MuseData to GUIDO Converter

by Martin Friedmann and Kai Renz

2002

as part of the GUIDO toolbox

http://www.salieri.org/GUIDO
http://www.noteserver.org

1 Preface

This text describes the development of a tool which converts single MuseData-files to GUIDO Music Notation (gmn) files. The main focus lies on the data formats and the algorithms used to convert them.
Both formats, MuseData and gmn are used to represent music in a computer-readable way. Nevertheless they are quite different.
At the MuseData-project people have taken great effort to transcribe classical music to the MuseData-format and store them in a database on the web. At the Salieri-project people are interested to use this database to test their computer-music related tools, which use the gmn-format to store musical data.

2 Problem Analysis

This section describes the similarities and differences between the music-representations used in MuseData and Guido. To do this, both formats are described briefly. It is not intended to describe the whole formats for MuseData- and gmn-files. Instead, only a short overview of the formats is given. More detailed descriptions of single features are given later when needed.

2.1 MuseData format

In MuseData the file-format is very strict: Every file has to start with a header containing information about the piece represented within the file and how the file is related to other files (it is possible, to have more than one file for the same piece, for example to have every voice in a single file). After the header, the music is represented by records. There are various types of records, but all of them share some characteristics: all records have a maximum length of 80 characters and each position in the record is used for a special purpose. The records used here are:

- note-records, which store information about single notes. Information stored includes pitch, duration, graphical representation[1], slurs, ties, beams, chords, cue- or grace-notes. There are four different (but quite similar) type of records for note, notes in chord, cue- or grace-notes and cue- or grace-notes in chords. To make analysis of polyphonic pieces easier, note-records may also bear information on the voice to which the note belongs.
- measure-line-records
- musical-direction-records, which hold information about tempo-changes, changes in clef and meter etc.
- combine/divide-records, which allow to go forward and backward for a given duration. Using these records to go backward is the only way to create polyphony in a single file

There are other record types like print-suggestions and sound-records which are not used here.
To represent a piece of music, the notes of the piece must be encoded as a list of note-records. Normal note records are understood to be played one after the other. If a note record is followed by one or more chord-records, the note-record and the chord-records are played as one chord. The next note-record after a note- or chord record is played, after the preceding note/chord has been played. If more than on voice shall be represented in one file, it is necessary to “go backward in time”, to have a second note at the same time. To do this, combine/divide-records are used[2]. By using these records, the voices of the piece are interwoven.
Note records may (but need not to) include information on the “track” of which the note is part. This may make it easier to understand which note is part of which voice. Another way to split the voices of one piece is to put them in different files, of course.

2.2 Guido-Music-Notation

In a gmn-file, the musical information is stored in one or more voices. The voices are stored one after the other. Except for chords there is no polyphony in a single voice.
A voice consists of notes and tags. Notes hold information about pitch, octave, alteration and duration of the notes. Information which does not change from one note to the next may be omitted. Tags are used to represent anything that is beyond the mere notes. Examples for things which may be represented by tags are changes of tempo or intensity as well as ties, slurs or beams. Tags which influence on or more notes normally mark the influenced range of notes in a pair of brackets. As this is only possible, if ranges don’t overlap, a second way for marking ranges are pairs of ...Begin and ...End-tags. For many tags both styles of ranges are supported.
The gmn-file has no strict format. Instead of this, notes and tags may be one text, with linefeeds and comments when needed.

[1] The graphical representation of a note’s duration needs not to fit to the notes musical duration.
[2] In the MuseData-files used to test the converter, the divide-records where always at the end of the measure, but there is no need to do it in this clear way.

2.3 Comparisson

As described above, the two file formats are quite different. In gmn the timeline for one voice goes only forward. After one voice has finished, the next voice is described. In MuseData all voices are represented overlapping in one long list jumping forward and back in time.
In MuseData all information concerning one note is stored within the note's record, while in gmn appearance and expression of a note may be influenced by the surrounding tags.

2.4 Concept for conversion

MuseData and gmn choose quite different ways to represent music. To convert MuseData to gmn, the MuseData-file has to be analysed and afterwards a gmn-file has to be created from the outcome of the analysis. So there are two main tasks:

- Analysing the content of the MuseData-file
- Creating a gmn file

If the MuseData-file is analysed correctly, it is quite easy to create a gmn-file, the harder part is the analysis. To perform this task, it is decomposed in the following sub-tasks:

- Parsing the MuseData file
- Calculating absolute times for the records
- Marking chords
- Sorting records by time
- Splitting the notes to voices
- Analysing the voices for ranges of notes which are somehow linked (for example ties, slurs or chords) and marking beginning and end of the ranges

The details for these tasks will be described in the following sections.
No special efforts have been taken to convert pieces which are stored in multiple files. It is quite easy to convert every file alone and later link the voices from the gmn-file to one larger gmn-file.

3 Analysis and internal representation of the MuseData-file

In the following sections, the steps for analysing the MuseData-file are described in more details.

3.1 Parsing the MuseData-file

The first step for conversion is, of course, to parse the MuseData-file. To do this, the file is read line-by-line. In the MuseData-file-format each line is one record. After reading a line from the file, this line is converted to a record-object, by examining the line's content and storing the (explicit) information in the new created object. As the type of a record is determined by the first character of the line and most information within the line has a fixed position this task is quite easy.
All record-objects are stored in a linked list after reading them.
After this step of the conversion, the information given explicit is stored in one double-linked list of record objects.

3.2 Calculating absolute times for records

Within the MuseData-file the times which apply to a record are not given explicitly. Instead of this, the time has to be calculated from the durations of the preceding notes. This is done by going over the list of records one time and accumulating the durations of the single notes. Special care has to be taken for chords, as the durations of the notes may not be added up. Instead of this, the time of the first note in the chord is used for all notes, the duration of the first note is taken for the whole chord, the other durations are omitted. Another thing which has to be taken care of are combine/divide records, which change the time-position for the following records.
After this step of the conversion, the time for each record, which was given only implicit within the MuseData-file is stored explicitly in the records. It is now possible to remove the combine/divide-records from the record-list, as all their information is stored now in the other records and they are not needed anymore.

3.3 Marking chords

Now the chords have to be marked explicitly to make the following steps easier. By now the only information about a chord is given by a flag in the notes following the first note of the chord. It is not possible to recognize a chord by looking at it's first note!
To mark the chords, two substeps have to be taken.
At first, the notes of one chord are linked together. To do this, the list is traversed one time, whenever a note-record is marked as a following note in a chord, the record's predecessor is linked to this record. These new links do not change the link-structure of the main record-list. They just hold extra-information about the chords.

Now more implicit information about the chords can be extracted. To do this, the list is traversed a second time. Whenever a new chord is found, the chord's duration is stored in each note of the chord. As there may (but need not to) be information on voices in the note-records, and as it may be that a chord is split over several voices, it is determined for each note of the chord, if the note is the first, the last, or a note from the middle of the chord in it's voice.
If a chord is split over several voices, the chord is also given an unique label-number, which will be used later to mark chord-parts in several gmn-voices as connected.
After this step, for any note which is part of a chord, the note's position in the chord (first, last, in the middle) is known. The duration of the chord is also stored in any note. This redundancy is necessary, as the chord may be split over several voices. Any information to create gmn-chord-data now is explicitly known within each note record. It is not necessary anymore to know any other note record of the chord to create tags.

3.4 Sorting records by time

Until now, the records in the record-list have the same order they had in the file. Now they are sorted by their time, thus destroying the old order.
To sort the list, the following approach is taken: The list is traversed beginning from the first record. Each record's time is compared to it's predecessor's. If the predecessor's time is larger, the current record is moved one position back in the list. Comparison than is repeated for the current record.
This algorithm is finished, once the last record of the list needs not to be moved anymore.
After this step, the records are sorted by time.

3.5 Splitting voices

As there is no polyphony in a single gmn-voice, the musedata-representation has to be split into voices. This can be done just now, after the note-records are sorted by time and overlapping ranges can be recognized.
Splitting is done in two passes. In the first pass, the record list is split up into one record list for each explicitly given voice. All notes with no information on their voice remain in one list.
In the second pass, each of the new lists is checked for overlapping notes. If there are overlapping notes within on list, the list has to be split in subvoices. This is done by iterating over the list from the beginning and moving as many records into a new list, as fit without creating overlapping notes. This can be done simply by comparing the ending time of the last note in the new list with the next note in the existing one. If the starting time of the next note is after the end of the last one, it can be moved to the new list, if not, it has to remain where it is. After going over the whole list, it is checked, if the list has still overlapping parts, and if this is the case, another subvoice is taken from the list. This is done repeatedly, until there are no more overlapping parts in the voice.

3.6 Marking ranges

The final step in transforming the internal representation is the marking of ranges. Ranges are needed to represent information which links several notes together like slurs, ties or beams. In musedata there is no information about this link stored, but the single parts or a range are just marked as being part of a range. In contrast to this, in gmn ranges are marked at the beginning and the end. For every type of range, the record has one of the following five states: being part, being not part, being beginning, being end and being beginning and end. The following subchapters describe the measures taken to mark the different kinds of ranges. These measured have to be taken for each of the voices.

3.6.1 Ranges for cue- and gracenotes

Cue- and gracenotes are handled similarly. Within the musedata file they are marked with a single flag as being cue- or gracenote. This flag is stored in the internal representation. To mark these ranges a very simple algorithm is used: The algorithm iterates over the whole list and stores in one variable, if it is within a range or not. Pointers to the current record as well as the last note-record are stored. Now there are four cases which may occur:

	note is part of cue/grace-range	note is not part
last note was part of range	mark current note as being *part* of range	*current* note is *not part* of range *last* note is *end* of range
last note was not part of range	mark current note as being *start* of range	current not is not part of range

If the last note of the list was part of the range, it is marked as being the end.

3.6.2 Ranges for ties

In musedata ties are simply marked by setting a flag for any note which is tied to the next note. While converting the file every tie in the whole file is given an individual number. As the conversion-algorithm is run separately on each voice, it gets the next number to use (number are used consecutively).

The algorithm uses an array to store for each possible pitch, if it is currently part of a tie and the tie's number. This array is initialised for each voice. Now the algorithm iterates over the whole list. For each note-record it is checked, if the note is tied to the next one. Depending on the tie-state, there are four cases:

	tie exists for current note's pitch	no tie exists for current note's pitch
current note is tied to next	nothing has to be done	pitch is marked as tied tie's number is set for pitch note is marked as start of tie with current number tie-number is incrmented
current note is not tied to next	note is marked as end of tie with pitch's tie number pitch is marked as not tied	nothing has to be done

3.6.3 Ranges for slurs

For slurs beginning and ending of the range are already marked in musedata. There can be up to four open slurs at the same time. Like for ties, each slur is given an individual number while converting. As beginning and end are already marked within the note-records, only the numbers have to be calculated.
Conversion is done very easily. The algorithm iterates over the list and checks for each record, if one of the four possible slurs opens or closes at this note. If a slur opens at a record, the number for the slur is stored within the record and stored for the slur, afterwards the number is incremented. If a slur closes at a record, the slurs number is stored within the record.
With slurs there can appear three kinds of errors:

1. A slur is being opened, which is already open.
2. A slur is being closed, which was not opened yet.
3. A slur is open at the end of the voice.

Error-handling is quite easy:
Cases 1 and 2 can be ignored (just a warning is generated). To handle case 3 a pointer to the starting record of each slur is stored. If at the end of a voice a slur is still open, the open-information in the starting record is removed by setting the slur-number to zero (and a warning is generated).

3.6.4 Ranges for beams

In musedata there can be up to six beams between notes. While parsing the note record, it is checked, how many beams link a given note to the next, and how the number of beams changes at this note (both informations are given within the musedata file for each individual record, they need not be consistent of course).
To mark the beam ranges, the algorithm iterates over the voice and check for each note, if the beam-information is consistent with the last note. Then it calculates, how the count of beams changes at this note (this information is later used to open and close single beams).
If after the last note there are still open beams, the algorithm generates a warning and adjusts the beam-change for the last note, so that all beams are closed.

4 Creating gmn-files

After the musedata file has been transformed and analysed, the data can be used to create a gmn file. To do this, the proper gmn tags and notes have to written for all voices analysed before. To do this, each voice iterates over it's records, and each record generates the proper tags. The easiest way for this would have been to simply print all tags for every record in consecutive order to a file. This method has a big disadvantage: As it is possible, to improve readability of gmn-files by leaving out redundant information, the printing-information had to be shared between the records to generate an optimised output. This would have lead to unstructured and hard-to-maintain code, so another approach was chosen.
Instead of printing the tags directly to a file, the printing-methods for the records just call methods of a special gmn-printing-object. This printing-object can store information about tags printed before and thus optimise the output. It also is possible, to choose between several options for gmn-creation.
In the following paragraphs, some of these printing and optimising methods are described.

4.1 Creation of notes

A note in gmn carries information about the pitch, the octave and the duration of one note. Octave and duration need not be specified for a note, if they do not change from the last one.
Whenever a note has to written to the gmn-file, a printing method for note information is called. This method compares the current note to the last note and only prints the needed information to the file. For the first note of a sequence all information is printed.

4.2 *Stems and staves*

In gmn it is possible, to choose the staff and stem-direction for every note, so in every note-record this information is stored, and for every record the according printing-methods are called. Nevertheless tags are not created for each note, as the method stores the last staff and stem-direction and only creates tags, if there is a change.

4.3 *Clef, key and meter*

If changes of clef, key or meter occur, the according printing-methods are called, separated by calls setting the staff. Key and meter are set to the same value for each staff, clef may differ from staff to staff.
There are two methods how these calls are handled by the printing-object.
In simple-mode, the tags are just created, as the methods are called, in smart-mode, only the changes in key, clef or meter are recorded, the tags are only created, if a note is printed on the staff where the change occured.

4.4 *Beams*

In each note-record there information is stored, of how the number of beam changes at this special note. If the number increments, a method is called (*before* creating the note tag), to tell that n beams start at this note. If the number decrements, a method is called (*after* creating the note tag) to end n beams. There are four ways to handle these events:
In auto-beam-mode, at the beginning of the voice automatic beaming is enabled, and no further tags are creaged.
In no-beam-mode, beams are disabled at the beginning of the voice and no further tags are created.
In single-tag-mode, only one pair of \beamBegin and \beamEnd tags is created, when the number of beams rises above zero and falls to zero again.
In multi-tag-mode, for every increment on the beam-count, a \beamBegin tag is created and for every decrement a \beamEnd tag is created.

5 Program realisation

The implementation is realised as an object-oriented class hierarchy, which maps all of the previously described algorithms and data structures to a set of classes. The used naming convention makes it easy to identify the scope of any given class.
To represent the several types of records from a Musedata-file, there is one general class for records (holding the general information), and several subclasses of this for the special types of records. Each of these subclasses has the necessary functionality to convert the represented record-type. All kinds of records are stored in list-objects, which also implement the functionality needed to do the described algorithms on the records. After reordering and splitting the record-objects, each kind of record-object holds all information needed to generate the gmn-file. Thus there is no special representation of the gmn-data.

6 Command line options

There are several options to influence the conversion. They can be influenced by the following command-line options:

- Inputfile: the last word in the commancline is considered the input-file
- Outputfile: with `o`*`Filename`* the name for the outputfile is set to *Filename*
- Beams: There are three modes for the creation of beam-tags:
  - `bn`: no beams at all (creates \beamOff-tag and nothing else)
  - `ba`: autobeams (creates \beamsAuto-tag and nothing else)
  - `bs`: (default) single beam tags (every group of beamed notes is enclosed in a pair of \beamBegin and \beamEnd-tags.
  - `bm`: multiple beam tags (nested pairs of tags are created according to the Musedata-file)
- Clef, Key and Meter:
  - `s+` / `s-` switches smart-mode on and off (default is off)
- Print-options: the following options influence the formatting of the gmn-file:
  - `pm+` / `pm-` printing of linefeed after each measureline on / off (default is on)
  - `pt+` / `pt-` printing of track- and subtracknumber before each sequence on / off (default on)

7 Possible expansions

There are several possibilities to expand the converters' functions, which have not been implemented for reasons of time. Some of them are listed here:

The musedata-specification defines so called “print-suggestion-records”. These records can carry information about the way, musical symbols shall be printed. In combination with advanced GUIDO it should be possible to improve the appearance of scores printed from the generated gmn-files.
At the moment the converter generates one gmn-file from one musedata-file. Musedata also supports the encoding of one piece in multiple files (e.g. for voices). A possible solution to process such pieces would be to write a small wrapper which would convert the single files and after this merge the gmn-files to one file.
Currnetly the converter does not support all kinds of information encoded in the musedata-note-records. Markup like trills have been omitted, they can be easily implemented later by expanding the parsing-method for note-records and the printing-methods.
Another interesting project would be to link the musedata-database and the GUIDO-notesever to allow on-line generation of scores on the web.

8 Literature

The following literature has been used while doing the project:

GUIDO-specification at http://www.salieri.org/
Beyond MIDI – The Handbook of Musical Codes, Edited by Eleanor Selfridge-Field, MIT Press, 1997
Stroustrup, Bjarne, The C++ Programming Language, Second Edition, Addison Wesley
dtv-Atlas Musik, Band 1, 19. Auflage; Deutscher Taschenbuch Verlag, München; 2000