Random Sheet Music Generation Project: Design

Author: Cedric McDougal
February 11, 2011

Inputs

Input will be stored as plain text in a file. One of the parameters to the program will be the name of this file (see the *Execution* section below). Inputs will be specified in XML format in order to be easily parsed, yet still readable to users. The name of each XML tag will describe the property being set, and the value will be specified between an opening and closing tag. An example might be:

```
<complexity>67</complexity>
```

The properties available for the user to specify will be:

- complexity level
- key signature
- time signature
- number of bars
- tempo
- style
- range
- instrument

The available values for each property will be specified in the user manual. If an improper value is entered, the syntax of the input is incorrect, or the input file doesn't exist, the program will display an informative error to the user, explain the expected value/syntax, and offer an example.

All properties of the music *not* specified by the user will be generated at runtime (see the *Algorithms* section below).

Outputs

Output will be stored in a file that can be passed as input to the LilyPond application (which will be used to generate the graphical sheet music). The name of this output file will be a parameter to the program (see the *Execution* section below). If the output file does not already exist, a new one will be created. If it does already exist, the user will be prompted to overwrite the existing file.

Compiling

The program will be compiled using a makefile. As such, the steps for the build process are as follows:

1. Use any method to access the Linux command line on one of the CCIS computers.
2. Navigate to the directory that contains the source code (it will all be in one directory, not split up into separate packages).
3. Type "make" and press the *Enter* key.

Execution

The program will be executed from the command line and will run on one of the CCIS computers that uses Linux. The syntax for executing the program is as follows:

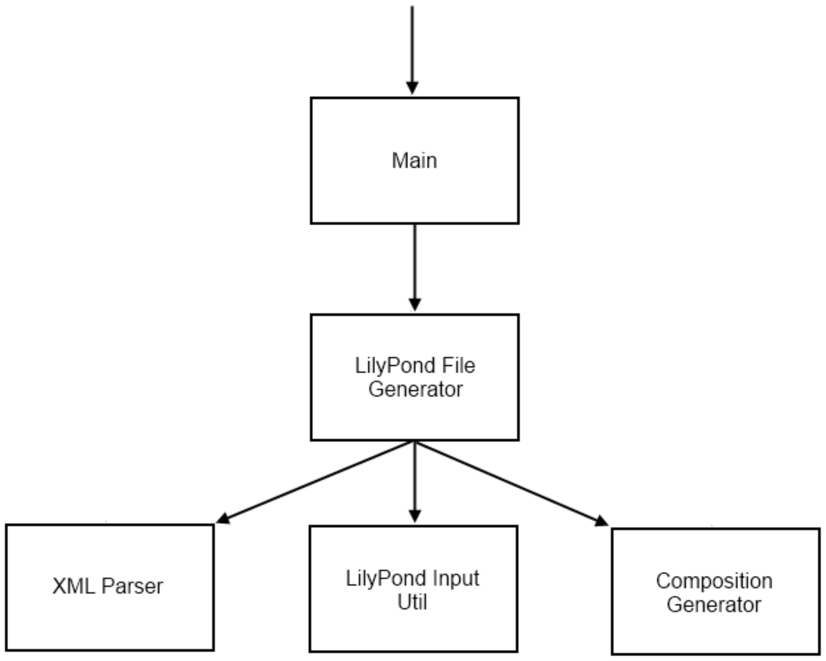
```
./c4500 <inputFileName> <outputFileName>
```

where `<inputFileName>` is the name of the file that contains the XML specifying the input properties and values (see the *Inputs* section above) and `<outputFileName>` is the name of the file that the output should be written to (see the *Outputs* section above). This command will execute a shell script that will in turn execute the compiled application.

Architecture

The application will be primarily comprised of a set of tools that perform specific functions. There will be a single class that will utilize each tool in order to construct a cohesive output. This design will maximize efficiency of development because it allows each team member to work in parallel on the different tools without any overlap of work or dependency conflicts.

Below is a diagram with the proposed tools and the class that will tie them together.



Main

This class is the entry point for the application. It will contain the main method that takes command line arguments, and will pass these arguments to the *LilyPond File Generator* class.

LilyPond File Generator

The main purpose of this class is to use the available tools to compile the output text and write this text to the specified output file. These are the general steps it will take to accomplish this task:

1. Extract the XML data from the given input file.
2. Use the *XML Parser* to parse the input into a format that is internally recognizable.
3. Pass this input to the *Composition Generator* to generate the notes in the music.
4. Pass the parsed input parameters and the generated music to the the *LilyPond Input Util* to convert it all into a format that can be recognized by the LilyPond application.
5. Arrange values returned by the *LilyPond Input Util* into a format that can be recognized by the LilyPond application.
6. Write the arrangement to a file.

XML Parser

This will be a simple XML parser developed specifically for this application. As such, it will not be very robust in functionality, but this should make it quicker to develop, debug, and run.

LilyPond Input Util

The purpose of this class is to take values in the internally recognized format and convert them to a format that can be recognized by the LilyPond application. For example, one method might perform the following function:

```
getTempo(120) → “\tempo 4 = 120”
```

This will reduce the complexity and increase the flexibility of the internal representations because they will not rely on the LilyPond syntax. Furthermore, if the LilyPond syntax changes in the future, it will be easy to update our application to support these changes.

Composition Generator

This tool will be responsible for generating the notes in the piece of music. As such, it will need access to all user-specified values. It will then have to use these values as a framework in which pseudo-random music can be created.

To simplify the music-generating algorithm it will at first ignore certain inputs, like the key signature. After the music has been generated, it will be transposed to the form requested by the user. This task may require a separate tool, depending on its complexity.

The algorithm will take a top-down approach, in which the more general aspects of the music will be chosen before the more specific aspects. All available data will be contained as static constants in a separate class. The steps will be as follows:

Select a pattern of sections
The composition will be comprised of a repeating pattern of sections, such as AAB, or AABA. These patterns will be predefined. The number of sections will depend on the number of bars. Each section will contain some number of phrases.

For each section, generate a pattern of phrases

The pattern of phrases will be more diverse than the pattern of sections to ensure that the generated music is suitably interesting for sight reading practice. The patterns will be pseudo-randomly generated based on a randomizing seed and the complexity level. The number of phrases and length of each phrase will depend on the number of bars. Each phrase will contain some number of chords.

For each phrase, generate a series of chords

The chords will be generated singularly and in order. To generate each chord the algorithm will:

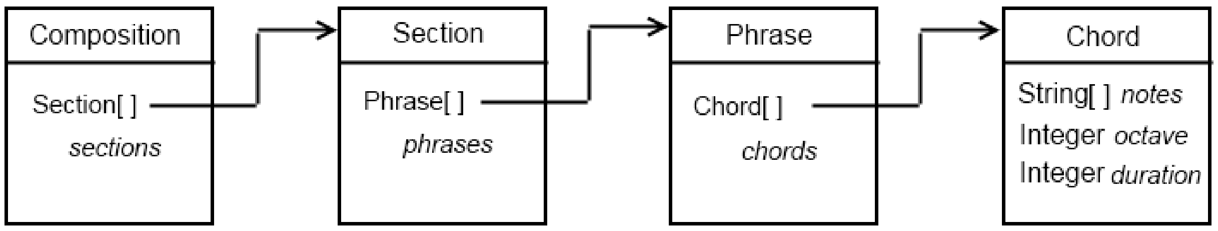
1. Pick a pseudo-random duration within some range. The range will be determined by the instrument, the complexity level, the style, the length of the phrase, and the previous selections.
2. Pick a pseudo-random chord from the set of available chords. Each set of chords will have a different probability of being chosen. The most likely will be the first, fourth, and fifth triads of the scale. The probability will also be affected by the complexity level, the style, and the previous selections.
3. Pick a pseudo-random selection of the notes within the chord. The size of the selection will range between one and all of the notes. The probabilities driving the selection will depend on the instrument, the complexity level, the style, and the previous selections.
4. Pick a pseudo-random octave within a certain range. The probabilities driving this selection will depend on the instrument, the range, the complexity level, the style, and the previous selections.

Internal Representations

The internal representations of the user specified data will be as follows:

complexity level	: Integer	: how difficult the piece is to play
key signature	: String	: the available notes
time signature	: String	: the number of notes in a bar and the duration that makes one beat
number of bars	: Integer	: the number of bars of music in the entire piece
tempo	: Integer	: the speed at which the music should be played
style	: String	: the types of intervals and patterns that are used
range	: String	: a range of pitches for a certain instrument
instrument	: String	: the instrument on which the music will be played

The internal representation of the song itself will be as follows:


Composition
Section[ ] sections
Section
Phrase[ ] phrases
Phrase
Chord[ ] chords
Chord
String[ ] notes
Integer octave
Integer duration