Make sure that there are no unsaved files since the program currently never checks for that.

Edit menu (in the Utilities menu on Mac)

`Tempo...`

Changes the score tempo of the loaded music. All durations are scaled accordingly, implying that the performance is unaffected. Note that it is important to use the right tempo when the rules are applied since they are sensitive to absolute durations.

`Octave...`

Octave transposition of the Active tracks.

`Meter...`

Changes the meter variable (mm) on the first note of each track, remove all old bar marks and attempts to mark the new bars using the rebar function.

`Key...`

Changes the key and modus variable (key, modus) on the first note of each track. The key information is used by the Harmonic charge rule. The printing of the music is not affected.

Rules menu

`Load rules...`

Loads a rule definition file (default extension ".rul")

`Open rule palette...`

Opens a rule palette file (default extension ".pal") This opens a window where to interactively work with the rules. The rule names can also be changed directly in the window.

`Open default rule palette`

Opens a rule palette window with default rule settings. This is opened at startup.

Display menu

`Score window`

Opens a score window. Note variables can be printed below each music system according to a selection made by clicking "Show vars…" By double-clicking on a note in the score, the variables on that note can be edited manually. When the "x-axis: ndr(dr)" is checked, the spacing between the notes are scaled proportional to nominal duration (i.e. score position). When it is unchecked the spacing is proportional to real duration (i.e. time). If the note property n is missing it displays a piano roll kind of bar instead of the note symbol. This applies to all graphs.

`delta Sound Level`

Draws a graph of the difference in sound level from the default value. Observe that the decibel values are only accurate for the specified synthesizer in the track window. Also, when the velocity sensivity in the synthesizer can be changed, its need to be set according to the settings in the selected synthesizer object.

`delta Volume`

Draws the volume deviation in decibel from a default value.

`delta Tempo`

Draws the tempo difference from the specified tempo.

`delta Duration (ms)`

Draws the duration difference from nominal duration expressed in milliseconds. (dr-ndr)

`delta Duration (%)`

Draws the duration difference from nominal duration expressed in percent of each tone's duration.

`Offtime duration (ms)`

Draws the offset to next onset duration (i.e. the pause duration) expressed in milliseconds. (dro)

`Offtime duration (% of IOI)`

Draws the offset to next onset duration (i.e. the pause duration) expressed in percent of the note's interonset duration. (dro)

`Duration`

Draws the duration expressed in milliseconds. (dr)

`Bar duration`

Draws the duration of each bar expressed in milliseconds.

`Beat duration (ms)`

Draws the duration the duration of each beat expressed in milliseconds.

`Beat duration (%)`

Draws the duration the duration of each beat expressed in percent of the nominal duration.

`delta Cent`

Draws the cent deviation from the equal tempered scale. Observe that the right pitch bend range has to be set in the synthesizer (usually 2 semitones).

`Vibrato amplitude`

Draws the vibrato amplitude expressed in +- peak deviation in cent. Observe that few of the synthesizers defined in DM have been calibrated for vibrato variables.

`Vibrato frequency`

Draws the vibrato frequency expressed in Hertz

Play menu

`Play preferences... (not activated)`

`Play`

The normal playing function. If rules have been applied, it will play the resulting performance. Technically, it first generates a list that is sorted by time order before the actual playing. Currently in the PC version a midifile is generated and the mediaplayer in the system is activated. The same function is used for playing from the Rule palette window.

`Playlast (Mac only)`

Plays the last list generated from "play"

`Print playlist`

Prints out the last generated playlist in the Listener window (Console window on PC). Used mostly for debugging.

`All notes off (Mac only)`

Sends the all notes off command to the synthesizer objects in the track window. Tricky since many synthesizers don't behave as they should. To stop the currently played music push Command-. , wait until a note hangs and then All notes off or Command-- (I will make a stop command soon!)

Tools menu (Utilities menu on Mac)

`Exporter… (only Windows)`

Used for exporting variables to a text file, suitable for input to e.g. an Excel spreadsheet.

`Export individual rule data…`

A tool for exporting the deviation from each rule in the top-most rule palette as a text file. This is used as input to the Expressive Director module for EyesWeb developed by Padova University.

`Convert chord list to chord name`

Changes the format of the harmonic analysis from chord list <(“C” “E” “G”)> to chord name <“Cmaj”>

`Distribute chord analysis`

A “q” variable found in one track is distributed to all simultaneous notes in the other tracks.

`Distribute phrase analysis`

A “phrase-start” or “phrase-end” variable found in one track is distributed to all simultaneous notes in the other tracks.

`Rebar`

Removes all old barlines (bar) and attempts to mark new barlines using any found meter variable. If the nominal durations of the notes not add up to the computed bar duration according to Meter and Tempo, a bar is not marked at that position. Observe that the barlines are mostly of cosmetic value since most of the current rules are unaffected by the barline position.

`Make Radio-Baton sync track`

Creates a new track that is used for controlling the playback in the Radio-Baton conduct software. The track is made of note values according to the meter and with the pitches according to the conduct program.

`Print all score vars`

Prints in the Listener window (Console window on PC, found in the lower right corner) all track and note variables sequentially. This is mostly for debugging.

`Print all score vars round`

Same as above but with all decimals rounded.

`DM var settings… (only Windows)`

Here most of the global parameters can be changed. This is not documented so it is mostly useful for the programmers.

Music File Format

The music is organized in tracks that consist of a list of segments (or notes). All parameters are associated with a segment. A segment can have several pitches, but have only one duration. Examples of the file format can be studied by open any ".mus" file in a text editor.

| means logical *or*

; means comment

The present format is a normal text file formatted in the following way:

```
<track-type>
<track-variable> <value>
<track-variable> <value>
.
.
.
(<var-name> <value> < var-name> <value> ...)     ;each parenthesis corresponds to a segment
(<var-name> <value> < var-name> <value> ...)
.
.
.
<track-type>
<track-variable> <value>
<track-variable> <value>
.
.
.
(<var-name> <value> <var-name> <value> ...)
(<var-name> <value> <var-name> <value> ...)
.
.
.
```

In the old format (still supported) a new track starts with the track name followed by the segments:

```
"<track name>"
(<var-name> <value> <var-name> <value> ...)
(<var-name> <value> <var-name> <value> ...)
.
.
.
"<track name>"
(<var-name> <value> <var-name> <value> ...)
(<var-name> <value> <var-name> <value> ...)
.
.
.
```

```
track-type is     mono-track | voice-track | multi-track
```

Track Variables

variable **value**

```
:trackname              "<string>"
:midi-channel           1-16
:midi-initial-volume    0-60 (dB)
:midi-initial-program   1-127
:synth                  "<synth-name>"    ;synth-name must be one of the
                                          predefined types in DM
```

Input music notation variables

These are the variables used for the representation of the input score. This set of variables is obviously not sufficient for describing a score. Many things are missing such as articulation marks. However, it is easy to extend the input notation with new variables and write rules that convert then into performance variables.

variable **value**

```
n           ("<tone><octave>" . <notevalue>) |
                  (("<tone><octave>" "<tone><octave>" ...) . <notevalue>)
            tone = C | C# | Db | D | D# | Eb | E | Fb | E# |F | F# |
                    Gb | G | G# | Ab | A | A# | Bb | B | Cb | B# |
            octave = 0 ... 9
            notevalue = 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128
            ;the representation of a note
            ;if several pitches, the notes should be in decending order of pitch

dot         1 | 2

tie         t
            ;tie to next note

tuple       3 | 5 | <tuple-list>
            3 ;triplet  => ndr = ndr * 2 / 3
            5 ;quintuple => ndr = ndr * 4 / 5
           <tuple-list> = (<mul> <div>) ;=> ndr = (ndr * <mul> / <div>)
            ;Must be inserted for each note it applies to.
            ;old symbol: t
```

;Alternative representation of note values used by the MIDI file input reader:
; no need for tuples, dotted or tied notes.

```
n           ("<tone><octave>" <ratio> <ratio>...)
            | (("<tone><octave>" "<tone><octave>" ...) <ratio> <ratio>...)
            ;each ratio is a notevalue, eg, 1/4, 5/7, 2/3
            ; In this way any fraction or subdivision of the beat can be represented
            ; the other notevalue variables are not used: dot tie tuple


rest        t
            ;must also have n (() . <notevalue>)

slur        t
            ;slur to next note
```

```
dyn          ppp | pp | p | mp | mf | f | ff | fff
             ;Dynamics

mm           <real number>
             ;Metronome tempo in quarters/minute.

meter        (<mul> <div>)
             mul = 1 ... 16
             div = 1 | 2 | 4 | 8 | 16
             ;ex: meter (4 4) means 4/4

key          "<chord-name>"
             chord-name = "Cmaj" | "F#min" | ... ;not implemented
             chord-name = "C" | "F# " | …

modus        "maj" | "min"
             ;The modality of the key.

q            <chord-name> | ("<tone>" "<tone>" ...)
             chord-name = "Cmaj" | "F#min" | ...
             tone = see n above
             ;This is the harmonic analysis as either a name or a list of the chord notes beginning
             ;with the bass (or the root)
             ;A Cmaj chord: q ("C" "E" "G")

bar          <integer>
             ;The bar number beginning from 1.
             ;Will be inserted if not present.
             ;Optional

pr           <1..128>
             ;MIDI program number
             ;Optional.

rit-start    t
rit-end      t
             ;ritardando start and end
              ;the note marked with rit-end should have a mm mark as well

acc-start    t
acc-end      t
              ;accelerando start and end
             ;the note marked with acc-end should have a mm mark as well

phrase-start (<level> <level> ...)
phrase-end    (<level> <level> ...)
             level = 1...7
             ;The levels are the hierarchical level of the analyzed
             ;phrase structure. 7 is the lowest level representing
             ;short melodic motives. 6 is the next level above.
              ;The format itself permits overlapping phrases.

legato-start      t
legato-end        t
             ;legato start and end
```

```
staccato          t
```
;staccato on single notes

```
pedal       0 | 1
```
;piano pedaling
;a value of 1 means pedal down and 0 pedal up
;a value of 0 needs only be inserted when there is no down pedal following

Music performance variables

These are the properties generated and altered by the rules and are used for playing.

variable **value**

```
f0          <integer number>
```
;midi note number

```
dr          <real number>
```
;Total interonset duration in ms.
;This quantity is used for playing and is changed by the rules.

```
ndr         <real number>
```
;Nominal interonset duration in ms.
;Will not be altered by the rules.

```
dro        <integer number>
```
;Offtime duration. i.e. offset to next onset.
;Optional.

```
sl          <real number> | <time-shape object>
```
;Sound level in decibel relatively a constant found in the synth object.
;Default value is sl=0.

```
dc          <integer number> | <time-shape object>
```
;Pitch deviation in cent from equal temperament
;Optional

```
va          <integer number> | <time-shape object>
```
;Vibrato amplitude in cent (+ -).
;Optional

```
vf          <real number> | <time-shape object>
```
;Vibrato frequency in Hertz.
;Optional

```
vol        <real number> | <time-shape object>
```
;volume in dB
;range: 0 to -64 dB
;will be sent as midi volume
;Optional

```
pedal-perf  0 | 1 | <time-shape object>
```
;piano pedaling with time table

;value 1 means pedal down and 0 pedal up

Speech notation variables

```
text   "<text>"
```
;input original text

Speech performance variables

```
ph     "<phoneme>"
       phoneme = a | a: ......
```

Performance Rules

Following is a comprehensive list of the performance rules. All rules (whenever applicable) has a *quantity parameter k*, which controls the general effect. A k value of 1 corresponds approximately to a normal application of the rule. This is, however, dependent on the musical context. Negative k values can be used for the reversed effect. For example, the rule Duration-contrast can be used both with negative and positive values depending on the musical intention. The *key parameters* can be used for additional control of the rules. In cases where the rule changes several performance variables, the key parameters can be used for controlling the amount of change for each performance variable. The rules are described in detail in different publications, see the references and bibliography list below. Most of the rules are discussed in [4]. There is technical description of many rules in [8]. An overview is given in [21]. The references in the tables below refer primarily to the technical description. A complete list of publications related to the rules, including a few online papers, is found on the web: http://www.speech.kth.se/music/publications/ruleref.htm

Phrasing

Rule name	**Affected performance variables**	**Key parameters** and default values	**Short description**	**Implemented in DM 2.5**
Punctuation	dr dro	:dur 1 :duroff 1 :markphlevel7 nil	Automatically locates small tone groups and marks them with a lengthening of the last note and a following micropause. [1]	x
Phrase-articulation	dro dr	:phlevel 5 :subphlevel 6 :dur 1 :duroff 1	Micropauses after phrase and subphrase boundaries, and lengthening of the last note in phrases. [13]. The phrase boundaries must be marked in the score.	x
Phrase-arch	dr sl	:phlevel 7 :power 2 :amp 1:next 1 :2next 1 :turn 2 :last 1 :acc 1	Each phrase is performed with an arch-like tempo curve: starting slow, faster in the middle, and ritardando towards the end. The sound level is coupled so that a slow tempo corresponds to a low sound level, see [5] for an explanation of the key parameters. The phrase boundaries must be marked in the score.	x
Phrase-ritardando		:shape :x2 :power 3.5 :amp 2 :phlevel 4 :next 1.5 :2next 2 :turn –2500 :last 1 :acc 0 :accfn 'power-fn-acc :decfn 'power-fn-dec	An attempt to make Baroque type of phrasing with only a slight ritardando at the end of each phrase. It applies the Phrase-arch rule with the listed default key parameters. Note that the negative turn parameter specify ritard start in ms from the end of the phrase. Not formally tested.	x
Final-ritard	dr	:q 3	Ritardando at the end of the piece similar to a stopping runner. [3]	x
High-loud	sl		The higher pitch the louder. [8]	x

Harmonic and Melodic Tension

Rule name	**Affected performance variables**	**Key parameters** and default values	**Short description**	**Implemented in DM 2.5**
Melodic-charge	sl dr va	:amp 1 :dur 1 :vibamp 1	Emphasis on notes remote from the current chord. [8]	x
Harmonic-charge	sl dr	:amp 1 :dur 1 :vibfreq 1	Emphasis on chords remote from the current key. [8]	x
Chromatic-charge	dr sl		Emphasis on notes closer in pitch. Intended primarily for contemporary atonal music. [9]	x

Metric patterns and grooves

Rule name	**Affected performance variables**	**Key parameters** and default values	**Short description**	**Implemented in DM 2.5**
Inegales	dr		Makes long-short patterns of consecutive eighth notes [8]. Also called swing eighth notes. Use no-sync in rule window. k=1 gives a ratio of 1.56:1.	x
Ensemble-swing	dr	:solo :bass :drums	Swing eighth notes proportional to tempo. Soloist and drums approx. synchronized at offbeat. Key parameters specify track names (e.g. :solo "Melody") Partly described in [14]. Use no-sync in rule palette window	x
Offbeat-sl	sl		Increase sound level at offbeats. K=1 gives 3dB increase. Not yet published.	x
Double-duration	dr		Decrease the duration contrast for two notes with the duration relation 2:1 (e.g. a quarter followed by an eighth). [8] k=1 gives a ratio of 1.68:1.	x
Triple-duration	dr		Decrease the duration contrast for two notes with the duration relation 3:1 (e.g. a dotted eighth followed by a sixteenth). Not tested.	x
Note-triplet-duration	dr		Increase contrast between a note and a following triplet of the same total nominal duration. Tentative	x

Articulation

Rule name	**Affected performance variables**	**Key parameters** and default values	**Short description**	**Implemented in DM 2.5**
Leap-articulation-dro	dro		Micropauses in leaps [8]	x
Repetition-articulation-dro	dro		Micropauses in tone repetitions [8]	x
score-legato-art	dro		Tone overlaps for notated legato [23]	x
score-staccato-art	dro		Micropauses for notated staccato [23]	x

Timing

Rule name	**Affected performance variables**	**Key parameters** and default values	**Short description**	**Implemented in DM 2.5**
Duration-contrast	dr sl	:amp 1 :dur 1	The longer note the longer and louder, and the shorter the shorter and softer. [8]	x
Duration-contrast-art	dro		The shorter note the longer micropause.[2, 22]	x
Social-duration-care	dr		Increase duration for extremely short notes [9]	x
Faster-uphill	dr		Decrease duration for notes in uphill motion.[8]	x
Leap-tone-duration	dr		Shorten the first note of an ascending leap and lengthen the first note of a descending leap. [8]	x

Ensemble timing

Rule name	**Affected performance variables**	**Key parameters** and default values	**Short description**	**Implemented in DM 2.5**
Melodic-sync (no k parameter)	dr		Generates a new track consisting of all tone onsets in all tracks. When there are several simultaneous onsets, the note with the maximum melodic charge is selected. All	x

			rules are applied on this sync track and the resulting durations are transferred back to the original tracks. [8, 10]	
Simple-mel-sync (no k parameter)	dr		Generates a new track as above but uses always the top note in chords. Pauses are treated as notes. Is more reliable and faster for larger scores.	x
Bar-sync (no k parameter)	dr		Synchronize the tracks on each barline.	

Intonation

Rule name	**Affected performance variables**	**Key parameters** and default values	**Short description**	**Implemented in DM 2.5**
High-sharp	dc		The higher pitch the higher [8]	x
Mixed-intonation	dc		Each note is initially intonated as melodic intonation and then gradually changed to harmonic intonation [10]	
Harmonic-intonation (no k parameter)	dc		Beat-free intonation of chords relative the root.	x
Melodic-intonation	dc		Close to pythagorean tuning with e.g. sharp leading tones. [8]	x

Piano specific

Rule name	**Affected performance variables**	**Key parameters** and default values	**Short description**	**Implemented in DM 2.5**
Pedaling	Pedal-perf		Performs piano pedaling from the input variable Pedal.	

Noise

Rule name	**Affected performance variables**	**Key parameters** and default values	**Short description**	**Implemented in DM 2.5**
noise	dr, sl	:amp 1 :dur 1	Generates random variations of inter-onset durations and soundlevel. Modeling variations in the internal clock and the motor system of a performer. [24]	x

Utility rules

Rule name	**Affected performance variables**	**Key parameters** and default values	**Short description**	**Implemented in DM 2.5**
Normalize-sl (no k parameter)	sl		Normalize the sound level over each track in the piece	x
Maximize-sl (no k parameter)	sl		Translate the sound level so that max sl corresponds to max midi velocity. The result depends on the selected synthesizer.	x
Set-Sound-Level	sl		Translate the sound level. K=1 gives 3 dB increase	x

Normalize-dr (no k parameter)	dr		Scale the durations so that the total length is the same as the nominal length	x
Normalize-dr-bar (no k parameter)	dr		The same as above for each measure	x
Scale-duration	dr		Scale all durations with the factor given by the k value. Note that k=1 gives no change.	x
Overall-articulation	dro		Applies a constant articulation to all notes followed by another note, thus not on notes before rest. k=1 gives 25% silence.	x

References and bibliography

See also http://www.speech.kth.se/music/performance/performance_publications.html

[1] Friberg, A., Bresin, R., Frydén, L., and Sundberg, J. (1998) "Musical punctuation on the microlevel: Automatic identification and performance of small melodic units", *Journal of New Music Research*, 1998, Vol. 27, No. 3, pp. 271-292

[2] Bresin, R. & Friberg, A. (1998) "Emotional expression in music performance: synthesis and decoding", in *TMH-QPSR, Speech Music and Hearing Quarterly Progress and Status Report*, 4/1998, Stockholm, pp. 83-92

[3] Friberg, A. and Sundberg, J. (1999) "Does music performance allude to locomotion? A model of final *ritardandi* derived from measurements of stopping runners", *J. Acoust. Soc. Am.,* 105(3) pp 1469-1484

[4] Friberg, A. (1995). *A Quantitative Rule System for Musical Performance*, doctoral dissertation, Royal Institute of Technology, Sweden

[5] Friberg, A. (1995). "Matching the rule parameters of Phrase arch to performances of Träumerei: A preliminary study", in A. Friberg and J. Sundberg (eds.), *Proceedings of the KTH symposium on Grammars for music performance May 27,1995*, pp. 37-44.

[6] Friberg A, Sundberg J & Frydén L (1994) "Recent musical performance research at KTH", in J. Sundberg (ed.), *Proceedings of the Aarhus symposium on Generative grammars for music performance 1994*, 7-12.

[7] Sundberg, J. (1993). "How can music be expressive?", *Speech Communication* 13, pp. 239-253.

[8] Friberg, A. (1991). Generative Rules for Music Performance: A Formal Description of a Rule System", *Computer Music Journal*, 15 (2), pp. 56-71.

[9] Friberg, A., Frydén, L., Bodin, L.-G., and Sundberg, J. (1991) "Performance Rules for Computer-Controlled Contemporary Keyboard Music", *Computer Music Journal*, 15-2, pp. 49-55.

[10] Sundberg, J., Friberg, A. & Frydén, L. (1989). "Rules for automated performance of ensemble music", *Contemporary Music Review*, 3, 89-109.

[11] Sundberg, J., Friberg, A. & Frydén, L. (1991). "Common Secrets of Musicians and Listeners – An analysis-by-synthesis Study of Musical Performance" in P. Howell, R. West & I. Cross (eds.), *Representing Musical Structure*, London: Academic press.

[12] Carlson, R., Friberg, A., Frydén, L., Granström, B. and Sundberg, J. (1989). "Speech and music performance: parallels and contrasts", *Contemporary Music Review* 4, pp.389-402.

[13] Friberg, A., Sundberg, J. & Frydén, L. (1987). "How to terminate a phrase. An analysis-by-synthesis experiment on the perceptual aspect of music performance", in A. Gabrielsson (ed.), *Action and Perception in Rhythm and Music*, Stockholm: Royal Swedish Academy of Music, Publication No. 55, pp. 49-55.

[14] Friberg, A. and Sundström, A. (1997) "Preferred swing ratio in jazz as a function of tempo", *Speech Music and Hearing Quarterly Progress and Status Report*, 4/1997, pp. 19-28.

[15] Frydén, L., Sundberg, J. & Askenfelt, A. (1988). "Perception aspects of a rule system for converting melodies from musical notation into sound", *Archives of Acoustics* 13 (3-4), pp. 269-278.

[16] Sundberg, J. (1988). "Computer synthesis of music performance", in J. A. Sloboda (ed.), *Generative Processes in Music*.

[17] Sundberg, J. and Frydén, L. (1984). "Teaching a computer to play melodies musically", *Analytica, Festschrift for Ingemar Bengtsson*, Publications issued by the Royal Swedish Academy of Music, Nr 47, 67-76.

[18] Sundberg, J., Askenfelt, A. and Frydén, L. (1983). "Musical performance: A synthesis-by-rule approach", *Computer Music Journal*, 7, 37-43.

[19] Sundberg, J., Frydén, L. & Askenfelt, A. (1983). "What tells you the player is musical? An analysis-by-synthesis study of music performance",in J. Sundberg, (ed.), *Studies of Music Performance*, Publication issued by the Royal Swedish Academy of Music Nr. 39, Stockholm, pp. 61-75.

[20] Thompson, W. F.,Sundberg, J., Friberg, A., and Frydén, L. (1989). "The Use of Rules for Expression in the Performance of Melodies", *Psychol. of Music* 17, 63-82.

[21] Friberg, A, Colombo, V, Frydén, L and Sundberg, J (2000) “Generating Musical Performances with Director Musices”, *Computer Music Journal*, 24(3), 23-29.

[22] Bresin, R. & Friberg, A. (2000) Emotional Coloring of Computer-Controlled Music Performances. *Computer Music Journal*, 24:4, 44-63

[23] Bresin, R. (2000) *Virtual Virtuosity - Studies in automatic music performance*. Doctoral Dissertation, Speech Music and Hearing, Stockholm: KTH, TRITA-TMH 2000:9, ISSN 1104-5787, ISBN 91-7170-643-7

[24] Juslin, P. N., Friberg, A., & Bresin, R. (2002). Toward a computational model of expression in performance: The GERM model. *Musicae Scientiae specal issue 2001-2002*, 63-122.

Playing

Currently in the PC version, a midi file is written to a temporary directory and the media player is started. The file mplayer2.exe must be in the same directory as where DM starts. In the Macintosh version, the Apple Midi Driver is used. The playing function uses the performance variable dr for the total duration of a note until the next note onset and dro for the offtime duration. The playing function also sends all the other performance variables as listed above before note on.

Synthesizer Objects

The performance variables are sent to the selected synthesizer object in the track window. The synthesizer object translates the performance variables to MIDI-codes depending on the selected synthesizer. The purpose of this transformation is to have synthesizer independent parameters in the program that is expressed in physical measures. Most of the synthesizers listed in the track window have calibrated functions for soundlevel to MIDI velocity and for volume to MIDI volume. This assumes certain settings in the synthesizer since for example velocity scaling often is available as a parameter in synthesizer patches. The pitchbend range in the synthesizer object is usually 2 semitones. The durations will always be played right but if the sound chosen in the synthesizer has long attack and/or decay times, the micropauses will not be properly performed.

Pinnacle

General midi module from Kurtzweil found on the Pinnacle PC sound board.

SBlive

Sound Blaster live PC sound card.

Roland-A90

The optional sound module in Roland A90 keyboard controller.

Roland-PMA5, Roland-1010

General midi synthesizers

Technics SX-P30

A piano keyboard. MIDI velocity is not recognized.

Proteus

Sample player from e-mu. Close to the factory settings in Proteus/2 especially the woodwinds. Assumes a pitchbend of ±2 semitones, vel curve 2. Vibrato parameters are not included.

SampleCell

Sample card for Mac. Included in the DM Mac version are some patches in SampleCell that are adapted to Director Musices parameters. They assume that the samples are on the basic CD-ROM shipped with SampleCell. A good way to start is to connect it to SampleCell with the FB01 organ sound loaded. Uses pitchbend ±1 semitone, vibrato extent as midi pressure and vibrato speed as the modulation wheel.

S3000

Sampler from AKAI. Pitchbend ±2 semitones, vel sensivity +20 (factory setting).

FZ1

Old sampler from Casio.

Fb01, dx21

Old Yamaha synthesizers.

Musse

Singing synthesizer developed at Speech, Music and Hearing, KTH.

Generator

A general purpose software synthesizer from Native instruments. Now called Reactor. Only used for patches developed at the department.

For non-scientific purposes it can be sufficient to try the different synthesizer objects (for example SBlive) and choose the one that has a good balance of both soundlevel and duration changes.

We can implement new synthesizer objects for your synthesizers if you help us with the measurements. Contact us for further info.

Writing New Rules

The following is a partial list of utilities for rule definition. The tools are built within the common lisp environment. Therefore, the syntax is the same as in common lisp (ANSI CL for PC and close to ANSI for Mac) and all common lisp resources are available. The serial note macros are the ones mostly used in the past and are therefore more robust. The recent extensions allowing grouping of segments and notes have not been very much tested yet.

Music Structure

The music is organized in tracks that can be processed either serially or in parallel. Each track consists of a list of notes, each of which has a list of variables. These variables can be the duration, the sound level, the MIDI key number or other properties that have been assigned to the notes. The assignment of such variables is normally realized by the rules by means of access functions, but the value can also be edited manually if the music is saved in a performance file (or in the score window on PC).

Example of a variable list:

```
(dr 793.0  dro 54 ndr 854.0  sl 10  f0 50  bar 1  phrase-start (4)  key "D"
modus "maj")
```

This means that the note has interonset duration = 793.0 ms, "off-time" duration = 54 ms, nominal interonset duration = 854.0 ms, sound level 1 dB over middle value, MIDI key number = 50, bar #1, A phrase beginning on phrase level 4, and that the key is D major. A more complete description of variables is given in the section Music File Format above.

Rule top-level definition

Rule are defined by the normal lisp defun function using the format specified below. These are the formats recognized by the rule palette window.

```
(defun <rulename> (<k parameter> <additional key parameters>)
   <body>)
```

Defines a rule with the main rule parameter k. The k parameter should be included in all changes of performance variables within the rule. For example, a doubling of k should correspond to a doubling of the *changes* made by the rule. Additional parameters are specified using key parameters.

```
(defun <rulename> (<additional key parameters>)
   <body>)
```

Defines a rule without a k parameter. Additional parameters are specified using key parameters.

Serial Rule Macros

These macros will step through the score in chunks as specified by each macro and are used within the body of a rule definition. For example, each-note will consider all segments up to next note marker as one chunk.

```
(each-track <body>)
```

Evaluate the <body> for each track in the score. Within the body, access functions are used to access the variables of the tracks.

```
(each-note <body>)
```

Evaluate the <body> for each note and track in the score serially. Within the body, access functions are used to access the variables of the notes.

```
(each-note-if
   <condition 1>
   <condition 2>
   <condition 3>
   .
   .
   .
   (then
     <body>)
```

This is the same as each-note with the addition of conditions. For each note, all conditions must be true in order to evaluate the body.

```
(each-segment <body>)
(each-segment-if <body>)
```

Evaluate the <body> for each segment and track in the score serially. Within the body access functions are used to access the variables of the segments. These macros have the same function as each-note and each-note-if if the track is a mono-track. For a voice-track these macros works on a lower level, each segment corresponding to a voice segment or a phoneme.

```
(each-group
  `<group begin condition>
  `<group end condition>
  <body>)
```

Evaluate the <body> for each group of segments as specified by the begin and end conditions. Within the body, access functions are used to access the variables of each group as one chunk.

```
(each-group-if
  `<group begin condition>
  `<group end condition>
  <group condition 1>
  <group condition 2>
  <group condition 3>
   .
   .
   .
   (then
     <body>)
```

This is the same as each-group with the addition of conditions. First a new track is created consisting of segment group objects (chunks). Then for each chunk, all group conditions must be true to evaluate the body for that chunk.

If an each-track macro is omitted the other macros are iterating over each track as well.

```
(each-note
```

is equivalent to

```
(each-track
    (each-note
```

Parameters available within macros

`*v*`

Contains the current list of objects

`*i*`

Contains the current object index (zero to length-1)

Serial Access Functions

Within the body of the sequencing macros, the access functions are used for accessing the variables in each chunk. The access functions manipulate normally the data for the whole chunk. They are also used for context definitions. A slowly changing variable (time-shape object) can be added over a whole chunk.

Any variable

```
(this <variable>)
(next <variable>)
(prev <variable>)
```

Return the specified variable of the current, next, or previous chunk. Also available are next2, next3, prev2, prev3.

```
(set-this <variable> <value>)
(set-next <variable> <value>)
(set-prev <variable> <value>)
```

Assigns the specified value to the variable in the current, next and previous chunk. Also available are set-next2, set-next3, set-prev2, set-prev3.

```
(add-this <variable> <value>)
(add-next <variable> <value>)
(add-prev <variable> <value>)
```

adds the specified value to the variable in the current, next and previous chunk. Also available are add-next2, add-next3, add-prev2, add-prev3.

```
(rem-this <variable>)
(rem-next <variable>)
(rem-prev <variable>)
```

removes the specified variable. Also available are rem-next2, rem-next3, rem-prev2, rem-prev3.

Duration

```
(this-dr)
(next-dr)
(prev-dr)
(next2-dr)
(prev2-dr)
```

```
(next3-dr)
(prev3-dr)
(set-this-dr <value>)
(set-next-dr <value>)
(set-prev-dr <value>)
(set-next2-dr <value>)
(set-prev2-dr <value>)
(set-next3-dr <value>)
(set-prev3-dr <value>)

(multiply-this-dr <factor>)
(multiply-next-dr <factor>)
(multiply-prev-dr <factor>)
(multiply-next2-dr <factor>)
(multiply-prev2-dr <factor>)
(multiply-next3-dr <factor>)
(multiply-prev3-dr <factor>)
```

Pitch

If f0 is a list (chord) these functions return only the first tone.

```
(this-f0)
(next-f0)
(prev-f0)
(next2-f0)
(prev2-f0)
(next3-f0)
(prev3-f0)
```

Accessing any note

```
(iget <index>)
(iset <index> <value>)
(iadd <index> <value>)
```

Destructive segment operations

```
(remove-this-segment)
(insert-segment-before-this <new-segment>)
```

Use these functions with caution using simple contexts as they destructively modify the segment-list e.g. (each-segment-if (first?) (remove-this-segment)) will remove ALL segments

Utilities

```
(first?)
(last?)
```

Returns true if it is the first or last chunk, respectively, otherwise returns false. Also first+1 and last-1?.

```
(i?next <index> <variable>)
```

Returns the index of the chunk containing variable after chunk with index < index>. If not found ->nil(i?prev <index>)Returns the number of the chunk with variable before chunk with index < index>. If not found ->nil

`(i?last)`

Returns the index of the last chunk.

`(exit-track)`

Go to the end of the track.

`(i?next-note <index>)`

Returns the index of next note excluding rests. Not found -> nil.

`(i?prev-note <index>)`

Returns the index of previous note excluding rests. Not found -> nil.

`(sharp?)`
`(flat?)`

Returns true if the current note is sharp or flat, respectively.

`(i?drsum <index> <total duration>)`

returns the index of the chunk up to total duration included the first note specified by index
returns at least index+1
if it gets to the end - returns last

`(ndrsum <i-from> <i-to>)`
`(drsum <i-from> <i-to>)`

returns the sum of the durations (float) not included i-to

Examples

```
(defun phrase-rule (k)        ;a complete rule for lengthening notes
    (each-note-if             ;before phrase-start markers
        (not (last?))
        (next 'phrase-start)
        (then
           (add-this 'dr (* 40 k))
         )))


(each-track
      (set-this-dr              ;increase the duration for the whole track
          (* (this-dr) 1.2) )) ;with 20 %


(each-note-if                   ;process this note if:
   (< (this 'dr) 500)           ;it is shorter than 500 ms
   (> (this 'f0) (prev 'f0))    ;and if the pitch is higher than previous
   (then
      ...

(each-group                         ;process phrase by phrase
   '(this 'phrase-start)                  ;group beginning
   '(next 'phrase-end)              ;group end
   (then
     (set-this                      ;increase the sound level with an
       'sl                          ;envelope over the phrase
       (make-time-shape ...
```

```
(each-group                                ;similarly process phrase by phrase
   '(this 'phrase-start)                   ;group beginning
   '(or (last?) (next 'phrase-start))      ;group end, remember not specify
                                           ;context after the last segment
   (then
     (print (this-dr))                     ;print the duration of each group
     (set-this 'sl (- *i*)) )))            ;set the sound level to each group
                                           ;index
```

Time shapes

The time shape objects has recently been developed and will be documented in the next version.

Parallel Rule Macros

This macros step in parallel through the tracks one step for each new note in any track

`(p-each-bar <body>)`
Executes the body each time there is a new bar in the music (not implemented)

`(p-each-note <body>)`
Executes the body each time there is a new note event in the score

Parameters available within macros

For every step in the parallel rule macros, the following variables are set:

`*all-notes*`
A list of all notes (track-object and segment index) contained within the current position (alist)

`*cur-notes*`
all notes that is new for each step (alist)

Parallel Access Functions

`(p-this)`
returns an alist of the tracks and variable values in *cur-notes*

`(p-set-this <value>)`
Sets a variable value for each note in *cur-notes*

`(p-this-all <variable>)`
returns an alist of the track and variable value for each track and note in *all-notes*

```
(v-iget (<track> . <index>) <variable>)
```
get the variable value for the specified note

```
(v-iset (<track> . <index>) <variable> <value>)
```
set the variable value for the specified note

```
(v-this <track>  <variable>)
```
get the variable value in the current note in the specified track

```
(v-this-all <track>  <variable>)
```
get the variable value from current or last current in the specified track

```
(v-this-plist <track>)
```
get the variable list in the current note in the specified track

```
(v-set-this <track> <variable> <value>)
(v-add-this <track> <variable> <value>)
(v-set-next <track> <variable> <value>)
(v-set-prev <track> <variable> <value>)
```
Set the variable value in the current, next or previous note in the specified track

Parallel Utility Functions

```
(p-first?)
(p-last?)
```

```
(v-i?next (<track>  . <index>) <variable>)
```
Returns a pair with the note index and variable after note i. If not found ->nil

```
(v-i?next-or-last (<track>  . <index>) <variable>)
```
Returns a pair with the note index and variable after note i. If not found -> last note

```
(v-i?last <track>)
```
Returns the number of the last note

```
(i?-barnr <track> barnr)
```
return note number from track and bar number else nil

```
(p-min pair-alist)
```
return minimum

if same: take only one

```
(p-min-all pair-alist)
```
if same rounded return a list of those

```
(p-max pair-alist)

(v-to-ni <track>)
```
