TEX and music: An update on TEX*muse*

Federico Garcia

1 Introduction

For some years I've been working on a professional music typesetting system using TEX and METAFONT, called TEX*muse*. This article has some background and examples of the main idea of the TEX*muse* system which, as will be seen, is entirely inspired by the TEX 'spirit'. This touches on the potentially disastrous problems of WYSIWYM for music typesetting, and on what similarly oriented systems have done (and not). Some achievements of TEX*muse* could potentially change the life of any composer.

2 A sample of the current prototype

2.1 The user's input and the first scan

In the main input file of a TEX*muse* document the user defines a series of instruments (more accurately 'staves'), and then requests TEX*muse* to assemble a score out of a subset of them (possibly all of them):

```
\newstaff[\trebleclef\AMajor]
    {righthnd}{MztAMj.txm}
\newstaff[\bassclef\AMajor]
    {lefthnd}{MztAMj.txm}

\score: {righthnd,lefthnd}.
```

This convention makes it possible for the same document to produce both a general score (which includes all the instruments) and the individual instrumental parts; it also allows changing the order of the staves, replacing one by another, reusing the same music in several different documents, etc. — while keeping the actual musical content independent.

The `\score` command is (planned to be) flexible enough to allow several kinds of groupings between the staves. Piano music is typically typeset as two staves joined by a brace at the left of each musical line, and that is what the braces in `{righthnd,lefthnd}` mean. But a construction like `[oboe,clarinet]` would instead join the staves by a bracket, and the simple `flute, violin` would simply include the staves in the score without any particular grouping or joining shape.

The `\newstaff` command defines the staves, and instructs TEX*muse* where to look for the actual musical content of each staff. The actual musical content is typically collected in `txm` files, in the form of a series of staff environments, each named as the staves themselves. The file may contain staff environments even if they are not used in the document, in which case the program would simply ignore them.

In this case — an excerpt from Mozart's Sonata in A — the music for both staves is collected in the auxiliary file `MztAMj.txm`. The following is the environment for the staff '`righthnd`' as it appears in that file (the output appears in section 2.5):

```
\begin{righthnd}    \rangefrom{g4}\meter68
4c.3d4c 5e4e  4b.3c4b 5d4d  5a4a 5b4b
5c\split{3ed5c4b\\4b 5a4g}  4c.3d4c 5e4e
4b.3c4b 5d4d
5a4b 5c\split{4<bd>5<ac>4<bg>5<ae,>\\
      \,4f5e4d5c}
     4r
\end{righthnd}
```

This kind of input is not too different from the model used in commercial music software: numbers tell the program what rhythmic value to expect for the coming note(s): quarter notes are indicated by '`5`', eighth notes (smaller by one than the quarter note) by '`4`', sixteenths by '`3`', and so on.

Rests are issued with an '`r`', the characters `'` and `,` are used to shift between octaves, and `.` adds a dot to the previous note. Multi-note entries (i.e., individual notes that have several noteheads on the same stem — pitches sounding at the same time) are indicated by `<` and `>`, although there are *many* utilities and shorthands for different multi-note situations.[1]

The `\split` operation tells TEX*muse* that the music in the staff is to be split into two voices, one of which will be typeset above the other (but on the same staff). The user does not need to indicate which note on the upper voice corresponds to (i.e., goes on the same vertical axis as) which in the lower: TEX*muse* scans both voices and pairs them up as expected, according to their rhythmic durations.

In fact, this ability of TEX*muse* to align voices by rhythm is true not only of `\split`, but is indeed the mechanism that allows the user to input staff by staff. Apart from the convenience of keeping each staff independent of the others, horizontal input (i.e., staff by staff) is the natural way the musician thinks of the music. The opposite — requiring the user to figure out in advance the rhythmic/vertical correspondences — is as unnatural as it would be to typeset a paragraph word by word, starting with the first word on the top line, continuing with the word that would be below it on the

[1] For example, after `\dyads` it won't be necessary to wrap every set of two notes with `<` and `>`; `\coll` will make TEX add a note at the specified interval (like in `\coll8b`, which adds the lower octave); and so on.

second line, then the first word on the third line, and so on. It is this unfriendly requirement that makes MusiXTEX (the remarkably complete system developed in the 1990s under the leadership of the late Daniel Taupin) hopelessly useless for the typesetting of music beyond small snippets; it has in fact prompted the implementation over the years of several 'pre-processors', non-TEX utilities that translate horizontal input into the vertical arrays required by MusiXTEX.

TEX*muse* achieves this because its code, so to speak, teaches music to TEX, so that it is able to interpret the music of each staff and deduce their mutual rhythmic relationship (and therefore their vertical alignment). It will be seen in the coming sections where exactly this deduction happens. But this discussion introduces a more general and relevant issue concerning the use of TEX for typesetting music: beyond the question of pure ability to produce the score itself, what concessions would any such program require from the user? How far can we go in preventing the author/composer from having to stop thinking of music to help TEX with typographical decisions? This is an important question, because by its very nature TEX can very well fail at this; but it will be shown that it is here too that it can succeed and surpass the alternatives. This discussion is the main topic of section 3 below.

2.2 From `txm` to `tm`

After scanning the user's input in the `txm` file(s), TEX*muse*'s first step is to translate it into internal functions that govern the appearance of each note.

In the previous section we saw that the user typed '`4c.`' for the very first note of the `righthnd` staff. After TEX*muse*'s internal translation, this first note looks (to TEX, privately) much more involved:

```
\@nntr\@nhd1{c5}\@invbl2{c5}\@agdt\@stem
    \add@bm1\end@{432}\relax
```

To paraphrase:

`\@nntr`	For this coming *new entry*,
`\@nhd1`	a *notehead* of kind 1 (the familiar filled-in ellipse)
`{c5}`	on the fifth c on the keyboard,
`\@invbl2`	with a kind-2 *invisible accidental* (a ♯)
`{c5}`	on that same c,
`\@agdt`	with an *augmentation dot*,
`\@stem`	a *stem*,
`\add@bm1`	added to a *beam* of index 1,
`\end@`	and *ending* at position 432.

The idea of 'a dotted eighth note on c5' (which is what the user requested with '`4c.`') has been made fully explicit into discrete commands that order the actual notehead, the stem, etc. But not only that, in the process TEX has done a couple of other things as well:

- After `\end@432`, TEX knows that the current staff won't have a note until position 432 is reached by all other staves. If, say, the second staff has a note that ends at 288, then that staff will have its second note starting at 288, but the first staff (which already spans up to 432) will have nothing on that note. The process is a 'quantization' of the rhythmic space, in which the 256^{th}-note (a note with six flags) contains 9 units. The first note in our sample is a dotted eighth-note: 288 (32 times a 256^{th}-note) +144 (a dotted note is increased by half) = 432.
- TEX*muse* also deduced that this note should be part of a beam.[2] It has looked ahead for the coming notes and has realized that since those notes are also beamable rhythmic values and they are part of the same beat in the current meter, they should be beamed together.
- The note also has an invisible sharp sign. By 'invisible' TEX*muse* means that it doesn't have to be actually drawn, and the question might arise then why to include it at all. This is part of TEX*muse*'s 'spelling' mechanism, a component of its ability to interpret the musician's intentions and to render it correctly onto the typeset score. This mechanism and its importance will be addressed in more detail later (section 3.3).

  For this note, TEX*muse* has deduced that the user meant c♯ (since we're in A major), but it also realizes that there is no need to explicitly give the sharp in the actual score. In addition to 'invisible', accidentals can be 'rigid' (when they must technically be added to mean the correct note) and 'courtesy' (sometimes also called 'cautionary', when the context makes it desirable to add the clarification).

In this way, notes in the user's input have been converted into sequences of internal commands like the line listed above. These 'entries' are written, staff by staff, on auxiliary files named ⟨*staffname*⟩`.tm` (in this case, the entry listed above is the first entry in `righthnd.tm`). Those files will be read by TEX*muse* in the next step of the run, to gather the notes and write METAFONT programs to draw them.

[2] The *beam* is the familiar thick line that joins the stems of several notes when they are small rhythmic values (an eighth-note or smaller).

2.3 From TEX to METAFONT

With the information in the `tm` files, TEX can now proceed to write programs in METAFONT that will draw the notes into a font. The following is the program that TEX writes for the first note in our sample:

```
newchar(1);
    staff_1;
        notehead(1,C5);
        openbeam(1);
        augm_dot;
        stem;
    staff_2;
        upper;
        notehead(1,E4);
        stem;
        lower;
        notehead(1,H3);
        openbeam(1);
        augm_dot;
        stem;
    endcharat(432-0);
```

The first thing to note here is that this listing contains information about *both* staves. In fact, it is the `tm`→`mf` stage that gathers the notes, from all the staves, to make the composite 'characters' that include any note that belongs in each vertical axis across the score.

In this first note, the right hand (`staff_1` for METAFONT) is simple enough; but the left hand contains two noteheads, which are in addition separated as '`upper`' and '`lower`' sub-notes. This comes from a `\split` in the original user's input for the left hand (not shown above), and illustrates what a split ends up looking like in the METAFONT program.

The `\add@bm` ('add to a beam') command has generated an `openbeam` in the METAFONT program. Since this is the first note, there are no beams currently open, so `\add@bm` is interpreted as opening one. Future notes with `\add@bm` will be given an `addtobeam` instead, until a `\close@bm` is found and the beam closed.[3] On the other hand, the '432' is retained and it still signals the end of the entry in `endcharat(432-0)` — METAFONT will do the math and find 432. TEX had used the ending positions of the notes to figure out the relationships between staves; but now the same information (which ultimately encodes the rhythmic profile of the piece) will be used by METAFONT for the spacing of the music. (In music, the longer the rhythmic value the more space there is after a note; the space after each note is later stretched proportionally, if necessary, to reach the right margin.)

TEX*muse* now enters the drawing stage, reading the character programs with the music-drawing functions defined in its code. The user (or a batch file) runs `mf` on these METAFONT programs generated by TEX, and out of this a new font is created (or many, if there are more than 256 characters in the piece). TEX will then use the font to typeset the music — which, for TEX, is just regular text.

2.4 From METAFONT to TEX

But the new font does not contain every single symbol in the music. METAFONT does not really draw some pre-formed symbols that are part of a musical score; for example, any lettering or any numbers will be inserted by TEX (retaining user control on their font and appearance), as well as some musical symbols like the clefs that don't really change from instance to instance and would be inefficient to draw on the fly. TEX will insert those symbols at the appropriate positions in the score, but for that it needs METAFONT to tell it where those positions are.

This information is passed to TEX in the log file of the METAFONT run. This is an excerpt from it:

```
Line at measure 3
Next break after measure 6
\fi\leavevmode\iffalse
\fi\rlap{\hbox{\raise 7.0pt\hbox
    {\mae ?}\kern2.2pt
\raise-3.5pt\hbox
{\mse \char 146}}}\iffalse
\fi\rlap{\raise38.5pt\hbox{\mae \&\kern2.2pt
{\mse \char 146}}}\iffalse
[1]
\fi{\tmfont\char1}\iffalse 3.9 [2]
\fi{\tmfont\char2}\iffalse 1.5 [3]
```

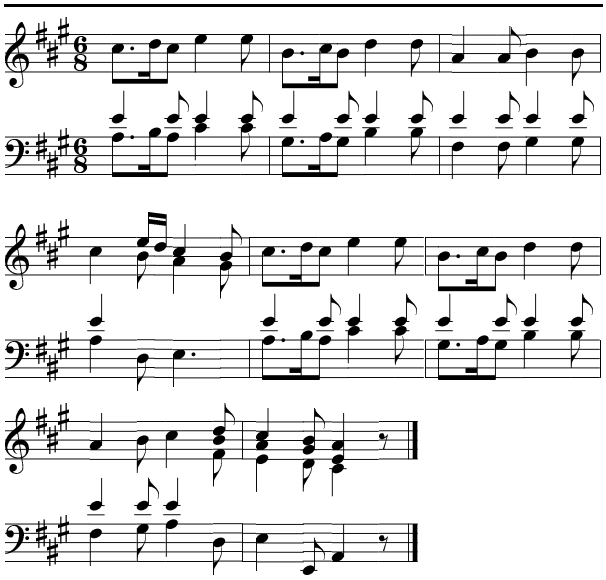
This is pretty low-level plain TEX, but you can still spot the `\char` commands that request particular symbols. `\mse` is TEX*muse*'s base musical symbols font ('muse'), in which character 146 is the key signature of A major. The clefs, at the time of writing, come from the TrueType font 'Maestro' (the default font for Finale, a mainstream commercial music program): `?` in that font is the bass clef, and `&` is the treble clef.[4]

[3] The reason for this two-step conversion is that the rests, that can also be included under a beam, behave differently. In other words, `\add@bm` is not *exactly* equivalent to `openbeam` or `addtobeam`, and `\close@bm` is not directly `closebeam`; in this particular case the difference is innocuous because no rests are involved.

[4] Eventually, Maestro will be entirely replaced by TEX*muse*'s own fonts. It has so far been used to permit implementation of the prototype even in the absence of a complete native font, and also to test TEX*muse*'s ability to use and interact with user-selected fonts.

`\tmfont` is the command that stands for the actual font that METAFONT has just drawn, and so `{\tmfont\char1}` is the exact point where METAFONT tells TEX to print the very first note of the piece. The log file continues requesting what goes in the score — the characters that it just drew or insertions from other fonts — and simply by reading the file TEX gets to typeset the music.

2.5 Sample output



TEX*muse* is still very incomplete. Apart from the fact that it doesn't yet connect the barlines across the two staves of the piano, much less join the staves with a brace on the left sides, the '$\frac{6}{8}$' (which as of now can't be scaled down...) appears slightly misplaced to the left, and the '$\sharp\sharp$' is misdrawn (it should give c♯, not b♯!).

But the skeleton is in place, and many of the missing features are more or less simply analogous to features already working in the current prototype. TEX can indeed interpret intuitive user input, and it can program METAFONT virtually without help from the user. The system is promising in terms of flexibility and programmability.

3 TEX and music: Why and why not

There are legitimate reasons, however, to be skeptical about music typesetting in TEX — or in fact in any system like TEX, with plain-text input, a compile phase, and fixed output.

3.1 Music, tables, and TEX

As mentioned, TEX*muse* (like the pre-processors for MusiXTEX) goes to a lot of trouble to make out the vertical correspondences implied by the user's horizontal output. The analogy above (typing a paragraph in vertical fashion) can be complemented by the realization that music, from the typographical point of view, is more like a table than it is like running text: vertical alignment by rhythm is as important, detailed, and meaningful as the horizontal dimension where a melody is presented.

In a way what the system does is *hide* the table from the user, in order to spare him the need to think of it explicitly. But this doesn't change the fact that the music is still a table... and tables are one context where the shortcomings of a TEX-like system are acute.[5] As any LATEX user knows, maintaining a table (editing it, updating it, correcting it) is not the most straight-forward intuitive process for us. In Excel, or even in a table in Word, you can simply click on a cell and start typing or deleting; in LATEX you have to count `\hline`'s and `&`'s even to find the cell. Searching is impeded, selecting a portion of the table for copying or pasting is impossible, ... TEX is simply not a natural environment for tables the way Excel is.[6]

There is therefore a big built-in inconvenience in the use of TEX, or any such system, for music typesetting. For me, a conclusion has emerged, as work on TEX*muse* has progressed, that the advantages that do come naturally with TEX — programmability, for example, or *precisely* targeted control where user decisions are documented, visible, and accessible as part of the input — are, by themselves, not nearly enough to compensate for that inconvenience. In other words, that TEX*muse*, if it was only a matter of those advantages, would unfortunately *not* be the best choice for a musician's music typesetting needs. Much more than that is needed for a system based on plain-text input to compete with WYSIWYGs, no matter how deficient the latter are... (and they are).

3.2 WYSIWYM in music?

The reference in the previous paragraph is to the needs *of a musician*, as distinct from the needs or concerns of an engraver proper. A musician does not need to know, for example, the rules for positioning beams, any more than a mathematician needs

[5] As Frank Mittelbach pointed out in another talk at this conference, (LA)TEX has severe limitations in this area (for example, there's no way to make a cell span several cells *both* horizontally and vertically at the same time). The problem referred to here is even more immediate and pressing.

[6] Of course the problem is not only TEX's: the same is true of HTML, for example, or any system that takes what is inevitably one-dimensional input to manipulate what is inevitably two-dimensional output.

to know about the rules governing the positioning of superscripts. The real test for TEX*muse*, or any system that aspires to being a good choice for musicians (composers, analysts, etc.), is whether or not it assists the *musical* mind without imposing concerns that are more typographical, or, even worse, syntactical, than musical.

For example: when we are in, say, c minor, every e is flat by default — that's part of what 'being in c minor' means. So, in c minor, the musician will write a scale simply as c-d-e-f-g, even though the e is actually e♭. Requiring the musician to explicitly request e♭ is forcing the mind to step outside a purely musical train of thought: think of a mathematician having to request explicitly the italics for the letters in the equations. From the typographical point of view, this is correct: the copyist or engraver, in music or in math, does indeed go through the extra step. But from the 'semantic' point of view, so to speak, of someone who is writing down a thought, this gets in the way. In a sense, those kinds of extra steps, foreign to the train of thought, are the whole point of using a computer — rule-based, mechanical, in principle automatic: isn't that what the computer exists for?

It is a priority that the input the user is required to learn and apply when typing music be as intuitive as possible: as close as possible to the actual steps and motions of writing the music by hand. In the c minor example, e♭ should be simply typed as e (just as you would write it on paper), the flat being silently deduced by the engine. It's not a matter of '*what you see is what you mean*', because we do mean e♭ although we don't write it out (or see it) in full. Rather than WYSIWYM, the paradigm is more one of '*what you see is what you'd write*'.

Or even more to the point: it's a '*dear computer, you know what I mean!*' kind of thing. Say there's a passage where every entry will consist of two notes — well, the user should be able to say `\dyads` and not worry about grouping every pair of notes with whatever syntax; if an entry will be repeated the same way (which happens quite often in music), shouldn't we have a shortcut, like using `%`,[7] so that we can type '`<ceg>%%%`' and obtain the same result as if typing '`<ceg><ceg><ceg><ceg>`'? Not that we want the %-like symbol to be output — we want to do less typing, and our meaning should be clear to the computer.

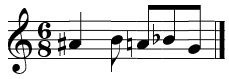
TEX*muse*'s command `\coll`, similarly, permits constructions like `\coll8b{cdef}`, meaning to add a lower octave ('*8va bassa*') to each note — a trick composers have used in their manuscripts for the copyists to add the extra note.[8]

And so on. These are very simple tools to program, and they follow naturally from admitting that horizontal input is, well, a necessary evil, that needs to be alleviated as much as possible.

The point, however, is that even with a successful syntax — one that helps rather than infuriates the user — even then it is legitimate to fear that TEX*muse*, or any system like it, is not a real alternative for use by musicians... the successful syntax just means that entering the music won't be more annoying than entering it in a WYSIWYG; but there's still a long way to go in order to overcome the structural inconvenience pointed out above, of not having direct, mouse-click access to each note in the score.

3.3 Tipping the balance

Consider the following snippet:



The three 'accidentals' in this example (the sharp ♯, the natural ♮, and the flat ♭) are all necessary parts of the pitch specifications; in particular, the second one cancels the effect of the first one (since in principle accidentals carry through full measures and therefore the original sharp on the a would apply to the second a as well).

Next, let's observe that the note with the ♯ and the one with the ♭ are in fact the same pitch (i.e., the same key on the piano). The musical spelling in the example (the particular choice of a♯ for the first one and b♭ for the second) is the correct one musically: since sharps are generally 'felt' to point up while flats tend to point down, the chosen spelling makes the notes actually reflect graphically the contour of the melody — a fact that, apart from looking better, is important semantically as well. Context matters, however: if the note following the b♭ was another b (natural), then the b♭ should be spelled as a♯ (a better way of leading to b); but if the last note was c, then b♭ would again be the preferred spelling (even though that flat would lead upwards in that particular case).

Most music typesetting programs today allow for 'MIDI input', meaning that they can take pitches in as keystrokes on a musical keyboard plugged into

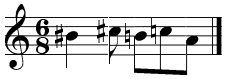
[7] A sign resembling the % is widely used informally for musical repetitions.

[8] Italian *coll* is an archaic contraction of *con il*, which was used since early times in instructions like "*coll violino I*" ("with the first violin"). The engraver would then copy the same music that was in the first violin. "*Coll*" was later generalized to other shorthand uses.

the computer. But they don't have a dynamic interpreter that would spell the notes correctly. You can (with more or less work) set each particular key on the piano to be spelled one way or another, but that isn't good enough: in our example, the two occurrences of a♯/b♭ would be spelled *both* as either b♭ or as a♯, and then you'd have to go back and correct them.

This is actually only a minor inconvenience, in part because it's only one small side of a much bigger annoyance. The 'dynamic pitch interpreter' that is lacking here would not only be able to make musical sense of pitch keystrokes, but, much more importantly, it would free the user from making spelling decisions for each context. It's not simply that the second-to-last note in the example should be spelled differently according to what note follows — the passage can change context in innumerable ways that would affect spelling:

- Say a different instrument will join the melody only starting in the middle: then it doesn't need the ♮, since there's no ♯ to cancel. A copy-paste operation in many a program today would incorrectly include the ♮.
- Say the composer decided to split the one measure into two (changing the time signature): then the ♮ is not technically needed, since a new measure cancels any previous accidentals. But in most cases it is a good idea, for clarity's sake, to put in the accidental anyway (the so-called 'courtesy' accidental). Many programs today would simply remove the accidental altogether, following the technical rule, and once the composer requests that the accidental be shown anyway, that decision will apply in any further use of the music, regardless of whether the new context requires it or not; and ten the courtesy accidental will create more, not less confusion...
- Say this line is for clarinet, and since the clarinet is a transposing instrument, you'll have to transpose it up a whole-tone for the clarinetist to read. In programs today, you'll get:



which is completely unacceptable: b♯–c♯ should have been spelled c–d♭.[9]

Because of all these potential problems, experienced music typesetters know that the process of setting a score includes a special step dedicated solely to proof-read the spelling. Even so, this is the area of music typesetting that is *by far* responsible for most typographical errors found in scores, new or old.[10]

This is a *huge* issue indeed — the kind of issue that, if solved, would make a program so useful above and beyond WYSIWYGs that it may well start to compensate for the inconveniences of plain-text input. TEX*muse* has a prototype of a spelling mechanism that does just this. (And has prompted several colleagues of mine to ask me regularly, but especially after struggling to meet deadlines, how close I am to completing the program so that they can use it!)

Less ground-breaking but similarly welcome is TEX*muse*'s system of beaming automation that also takes context in mind. This is not as significant in terms of a score's correctness, but it does separate amateur from professional musical engraving.[11]

And another important advantage of TEX*muse*: by deciding on line-breaking and page-layout — this latter not fully implemented yet — once again according to context and without user intervention, TEX*muse* will simplify the handling of the multiple looks a score takes. Any piece for more than one instrument consists not only of the general score (with every instrument in it), but also of the individual parts (the flutist gets the flute music only, the oboist the oboe music only, and so on). If, as in commercial software today, user intervention is needed to lay the pages out, then that means that the user will be helping out not with one document, but with a multitude of them. Spelling creates the most typos in scores, but it is this process, 'part extraction', that consumes most of a composer's time these days.

You could say that WYSIWYGs could just add these functionalities to their engines, and that is true to a certain extent. But the fact is that these utilities are actually more at home in an input-compile-output model than in a GUI. They all depend heavily on

[9] The problem arises from the fact that the correct b♭ transposition is not necessarily always a 'major second,' but may actually have to be a 'diminished third' in some portions; the two things are equivalent in terms of pitch — key on the piano — but not in terms of note on the staff. The program's downfall is again the lack of a dynamic, context-sensitive pitch-to-note interpreter.

[10] I saw an exercise in a recent conducting class where students were given one page from a score by Debussy where they had to find five typos (five!). *All* the errors (in a score that is close to 100 years old) were spelling mistakes.

[11] The paradigmatic case is a 3/4 time signature, where beams span across the three beats if there are only eighth-notes, but should break into beats in the presence of sixteenth-notes or smaller rhythms.

context, and on the context of the whole piece at that. With a GUI, automation would mean that editing (by the user) would create on-the-fly changes (by the program) that could easily get the user lost. A spelling mechanism would make notes dance from spelling to spelling after each new note was input; every copy-paste would necessitate the computer going over the full score; every new bar would potentially make you jump to a new line or even a new page (have you seen Word trying to deal with orphans and widows?). TEX, on the other hand, is a parsing, transformation language that operates on an input stream: exactly the right environment to implement the spelling of a melody and the beaming of a rhythm.

To sum up: by (a) keeping the inconveniences of plain-text input as minor as possible, and (b) implementing automation and interpretation mechanisms far beyond present applications, TEX*muse* is indeed promising as a real alternative for high-quality, sophisticated, and musical music typesetting.

4 Future directions

The presentation of the current state of TEX*muse* and of music typesetting in general at the 2012 TUG meeting generated excellent comments. Two major areas of feedback are particularly significant, and I would like to mention them as future lines of work.

4.1 TEX*muse* and LilyPond

An existing program has been conspicuously absent from the foregoing — the music typesetting system LilyPond. Also plain-text-compile-fixed-output, it stemmed from work on a preprocessor for MusiXTEX, MPP, whose author (Jan Nieuwenhuizen) was concerned not only about providing horizontal input capabilities, but also about the relatively poor quality of MusiXTEX's output. In the latter part of the 1990s, Jan and Han-Wen Nienhuys abandoned MPP and started work on LilyPond — now a non-TEX application, but based on TEX (and LATEX) for inspiration and approach.

LilyPond today is a complete working system, well known, with excellent output and an active community of both users and developers. With a much more reasonable input system than MusiXTEX's, it still fails in meeting the need for input that is intuitive to a musician—section 3.2 above makes in fact constant if veiled reference to LilyPond's input model, attempting an explanation to the observed fact that most musicians who try it find it unsatisfactory. (Mentioning LilyPond directly would have been unfair, since on the one hand the points are of general scope, and on the other these 'complaints' are not all there is to LilyPond or all I think of it.[12])

The fact is that LilyPond has essentially solved the problem of graphically realizing a musical score that has been encoded in some kind of plain-text syntax. If, on the other hand, TEX*muse* is able to interpret intuitive input — with a syntax tailored to composers and allegedly amenable to them — and from it write METAFONT programs to draw the music, could it not write LilyPond programs instead? That way we don't have to build METAFONT's musical engine from the ground up.

This is an excellent point and working on this is the most reasonable path to getting TEX*muse* to any degree of completeness in a reasonably short time. Even if the METAFONT macros will eventually be fully developed (so we keep the whole process within the TEX installation), a 'lilytex' package where TEX*muse* uses LilyPond (instead of METAFONT) is certainly high on the agenda of TEX*muse*'s development.

4.2 LATEX3

Some of the modules of TEX*muse*, including the signature automation capabilities, are general-purpose computer programming, and in developing them in TEX some of the well-known 'features' of TEX have made their presence felt. `\expandafter` must be among the top five command names in occurrence in TEX*muse*'s code. With the 'coming of age' of LATEX3 and the update about it at TUG 2012 (by Frank Mittelbach and Will Robertson), a new line for future work on TEX*muse* suggests itself: the conversion to LATEX3's programming environment expl3. Individual, self-contained modules (the spelling mechanism, for example) will be translated into the new language, as a way to test the algorithms, have a taste of expl3, and make TEX*muse* an active part of the most current developments in the TEX world.

◇ Federico Garcia
Artistic Director
Alia Musica Pittsburgh
federook (at) gmail dot com
http://www.fedegarcia.net

[12] In particular, LilyPond has full beaming automation (see note 11).