Plaintext “Ecosystem” for Musicological Work
Proposed Additions to LilyPond and LaTeX

Urs Liska

December 17, 2013

Introduction

Recently I have published a number of articles which more or less explicitly propose the use of plain text driven toolchains for serious scholarly work in the musical disciplines. In particular I'm talking of the essay "Plain Text Files in Music", published at the *Scores of Beauty* blog[1]. To some extent the present paper assumes some familiarity with the contents of that essay or its German version (which is rather compressed but explicitly focused on scholarly work).

Plain text tools like the LilyPond engraving software and the LaTeX typesetting system can be combined to very integrated, collaborative workflows using version control software such as Git. While existing possibilities are already very impressive I believe that from this perspective there *are* some ideas for possible improvements. This should in no way be mistaken for a list of shortcomings, quite the contrary. The fact that text based toolchains *do* allow for such considerations is awesome in itself, and I believe that through a number of developments one could head towards a near-perfect ecosystem with tools for completely integrated scholarly workflows.

In this document I will outline a number of projects that I consider interesting or important in this respect. Some of them are already in the planning or implementation phase, some are mere ideas that could be realized at any point if sufficient interest in them arises. Some of the ideas are small-scale projects that can be implemented "along the way", but there are also projects that would require a considerable amount of time, developer's power – and money. These would be best tackled in the context of larger (research oriented?) projects.

[1] `http://lilypondblog.org/2013/07/plain-text-files-in-music/`

Contents

1 Additions to LilyPond

1.1 lilypond-doc

`lilypond-doc` will be a tool to document LilyPond source files. Its goals are two-fold, while only one is concretely relevant for the current overview. Documenting an API[1] is useful for navigating in documents or for documenting libraries, but documenting *content* can greatly increase musicological productivity. `lilypond-doc` has several aspects to its "content" part.

Design an Interface The first important step is to design an interface to *insert annotations* of all kinds in the LilyPond input file. Apart from documentation these annotations also serve as a communication channel between the participating editors. Annotation types are for example technical remarks ("How can I realize this notation?"), musicological discussion ("There is a pencil marking in the manuscript. Should this go into the score or only into the report?"), TODO items ("Review manuscript B", "Check the beaming of this passage") or finally real entries for the critical report.

There has already been significant progress with this stage, and this will evolve "by itself".

Process Annotations The second step is to process annotations, which will be done during the compilation of a score. Annotations are printed to the console output during compilation so other editors of the score will notice them. Optionally they will be visible in the document (when in draft mode, see below), either by coloring objects or by adding annotation markers (without affecting the layout). Additionally the annotations will be output to a set of intermediate files, intended to be a documentation of the score. This output will be sorted by the position of the annotation in the composition and be equipped with hyperlinks pointing back to the position in the input file. So there will be (separate) files with technical questions, musicological discussion, TODO items and critical report entries (and possibly more). When reading these files the reader can click on an entry and will be brought to the relevant line in the input file – so he immediately has the context for the issue at hand.

Development of this hasn't started yet, but it is clear that it will be done with Scheme, LilyPond's built-in extension language. It is intended to add this functionality to LilyPond itself once it's reasonably stable.

Use annotations The intermediate files produced during compilation can be processed by an external program and converted to a number of output formats such as PDF, HTML or XML files. One particular important use of the results will be the automatic inclusion of the list of critical remarks in a LaTeX document with the help of the `muscritreport` package (see 2.4 on page 7).

[1] Application Programming Interface

The implementation of such a secondary tool should reuse existing technology as extensively as possible (e. g. the `pandoc` format converter[2]).

To sum up: `lilypond-doc` will make it possible to produce a critical edition completely *within the actual score document!* (To judge the importance of this statement please consider the role of version control in collaborative workflows.) The effect of this will be even increased with development of a Frescobaldi interface to edit annotations directly from its music view (see 4.1 on page 10)[3].

1.2 Editorial toolbox

The editorial toolbox is a set of LilyPond commands to mark items in a score as "editor's amendments", which will affect their appearance in a predefined way (e. g. parenthesize them, print them dashed or make them smaller). These commands will interface closely with those from `lilypond-doc` (or even become an integral part of this project). There are several advantages to having editorial annotations as such commands. Similar to character styles in a DTP application they ensure consistent appearance throughout the score – and these styles can be part of a "house style" to guarantee consistency throughout multiple editions. Interfacing with `lilypond-doc` they can be used as a communication channel between editors and therefore as a quality control mechanism. And in combination with draft mode settings (see 3.2 on page 9) they are highlighted during the edition process.

A toolbox of this kind is already in use and has proven useful in a private edition project, but the development of a package will make the implementation of this concept generic and configurable.

1.3 Functional Analysis

One engraving task that can't be done in a satisfactory manner yet is the typesetting of functional analysis symbols. While this isn't of great relevance for editions it is quite important for engraving examples for musicological books or preparing teaching/exam materials.

Depending on the desired quality level, completeness and versatility this may range from a small-scale to a quite ambitious project. For example it may make a huge difference if one typesets symbols similar to articulations or dynamics, or if one wants them to share a single baseline like lyrics do. The latter becomes particularly complex when one wants to have several layers, for example in the case of modulations.

There haven't been made any real attempts in this direction so far, but the project should be started together with a corresponding LaTeX package (see 2.2). `riemann` should also be able to handle scale-step analysis symbols.

[2] `http://johnmacfarlane.net/pandoc/`

[3] See also `https://github.com/openlilylib/lilypond-doc/wiki/Documenting-musical-content` for more details and the current state of considerations.

2 Additions to LaTeX

2.1 lilyglyphs

`lilyglyphs`[1] is a package that prints *any* notational element from LilyPond in the continuous text of LaTeX documents. It is particularly useful for typesetting critical reports, analytical essays or educational material.

`lilyglyphs` has already been released and is available in some LaTeX distributions, while future development plans concentrate on making the package more stable and increasing the coverage with predefined commands (you can always use ad-hoc symbols by accessing the glyphs from LilyPond's music notation font directly but predefined commands are more convenient).

2.2 riemann

`riemann` will be a package to typeset functional and scale-step analysis symbols. It won't try to cover all aspects within a newly designed font (such as the *FinalAnalyse* TrueType font that would do that for the Finale notation program), but rather use any font the user selects and place the components of analysis symbols at their respective place. This will be similar to or actually using LaTeX's famous mathematics mode.

2.3 musicexamples

`musicexamples`[2] is a package that manages music examples within LaTeX documents. In its current state it can already print music examples in a variety of ways while managing captions, references and a list of musicexamples. Examples can be fixed or "floating" (i. e. floating through the text to find the optimal position). Specific care has been taken to handle full- and multi-page examples, with explicit support for scores that should explicitly start on odd or even pages.

One development plan is to watch over the used music examples (that is, over those created with LilyPond) and recompile them if they should have been modified. Another item on the TODO list is to develop specific templates and compilation modes to streamline the creation of consistent music examples for a given publication (series).

One major project will be the ability to handle LilyPond source code directly in the LaTeX document. LaTeX's will then use LilyPond to compile the score fragments whenever necessary and replace the source code with the resulting image files. This will be particularly useful for documents with large numbers of music examples, especially short ones.

[1] `http://www.openlilylib.org/lilyglyphs`

[2] `http://www.openlilylib.org/musicexamples`

2.4 muscritreport

`muscritreport` will be a LaTeX package to typeset critical reports. It will benefit from the fact that – unlike with word processors or DTP programs – LaTeX commands can have arguments. This can be used to treat entries for a critical report as "records" of a "dataset". It will be structured similar to LaTeX's existing BibTeX bibliography tools.

The data for the critical report can thus be stored in a structured format that is completely independent from its appearance in the printed edition. In fact it's independent from its use altogether: using generic storage formats enables us to use the entries not only for the printed report but also to populate a database or to serve requests from a web frontend. At the same time the visual appearance for a printed edition or a "house style" is completely configurable.

One important aspect that will make `muscritreport` a unique tool is that the storage files for the critical report entries can be generated from annotations within a LilyPond score's input file (through `lilypond-doc`, see 1.1). This way it will be possible to completely prepare a scholarly edition within one common code base!

3 Core LilyPond Improvements

While LilyPond is a very good piece of software it is not perfect – but no other software is perfect either. The LilyPond developers are quite eager to improve the program, fix bugs and add improvements. But there are a few specific enhancement requests which would significantly improve the program to become an even more useful tool for scholarly musicological work. Unfortunately as these are somewhat specific they might not be "automatically" addressed because there are always more urgent and less specific issues to be handled.

3.1 Export to MusicXML

Actually this section deals with the only issue I'd actually call a real deficiency with LilyPond: It can't export scores to other than graphical formats (PDF, PNG, SVG, EPS). Usually this isn't a problem because LilyPond's output is of such a high quality that there shouldn't be any need for post-processing. The only use case sometimes heard of is editing the final score with real graphics programs – which can open the PDF files.

But in fact the current state isn't good social behaviour – "vendor lock-in" is one of the most-dreaded issues in the Free Software world. It becomes a problem when most publishing houses, the major ones in particular, insist on their well-tested workflows and only accept files in Finale or Sibelius format to finally print their scores. Given all the unique features of plaintext based workflows it would be desirable to convince these publishers to also accept LilyPond/LaTeX files for their publications, but of course one can't expect this to happen in one go. But the advantages of plaintext and version control are so massive that it still makes sense to go through the complete preparation process of an edition using these tools and only convert the material to the more common commercial formats as a last step.

Therefore an option to export scores to the common interchange format MusicXML is one of the most important LilyPond desiderata from the scholarly perspective. There has been discussion about exporting the "raw" musical structure versus the complete (supposedly superiour) layout information. But for the intended purpose the latter isn't necessary at all, quite the contrary. The most pure representation of the musical content will interfere the least with the presets and standards of a post-processing engraver and his tools.

A solution will have to take several factors into account. First it has to capture the "music stream" containing all musical information. Then it will have to parse the input files themselves because they contain information that isn't part of the music stream. Finally it will have to merge all available information to a reliable MusicXML representation.

Currently there is some work going on in this direction, but it can't be said yet how much work, discussion and development is necessary to come to a working, versatile and reliable solution that ensures usable results with minimal need for manual post-processing. So this is a project that should be part of a "research" project that is able to raise enough funds to finish it successfully.

Once this is working – or at least clearly heading towards a working solution – it should be considered to enhance the concept to also embrace the MEI[1] concept. Using LilyPond as a rendering engine seems like an attractive idea if it can also be made possible to re-export scores edited in Frescobaldi to MEI files.

3.2 Implementation of a draft mode

There have been very successful experiments with using a "draft mode" for compiling LilyPond scores. During the development of a score many aspects of it can be visualized, mostly by the use of colors but also by adding elements that don't change the layout. For example we colored editorial additions, items to be discussed or items that have been manually positioned. Graphics that are added to the score may be the control-points of slurs etc., the reference points of objects or verbal comments.

While this is already working quite well there currently is a project to incorporate this functionality into LilyPond itself to make it generally available. As a first step a significant enhancement has been added to Frescobaldi's Preview Mode which is designed to be moved to LilyPond itself as soon as it has proven to be sufficiently stable. This isn't exactly specific to scholarly work, but it has proven to positively affect work on scholarly editions.

3.3 Support for PDF Layers

The ability to create multiple layers[2] in the output file and selectively place items on them would increase the impact of some of the other improvements, in particular `lilypond-doc`, the editorial toolbox and the draft mode. One wouldn't have to compile a score *with or without* features such as annotations or control-points highlighting but could generate a layered file and easily compare in the PDF viewer by switching layers on and off.

In a first step one would only implement the basic infrastructure to be able to print items on different layers, without affecting the layout at all. This will enable applications like those described in the previous paragraph.

In a next (optional) stage one could then extend the concept and think of layers that do or do not affect layout decisions (e. g. collision handling). With this technique one could start to think of advanced edition techniques such as to provide alternative readings in one document. But while the basic outputting technique is nearly trivial this whole topic is very complex and would only be manageable in the context of a research project. What makes this interesting is that this research could break new grounds for thinking about cross media publishing with scholarly editions – for which plaintext file formats are a very suitable foundation anyway.

[1] `http://music-encoding.org`

[2] These are called Optional Content Groups,
see `http://acrobatusers.com/tutorials/print/creating-and-using-layers-ocgs-with-acrobat-javascript`

4 Additions to Frescobaldi

Frescobaldi[1] is probably the most complete editing environment for LilyPond files today. Apart from its great interface and huge number of useful tools it is especially interesting from the perspective of the current document because it's so easily accessible and modifyable. It is actively developed, it is comparably easy to add new features to it, and it would also be very easy to create a custom version with tools specific to a project or institution. While the following ideas aren't really specific to scholarly work I'll describe them anyway because they give a good impression of the current options and the direction of future development.

4.1 Interface to lilypond-doc

One of the most interesting features in current Frescobaldi development is the enhancement of its Music View. There are several plans to enable graphical editing capabilities in order to increasingly blur the borders between graphical and source code editing while not compromising the advantages of the latter.

One project of particular relevance for our current cause is to implement a music view interface to `lilypond-doc`, so that annotations can be inserted or edited directly from within the graphical score. This will dramatically improve the integration of preparing a scholarly edition.

4.2 Partial/automatic Compilation

One issue when working with tools that "compile" plaintext files is the time one has to wait for a successful compilation. This can't be dealt with purely by increasing processing power and more efficient algorithms because it's part of the fundamental concept. One strategy to deal with this is to find solutions to selectively compile only the relevant portions of the score that are currently being worked on. For example I have set up a project for the edition of a large orchestral score where any editor only works with a very small segment file (one voice over the range of one rehearsal mark). When actually working on that segment he will only compile this specific segment which will compile really fast, while the complete instrumental part or the score have to be compiled separately.

Frescobaldi development intends to experiment in this direction to offer partial compilation, and also (optional) automatic recompilation, for example triggered by graphical edits in the music view. Of particular interest for our cause would be options to compile parts of the score out of the layout context of the complete score. While this of course doesn't make any sense when working on the typographical perfection of a score it may significantly improve the working experience when dealing with the *content* side of an edition.

[1] `http://www.frescobaldi.org`

4.3 Support for Workflows with Version Control

Frescobaldi could be made aware of the Git project context a given score is in. That is, it could get a notion of different versions of a score, whether they are in different branches or at different points in history of a project. For example it would be possible to add information about the specific commit that a resulting PDF file belongs to and automatically open the source files in exactly the state they were in when producing a given score. This way it would be easy to compare different versions of a score side by side, e.g. the original and the proof-read or (more useful) the default with a beautified version.

Other Git features that could be incorporated are (for example): 1) Display the differences of a document against the last committed state and selectively stage or revert current changes. 2) *Git blame* to display who last modified a given fragment of the input file.

Additionally one could think about integrating work with remote repositories and develop the notion of a shared session in order to enable "live" collaboration on a score over the network. In order to be useful these ideas would have to be thoroughly discussed, but I think they could have an enormous impact on the way we think about collaboration in musicological work.

Currently I'm working on the foundation for such an enhancement by developing an infrastructure to manage *program versions* and automatical updates of Frescobaldi through Git. While "lacking" most of the versioning complexity of project management it is an opportunity to gain experience with the topic and will provide a solid foundation for future development.