Migrating Musical Concepts: An Overview of the Bol Processor

Bernard Bel

Migrating Musical Concepts — An Overview of the Bol Processor

Bernard Bel
CNRS (France) — Centre de Sciences Humaines (CSH)
2, Aurangzeb road, New Delhi 110 011, India

The Bol Processor is the outcome of a migratory process, its design having been carried over in three phases and places: in collaboration with traditional North Indian musicians (1980-85), Western musicians in Europe (1985-93) and back in India with Carnatic musicians (1995-97).

The theoretical framework of the underlying research project also evolved in three stages, taking inspiration from *expert systems* in the early 1980s, *symbolic-numeric machine-learning* in the end of the decade, and *composition theory* (Laske 1989, 1992) in the 1990s.

Throughout this process, the designer has been faced with the challenge of blending software with "mindware", here taken to mean musicians' striving for *[…] tools enabling them to manipulate objects so as to imbue them with 'soul' or experiential value […]* (Laske 1996). In a cross-cultural approach this led to modelling descriptions of music and compositional processes at a level of abstraction sufficiently high to encompass "local" musical concepts without getting too abstruse.

Phase 1: setting-up the expert system

In 1982, ethnomusicologist Jim Kippen used to carry an Apple IIc, the ancestor of laptop computers, to the homes of tabla players in North India. He requested the author to design a word processor enabling him to type rhythmic data at performance speed — up to five strokes per second. These strokes are conventionally designated with semi-onomatopoeic syllables named *bol* (from the Hindi/Urdu *bolna*, "to speak"): 'dha', 'tirakita', 'dhin', 'ge'…

Bol Processor "BP1" resembled the AppleWriter program with a customised mapping of the keyboard to the vocabulary of *bol*s. The next step in its design was the automation of "find-replace" operations aimed at experiments on substitutions of word sequences. This led to the implementation of an *inference engine* handling formal grammars, an approach based on Chomsky's linguistics (Révész 1985). Given a grammar, the inference engine picks up rules at random to rewrite a string of variables and terminal symbols derived from the conventional starting symbol 'S'. This process goes on till the exhaustion of candidate rules (Roads 1984:16-21; Bel and Kippen 1992).

BEL, B. Migrating Musical Concepts: An Overview of the Bol Processor. Computer Music Journal, vol. 22, no. 2. 1998, p. 56-64.

Examples of grammars modelling improvisation on the tabla have been published by Kippen and Bel (1989a, 1992).

Phase 2: modelling compositional tasks

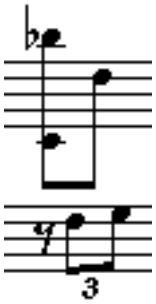
In the late 1980s, interaction with Western musicians led the author to revise the research agenda, focusing on explorative, rather than analytical, approaches. This followed experiments with a machine-learning device infering formal grammars from sets of musical examples. Inductive inference had been successfully tried on sample sets of tabla music, but when it came to real performance this method highlighted the need of background knowledge which only a consistant theory of compositional and improvisational strategies could provide (Kippen and Bel 1989b).

The Bol Processor was ported to the Apple Macintosh and the new version was named "BP2." The alphabet of terminal symbols (*bol*s) was mapped to *sound-objects* containing sequences of MIDI messages, and complemented with *simple notes* in English, French, Indian and key-number notation. These notes can be captured from MIDI devices. A 440Hz tone is respectively notated 'A4', 'la3', 'sa4' or 'key#69' in the four conventions.

The initial inference engine of Bol Processor was handling context-sensitive rules in formal grammars. To solve new musical problems it became necessary to implement a check of remote and even "negative" contexts. In addition, BP2's syntactic model accepts meta-variables (wild cards), repetition patterns, logic-numeric flags controlling the inference, and meta-grammars producing grammars (Bel and Kippen 1992; Kippen and Bel 1992).

The polymetric expansion algorithm

Migrating to Western music raised an important question: how can polyphonic structures be represented in a linear text format? The answer came with *polymetric expressions* (Bel 1992:72-85). A simple polymetric expression is shown figure 1 in staff notation, BP2 graphics and *phase diagram*. The latter is a table containing pointers to the instances of sound-objects 'C4', 'E#3', etc.



1	2	3	4	5	6
C4	_	_	-	_	_
-	_	E#3	_	G3	_
A#5	_	_	_	_	_
-	_	_	D5	_	_

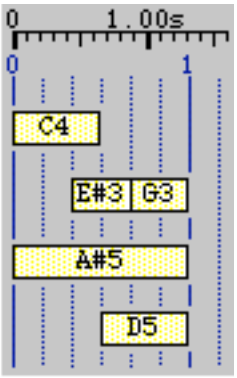


Figure 1: Staff notation, phase diagram and BP2 graphic display of a polymetric expression notated "{1,□C4□-,□-□E#3□G3,□A#5,□-□D5}". This expression is taken from a grammar producing a piece imitating Steve Reich's style. (Courtesy Thierry Montaudon)

This expression is notated "{1,□C4□-,□-□E#3□G3,□A#5,□-□D5}" on a BP2 text score. It shows five sequences (*fields*) separated by commas. The leftmost field contains '1', the *symbolic duration* of the expression. With a metronome set to 45□beats per minute, the resulting physical duration is 1.33 seconds.

The *polymetric expansion algorithm* imbedded in Bol Processor coerces all sequences of the structure to the same symbolic duration. The second field contains a note 'C4' followed by a silence '-', both of which will be treated as quavers. The third field contains the sequence "-□E#3□G3" which is performed as a triplet.

Thus, the polymetric expression "{1,□C4□-,□-□E#3□G3,□A#5,□-□D5}" is expanded to "/6□{C4_□□-_□,-_□E#3_□G3_,A#5_□□□□,-_□□D5_□}" as suggested by the phase diagram. Symbol '_' is a prolongation of the preceding sound-object, and '/6' indicates a change of tempo after which durations are divided by 6.

The polymetric expansion algorithm is called recursively by expressions containing multiple parenthesis levels. Sequences may optionally contain undetermined rests, the durations of which are computed to satisfy a constraint of minimum complexity in the resulting structure (op.cit:80).

Period notation

In the same way it deals with superimposed sequences, the polymetric expansion algorithm works out equal symbolic durations between beat separators notated '•' — the "period notation." A note sequence in period notation and the context-free grammar it originated from are shown figure 2. In this example, beats contain increasing numbers of notes resulting in an accelerating movement.

```
S --> _vel(60) A B _vel(65) C  D _vel(70) E F _vel(75) G _vel(77)
      H _vel(80) I _vel(85) J  _vel(87) K _vel(90) L
A --> E2 ·
B --> D2 A
C --> B2 B
D --> G2 C
E --> F#2 D
F --> A#2 E
G --> C2 F
H --> G#2 G
I --> A2 H
J --> D#2 I
K --> C#2 J
L --> F2 K
```

BP2 score: *Velocity controls have been left out*

```
E2 · D2 E2 · B2 D2 E2 · G2 B2 D2 E2 · F#2 G2 B2 D2 E2 · A#2 F#2 G2
B2 D2 E2 · C2 A#2 F#2 G2 B2 D2 E2 · G#2 C2 A#2 F#2 G2 B2 D2 E2 ·
A2 G#2 C2 A#2 F#2 G2 B2 D2 E2 · D#2 A2 G#2 C2 A#2 F#2 G2 B2 D2 E2
· C#2 D#2 A2 G#2 C2 A#2 F#2 G2 B2 D2 E2 · F2 C#2 D#2 A2 G#2 C2 A#2
F#2 G2 B2 D2 E2 ·
```

Figure 2: A grammar producing an accelerating sequence of notes, and the resulting item in BP2 score notation. (Courtesy: Harm Visser)

Once the grammar has been typed, the user may select "Produce items" to get a text or graphic display of its production, listen to it on the MIDI output, and optionally produce a Csound score. The output of the "acceleration grammar" of figure 2 is displayed on a piano-roll score (figure 3).

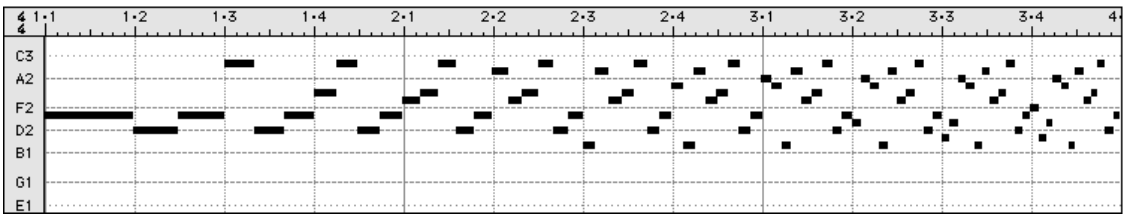


Figure 3: A piano-roll of the item produced by the grammar of figure 2.

Quantization as a response to complexity

The polymetric expansion algorithm works on integer ratios yielding absolute time accuracy. It produces an expanded polymetric expression used to construct the phase diagram. The phase diagram of figure 1 contains 6 columns or *frames*. This value was obtained as the lowest common multiple (LCM) of 1, 2, 3, the symbolic durations of sequences in the original expression.

Complexity quickly grows out of acceptable ranges. For instance, the phase diagram of the example displayed figure 3 should contain 12 x 27,720 = 332,640 frames, where 27,720 is the LCM of 1, 2, 3, …, 12. Since the

total duration is 12 seconds, the delay between two frames would be 12 / 332,640 = 36 microseconds, i.e. about 1,000 times less than a musical ear would be able to notice. Needless to say, large phase diagrams demand a great amount of computation space and time.

BP2 performs a *quantization* to reduce the size of the phase diagram before it is constructed. A reasonable time resolution, for instance 20 milliseconds, is declared by the user. The machine calculates a compression rate accordingly. In the example of figure 3 the rate is 20,000 / 36 = 555. Consequently, the simplified phase diagram contains only 332,640 / 555 = 600 frames. Quantization is performed at the symbolic level, modifying polymetric expressions before constructing the phase diagram. This method is prefered for two reasons: 1) it minimises computation time and space; 2) it ensures long-term time accuracy since rounding errors are not cumulated.

Quantization is indispensable to the performance of intricate structures (e.g. self-replicating fractals) produced by *recursive* grammars. A recursive grammar contains self-imbedding rules rewriting their own left argument to the string under derivation, or rules that rewrite each other's left argument. Thanks to the polymetric syntax, the symbolic durations of time-object structures produced by recursive grammars may be coerced to finite values regardless of the number of symbols they are made of. While recursivity allows musical structures to grow more complex in a specified time metrics, quantization is required to maintain complexity below the limit set by the time discrimination of the auditory system.

Sound-objects

The low-level components of musical structures in the Bol Processor are not conventional notes but *sound-objects*. A sound-object may contain a stream of MIDI messages, a Csound score or any combination thereof. Simple notes are predefined sound-objects reduced to a NoteOn/NoteOff pair in MIDI, or a single score line invoking a pitched Csound instrument.

The sequence of events in a sound-object and its metrical and topological properties (see infra) are stored in a template called a *sound-object prototype*. Each terminal symbol of the grammar is assigned a unique sound-object prototype. Instances of sound-object prototypes may be resized according to the metronome speed and their symbolic durations. Resizing can be prohibited or restricted to a particular range. An object may also be declared "cyclic:" beyond an introductory section the sequence of events is repeated instead of being stretched.

The perceptual and physical start/end points of a sound-object may be dissociated, as suggested by Dannenberg (1997a:50), using "preroll" and "postroll" values defined in its prototype (Bel 1997:24).

When a metronome is used the performance is set in *striated time* — Boulez' *temps strié* (Boulez 1963:107; Bel 1992:71). Accordingly, metronome beats or

their regular subdivisions are named *time streaks*. An anchoring point to the time streak, a *pivot*, is defined for each sound-object prototype.

In the sequence displayed figure 4, the sound-object 'a' behaves like simple notes 'C4' and 'D4'. Its pivot (a triangle at the top left corner) is placed on the time streak numbered '0' and its duration is exactly one beat. Sound-objects 'b' and 'c' behave in different ways. Their durations are less than one beat as they may contain sequences of events whose durations could not be dilated beyond a certain range. In addition, the pivot of sound-object 'b' is located after the first event. Consequently, the time-span of 'b' is overlapped by 'D4'.

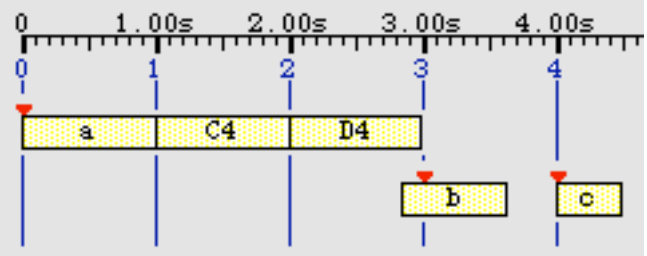


Figure 4: A sequence of five sound-objects notated "a C4 D4 b c" performed on a metronome beat (mm = 60). The horizontal axis represents physical time. Vertical positions are arbitrary. Vertical lines numbered 0, 1, 2, are the time streaks. Rectangles represent the time-span of elementary MIDI or Csound event sequences.

The time-setting algorithm

Sound-objects may be assigned topological properties pertaining to their time-span intervals in a sequence (Bel 1992:90-92). For instance, sound-object 'b' could be configured to prohibit its overlapping by the preceding sound-objects. In addition, it could be declared "non-relocatable." The time-setting algorithm would then try any of the solutions shown figure 5:

1) truncating the beginning of 'b';

2) if truncating 'b' is not allowed, it could "break the tempo," delaying by 200 milliseconds all time streaks beyond the one labelled '2';

3) if breaking the tempo is not allowed, the algorithm would attempt to relocate 'D4';

4) the same as (3) when sound-object 'c' has a property specifying that its time-span interval must be contiguous with the one of a preceding object.

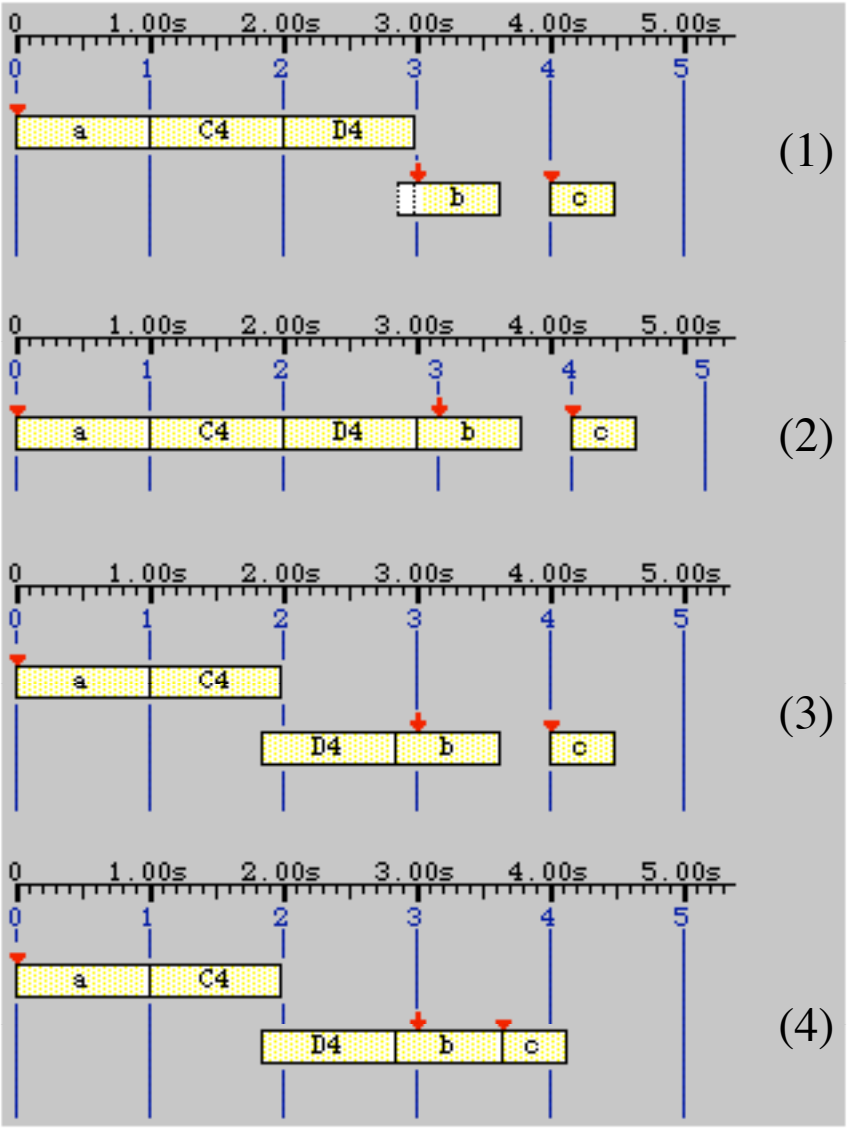


Figure 5: The sequence shown figure 4 with the additional constraint that 'b' is not relocatable. The time-setting algorithm may opt for any of the solutions displayed here. Solution (4) is the same as (3) with forced continuity between sound-objects 'b' and 'c'.

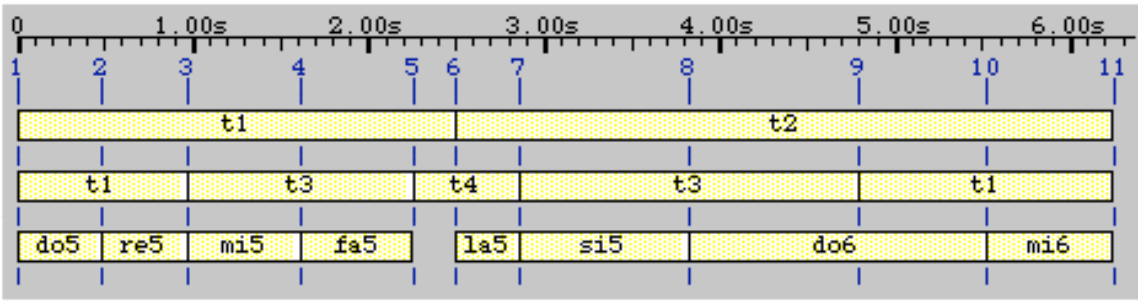
The time-setting algorithm may recursively modify the positions of all sound-objects in a sequence until all constraints are satisfied (Bel 1992:85-107). This constraint-satisfaction approach combines stipulatory top-down design at the symbolic level — formal grammars, polymetric structures — with bottom-up constructs based on sound-objects endowed with metric and topological properties.

Time patterns

In our view, an essential feature of the sonological component of music software lies in the proper distinction between "symbolic" and "physical" time, and their mapping which should take into account the plasticity of perceptual time. We use "symbolic" in a sense similar to Jaffe's (1985) "basic" time or Dannenberg's (1997b) "score" time. However, since these authors rely on numeric functions for the mapping of physical time to symbolic time (op.cit:62), they measure both with real numbers. In the Bol Processor, symbolic time is expressed by integer ratios, a format which lends itself to symbolic-numeric computation as exemplified with the polymetric expansion algorithm.

Time-stretch operators are implemented in BP2, but they work as *performance controls*. For instance, the "_legato(x)" and "_staccato(y)" tools alter the durations of sound-objects, and their numeric arguments 'x' and 'y' may be interpolated throughout a sequence and copied to sub-structures in a polymetric expression. Thus, time flexibility in BP2 is not the effect of arbitrary numeric functions. It stems out of a time structure — Xenakis' (1963) *structure temporelle* — deeply interwoven with the syntactic description of music.

The case of *time patterns* will clarify this point. In *smooth time*, the *temps lisse* of Boulez (1963:107), empty time-objects labelled 't1', 't2'… may be set-up (as integer ratios) to build a hierarchy of time patterns. Figure 6 shows a sequence of simple notes 'do5', 're5'… arranged on a lattice of time-objects setting up an irregular beat structure.




```
BP2 score:

{10,t1 t2,{t1 t3 t4,do4 re4 mi4 fa4 - la4}{t3 t1,si4 do5 _ mi5}}

Grammar producing this score:

S --> {10, t1 t2, Part1 Part2}
Part1 --> {t1 t3 t4, do5 re5 mi5 fa5 - la5}
Part2 --> {t3 t1, si5 do6 _ mi6}

TIMEPATTERNS:
t1 = 1/1   t2 = 3/2   t3 = 4/3   t4 = 1/2
```

Figure 6: A sequence of simple notes 'do5', 're5',…, arranged against a lattice of time objects 't1', 't2',…, resulting in an irregular "beat" structure in smooth time.

Symbolic time, marked by vertical lines numbered 1 to 11, is basically a graduation in which precedence relationships are checked by comparing integer ratios. There is no explicit mapping of symbolic to physical time, no "corrective factor" as the physical timing of this musical piece depends on the instantiation of time-objects imbedded in its deep structure.

Phase 3: the Csound interface

After 1994, the Bol Processor project resumed in India with a significant input from musicians conversant with computer technology (Kippen and Bel 1994). Finding no MIDI studios around was an incentive to look for new environments BP2 could fit in. Csound (Vercoe 1993) became a priority target because of its versatility and free availability on various platforms.

The interaction between BP2 and its MIDI and Csound environments is sketched out figure 7.

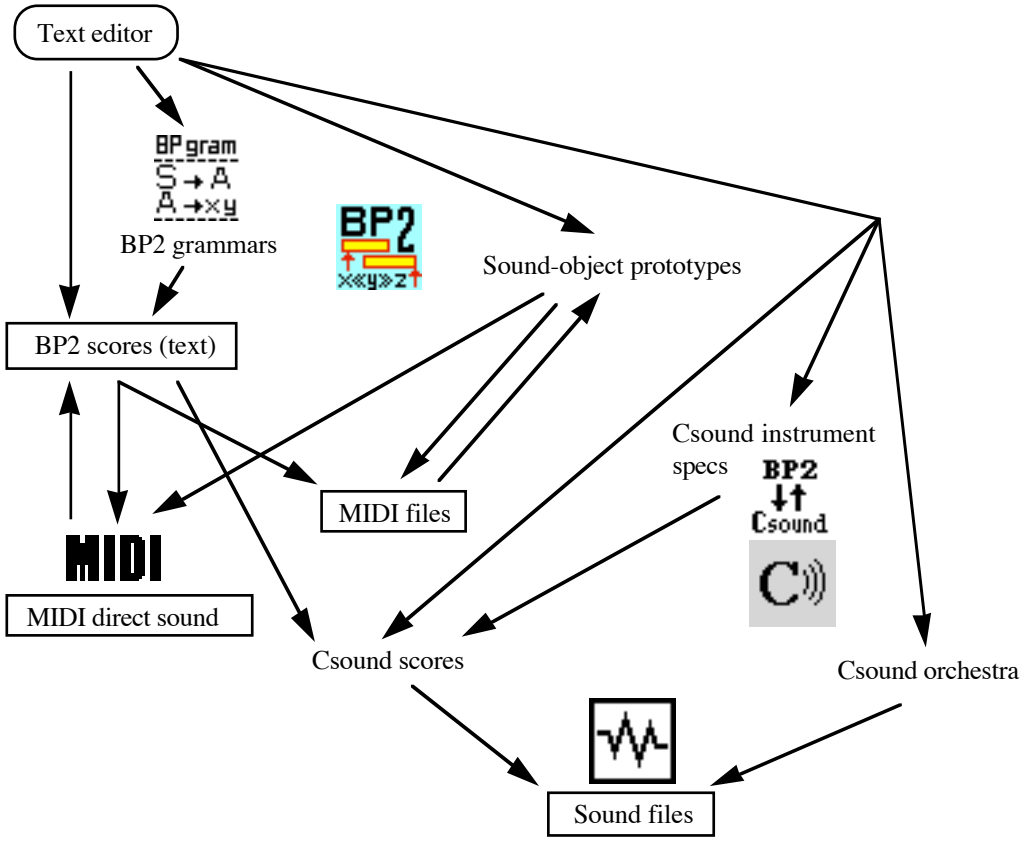


Figure 7: The interaction between Bol Processor BP2, MIDI and Csound. BP2 scores in text format may be typed in, produced by a grammar or entered from a MIDI device. These scores contain simple notes, sound-objects and performance controls. A BP2 score combined with sound-object prototypes works out a direct MIDI output. The same output can be saved to MIDI files which may in turn be exported to sound-object prototypes. The same score combined with a formal description of Csound instruments yields a Csound score. Csound is then invoked to make a sound file using the orchestra file and the score generated by BP2.

Csound handles sound synthesis and transformation algorithms with arbitrary parameters making it easy to represent features beyond the reach of the Western classical concepts underlying MIDI.

Given the description and location of parameters in a Csound orchestra, BP2 tries its best to produce scores handling all significant parameters. While the parameters defined in MIDI (pitch, volume, pitchbend, channel pressure…) can be mapped to those of Csound instruments, BP2 also manipulates user-defined parameters that have no equivalent in MIDI.

BP2 is able to convert a stream of MIDI messages to a Csound score using the parameter definitions and mappings defined in the Csound orchestra

description. Each Csound instrument may be assigned a MIDI channel so that it handles messages on that channel. When no instrument description is available, BP2 uses a default instrument accepting only pitch, volume and pitchbender information, with arguments pointing to optional function tables.

When both MIDI and Csound are used for the same project, it is practical to create dual sound-objects producing related sounds so that outlines of the composition are checked in real time and Csound is used for final versions.

The orchestra/score dichotomy has often been resented as a limitation of Csound. Bol Processor addressed this problem by allowing a high-level description of the orchestra to steer its Csound score generator.

Vectorising continuous parameters

MIDI control parameters such as pitchbend, volume, panoramic,…, or arbitrary parameters defined for Csound instruments, may be set to vary continuously during the performance. In a conventional MIDI sequencer, these streams of messages would be stored along with other data, an approach which is similar to "pixel" graphics.

BP2 uses a "vector" representation in which only a few points are provided and intermediate values interpolated during the performance (Bel 1996). A melodic shape on the note 'D4' could for instance be notated:

_pitchrange(200) _pitchbend(+200) D4 _ _ _ _ _ _pitchbend(-200) _ _ _ _ _ _ _ _ _ _ _pitchbend(+160) _ _ _ _ _ _ _pitchbend(-200) _ _ _pitchbend(0)

Instruction "_pitchrange(200)" instructs BP2 that the range of the pitchbender on the current patch is [-200, +200] cents.

While playing the item on MIDI, the interpreter generates a stream of pitchbend messages at a programmable sampling rate (typically 30 messages per second). When producing Csound scores, BP2 constructs a function table for the Csound parameter mapped to pitchbend. This function table is inserted before the score line of events invoking it to produce the required shape.

Real-time control

All processes in BP2 may be automated thanks to an interpreted script language akin to AppleScript. Some script instructions determine responses to the MIDI environment. For example, instruction "Wait for Continue" causes the performance to hang until a "Continue" MIDI message is received. BP2 can also be operated and synchronised by messages packed in AppleEvents.

Real-time interaction is similar to the situation of several musicians improvising together while they exchange information about parameters such as changes of tempo or compositional patterns through conventional (audible or inaudible) messages.

It is possible to synchronise processes, perform or repeat an item, modify tempo, adjust rule probabilities in grammars, etc., using a MIDI keyboard and

controllers, a MIDI sequencer, interactive programs like Opcode MAX, or even another BP2.

A commitment to text

BP2 gives preference to text representations of musical data and compositional processes (scores, grammars, scripts, glossaries, etc.). All files may be saved in hypertext format so that no data gets corrupted and no transcoding is required to exchange files with other computer platforms or via the Internet.

BP2 data and processes in text format can be stored in standard data-base systems such as Claris FileMaker Pro. We use this feature for storing song melodies that can be played in QuickTime Music by BP2 running in background as a server. Scripts attached to buttons of the data-base upload the contents of fields along with AppleEvents telling BP2 how to process this data.

Conclusion — back to pencils?

Thanks to interactions with musicians tired of Euro-centric music software, the Bol Processor has become a flexible environment for composition and improvisation in MIDI and Csound, taking advantage of text representations of musical data and processes.

The strength of the formal language approach — text representations of musical data and processes — lies in the possibility of structuring material in a comprehensive manner. BP2 helps the composer to concentrate on declarative knowledge while procedural aspects are taken care of by constraint-satisfaction algorithms. In sum, musicians can concentrate on "virtual" music, high-level design strategies, rather than depend on a library of preset algorithms imitative of existing music — for instance permutations *à la* Stravinsky or *à la* Schönberg.

The last word should be with musicians whose concern for flexible and powerful software tools cannot be dissociated from their penchant to adequate task environments. Recently, Harm Visser wrote to the author:

> *I think that the development of more and more visual stuff curtails the possibility of "thinking in your chair." Sometimes I develop grammars, not at the computer, but sitting with a pencil and paper. With programs [other than BP2] this is not possible: you must sit in front of the computer. The difference lies in the type of attention that each software environment demands on the part of the composer, and indeed reflects on the way s/he thinks about music.*

Interestingly, fifteen years ago the Bol Processor venture had been initiated by a musicologist using his computer on the research site to get rid of pencils!

The current design agenda includes new types of sound-objects (fragments of AIFF and MOD files, QuickTime), an editor for shape tables, interactive graphics, microtonal scale management and communication with other computer music software. Tools for the transformation of sequences, such as permutation, retrogression and intervallic expansion or contraction will soon

become part of BP2 syntax. It is hoped that the development team — currently the author and Srikumar Karaikudi Subramanian — will grow in proportion to the demand.

Acknowledgements

This research is an outcome of projects conducted under the banners of the International Society for Traditional Arts Research (ISTAR, New Delhi), the French *Centre National de la Recherche Scientifique* (CNRS, France) and the *Centre de Sciences Humaines* in New Delhi (CSH, French Ministry of External Affairs).

The author is most indebted to Thierry Montaudon and Harm Visser for their musical input and thoughtful comments during the writing of this paper.

Access to software

~~Bol Processor BP2 is a shareware program running under Macintosh operating system. Version 2.7.4 or greater may be downloaded from Info-Mac mirror sites, notably <ftp://ftp.amug.org/pub/mirrors/info-mac/gst/midi/>.~~

http://bolprocessor.sourceforge.net

References

Bel, B. 1992. "Symbolic and sonic representations of sound-object structures." In M. Balaban, K. Ebcioglu, and O. Laske, eds. *Understanding Music With AI: perspectives on Music Cognition.* Menlo Park: AAAI Press, pp.65-109.

Bel, B. 1996. "A flexible environment for music composition in non-European contexts." *Journées d'Informatique Musicale 1996*, Caen (France). Available with sound examples on <http://www.ircam.fr/jim96>.

Bel, B. 1997. *Bol Processor BP2: QuickStart and Reference Manual.* Available from <ftp://ftp.amug.org/pub/mirrors/info-mac/gst/midi/>.

Bel, B. and J. Kippen 1992. "Bol Processor grammars." In M. Balaban, K. Ebcioglu, and O. Laske, eds. *Understanding Music With AI: perspectives on Music Cognition*. Menlo Park: AAAI Press, pp.366-400.

Boulez, P. 1963. *Penser la musique aujourd'hui.* Paris: Gonthier.

Dannenberg, R.B. 1997a. "Machine Tongues XIX: Nyquist, a Language for Composition and Sound Synthesis." *Computer Music Journal* 21(3):50-60.

Dannenberg, R.B. 1997b. "Abstract Time Warping of Compound Events and Signals." *Computer Music Journal* 21(3):61-70.

Jaffe, D. 1985. "Ensemble timing in computer music." *Computer Music Journal* 9(4):8-48.

Kippen, J. and B. Bel 1989a. "Can the computer help resolve the problem of ethnographic description?" *Anthropological Quarterly* 62(3):131-144.

Kippen, J. and B. Bel 1989b. "The identification and modelling of a percussion "language", and the emergence of musical concepts in a machine-learning experimental set-up." *Computers and the Humanities* 23(3):199-214.

Kippen, J. and B. Bel 1992. "Modelling music with grammars: formal language representation in the Bol Processor." In A. Marsden and A. Pople, eds. *Computer Representations and Models in Music.* London: Academic Press, pp.207-238.

Kippen, J. and B. Bel 1994. "Computers, Composition and the Challenge of "New Music" in Modern India." *Leonardo* 4:79-84.

Laske, O. 1989. "Composition Theory: an enrichment of Music Theory." *Interface* 18(1-2):45-59.

Laske, O. 1992. "A Search of a Theory of Musicality." *Languages of Design* 2:209-28.

Laske, O. 1996. "Creativity — Where should we look for it?" In T. Dartnall, ed., *Creativity, Cognition, and Computation*. Cambridge: The AAAI/MIT Press.

Révész, G.E. 1985. *Introduction to Formal Languages*. New York: McGraw-Hill.

Roads, C. 1984. "An Overview of Music Representations." In M. Baroni and L. Callegari, eds. *Musical Grammars and Computer Analysis*. Firenze, Italy: Olschki, pp.7-37.

Vercoe, B. 1993. *Csound. A Manual for the Audio Processing System and Supporting Programs with Tutorials*. Massachusetts Institute of Technology. Internet distribution: <ftp://notam.uio.no/pub/mac/audio/>.

Xenakis, I. 1963. *Musiques formelles*. Paris, France: La Revue Musicale.