LilyPond for pyScore: Project Report

Stephen Sinclair

April 28, 2006

MUMT 611, Winter 2006
McGill University

1 Overview

For my seminar project, I chose to work with computer languages for representation of symbolic music. Specifically, I extended a format translation framework called pyScore, written by Michael Drottboom (Droettboom 2006). It is implemented in the Python programming language, and is modular in design. Previously, it translated between the GUIDO (Hoos et al. 1998) and MusicXML (Good 2001) formats, and also had output-only support for MIDI files.

I became interested in the LilyPond music typesetting system (Nienhuys and Nieuwenhuizen 2003). Its output is high in quality, and it has extensive capabilities for flexible formatting of all kinds of music, from medieval, to classical, modern and contemporary scores. In my non-expert opinion, LilyPond's output seems of higher quality than the GUIDO NoteServer. I created a module which translates the internal GUIDO representation (a tree structure of Python objects) into an internal representation of the LilyPond input format, which is then dumped to a file which can be rendered to a PDF document or PNG image using LilyPond. In this way, it is possible to typeset GUIDO files using the LilyPond software. Currently it handles most of the GUIDO "basic" subset.

2 Method

I started by trying to get an idea of how pyScore was organized. The first thing I did was add a "LilyPond" folder, and added the minimum code (copied and modified from the "MusicXML" folder) to perform a "null" conversion. That is, given a GUIDO tree, it did nothing except exit without error.

After spending some time studying the documentation for GUIDO and LilyPond, and trying out various small tests on both the NoteServer and using LilyPond, I ventured to try doing a simple conversion of metadata. The approach I took ended up being a template for how I proceeded with the rest of the features: first I copied a test file from the "Guido/input" folder in the pyScore distribution, and manually created a LilyPond version that rendered the equivalent output. I created "lilypond.py" which contained objects representing various elements of the LilyPond language, inspired from the similar organization of the GUIDO module. From what I understood of the LilyPond grammer, I based all objects on a class called LilypondElement, which consisted of the following structure:

```
\name { value }
```

in which "name" was a name associated with the element, and "value" could be a list of other LilypondElement instances. When converted to a string, it would resemble the above. All other elements declared in "lilypond.py" would derive from LilypondElement and override its function responsible for conversion to a string.

I wrote a simple function which recursively stepped through the GUIDO tree, generating LilypondElement object collections. I had to override it for the LilypondNote class, which should have tipped me off that I was doing something wrong, but I continued none-the-less.

It turned out this was a bit of a mistake. The LilyPond format is actually composed of atomic objects, such as strings, notes, and symbols, and "commands" – symbols prefixed with a backslash, which have a predefined number of arguments. These can be combined into "expressions", which can embed other expressions.

In the end, the LilypondElement class turned out to be adequate for representing LilyPond output. However, I suspect that it will need to be refactored before attempting to *read* LilyPond data.

I continued in this vein, going through the GUIDO test files one at a time and extending the LilypondElement object and the conversion functions to handle each case.

3 Data loss in symbolic format conversion

After I was able to convert simple streams of notes, and after I'd figured out how to handle tuplets, I took a step back and tried a more "top-down" approach. I studied the specifications for the two languages, and tried to decide which features would be possible to convert, and which would be difficult. I counted 47 features in common, 10 features that were only available in GUIDO Basic, and 37 features only available in LilyPond. These numbers should be taken with a grain of salt, since "feature" included basic things like "stem direction", "slurs", and "tremolo", as well as whole feature sets like "medieval music". However, it is important to note that some information which is differentiated in GUIDO, like diatonic vs. chromatic vs. solfege note naming schemes, are not in LilyPond. Thus, it is impossible to retain this information in the LilyPond format, and so data loss is inevitable. This speaks to an important aspect of the design of pyScore, which chooses to translate "language to language", rather than creating a single "superset" language containing all aspects of every music notation grammar.

The advantage is mainly one of ease of development. Creating a "superset" language essentially amounts to creating a new representation for notation, one that must take into account not only all aspects of every existing notation format, but also must predict what information might need to be retained in future formats. Indeed, both GUIDO and MusicXML are still under development, and may change what they represent (though hopefully not too much). The internal superset format would have be completely extensible. In short, it would have to be better than all of the existing formats.

Since a translator is not in the business of *designing* languages, it only makes sense to translate from one pre-existing language to another. However, it must be acknowledged that long chains of translation will occur in data loss. Thus, a smart translation system should be able to determine the minimal data loss chain. So far, pyScore is able to automatically determine a conversion chain. The capability to quantify data loss along a chain and thus minimize it is not in the cards at the moment, but it would be an interesting prospect for the future. When more formats are supported, it may become more important. Even now, with the LilyPond module, it may be interesting to analyze how much data loss will occur in the path from MusicXML to Guido to LilyPond. This would require methods for quantifying data loss, as well as "weighting" types of data loss in musical notation. For example, note name representation should weigh less than losing dynamic markings or lyrics.

4 Status

As of this date, the module is able to successfully convert 14 of the 21 GUIDO test files in the pyScore distribution. The following cases present some problems for conversion:

Problem Area	*Explanation*
Key signature	It almost works, but some aspects of the conversion seem to fail. The test file, key.gmn, converts properly, but notes.gmn seems to have an incorrect key signature.
Lyrics	I have not attempted to implement lyrics yet.
Cross-staff	I have not attempted to implement cross-staff notation yet.

Octavation	The relationship between when octavation is turned on, and the position on the staff where LilyPond positions the notes is not entirely clear. It seems to be related to the previous octavation position, but I have yet to determine a conversion that is simple and works reliably.
Cue notes	Cue notes are implemented very differently in the two languages. In GUIDO, a "\cue" tag is followed by a bracketed expression which is rendered in parallel to (on top of) the notes which follow it, but in a smaller notehead-size. In LilyPond, a "quotable" expression must be used, which is quoted as cue notes from another voice. In other words, the link between two instruments is maintained in the LilyPond format, while it is not maintained in GUIDO. Thus, converting cue notes implies creating an instrument voice which appears earlier in the document to contain the cue notes, and then quoting them in the actual expression. This can be done by preceeding the cue expression by the correct amount of rests, but I have not implemented it yet, as it involves "inserting" information into a previous part of the tree.

5 Conclusion

The domain of musical notation representation is particularly interesting, because music is just structured enough to seem a surmountable problem, but breaks the rules in just enough ways to make it a difficult one.

It makes sense that such a problem would have developed several solutions over the years, and like any domain, translators between these solutions are required. A framework such as pyScore can enable this idea of a "universal translator", enabling people to transfer information between software packages, and ultimately, allowing *choice*. For me, software is about choice, and the last thing that should stand in the way is incompatible data formats. There is no feeling quite like the sense that you have painted yourself into a corner due to a choice you made several years earlier.

As for music, and how to represent it, I can't help but quote the essay on the LilyPond website (Anonymous 2006):

> ... The ideal input format for a music engraving system is the content: the music itself. This poses a formidable problem: how can we define what music really *is*? Our way out of this problem, is to reverse it. Instead of defining what music is, our program serves as a definition: we write a program capable of producing sheet music, and adjust the format to be as lean as possible. When the format can no longer be trimmed down, by definition we are left with content itself.

Thus, music representation, like any problem of knowledge representation on computers, is an exercise in formalization. In formalizing music, we are bound to discover new things about it, and how we perceive and work with it. I cannot help but think that such things are really cognitive science in disguise.

References

Anonymous. 2006. Automated engraving (unpublished essay).
Available online: `http://lilypond.org/web/about/automated-engraving/`.

Droettboom, M. 2006. pyScore (Software).
Available online: `http://pyscore.sf.net/`.

Good, M. 2001. MusicXML: An internet-friendly format for sheet music. In *XML 2001*. Publication.

Hoos, H., K. Hamel, K. Renz, and J. Kilian. 1998. The GUIDO notation format: A novel approach for adequately representing score-level music. In *Proceedings of the International Computer Music Conference*.

Nienhuys, H.-W. and J. Nieuwenhuizen. 2003. Lilypond, a system for automated music engraving. In *Proceedings of the XIV Colloquium on Musical Informatics*, Firenze, Italy.