Design and Implementation of a platform independent
GUIDO Notation Engine

Kai Renz
Darmstadt University of Technology
Wilhelminenstr.7, D-64283 Darmstadt, Germany
e-mail: renz@iti.informatik.tu-darmstadt.de
http://www.informatik.tu-darmstadt.de/AFS/GUIDO/

Abstract. This paper describes how platform independent conventional music notation can be realized using the powerful music description language GUIDO in conjunction with the newly developed GUIDO Notation Engine (GNE). This new, fast, flexible, and very portable approach to platform independent music notation can be used for a broad range of applications, for example to display and edit conventional music notation on WEB pages or to realize score display and editing within musical applications running on almost any operating system.

Introduction and Background

GUIDO Music Notation (GMN) was first presented in 1998 (Hoos et al., 1998); it offers a simple yet powerful means of representing musical as well as notational aspects of a large variety of music. As was shown in (Hoos et al., 1999), GUIDO Music Notation is also capable of representing advanced score formatting information, thus making it a suitable platform independent notation interchange format. For these reasons, a growing number of applications use GUIDO as a general music representation format.

The increasing interest in using GMN for storing, transporting, and interchanging musical data evoked the need for a flexible and easy to use GUIDO renderer that converts textual GUIDO descriptions into conventional music notation, which can be displayed, printed, and graphically edited using a variety of operating systems and applications. Some of this has been realized with the GUIDO NoteServer (Renz and Hoos, 1998), a publicly available free service on the WWW that converts GUIDO descriptions into conventional music notation, which can then be embedded in web pages.\(^1\)

Even though this service is widely used, it requires a permanent connection to the Internet making it hard to use for a range of people. Therefore, the original idea of the GUIDO NoteServer has been extended towards a really portable GUIDO Notation Engine (GNE), a flexible and extensible tool for dealing with all notational aspects of GUIDO. Because of its platform independent design, the GNE runs on many computer platforms and interacts with different applications in a variety of ways, including components for the WWW. The GNE can for instance be used to display and edit conventional music notation within musical applications that do not provide conventional notation on their own.

In this paper it will be shown how the GUIDO Notation Engine can be used for score display and editing under Windows, Linux, Mac OS, and other operating systems supporting JAVA. Furthermore, the general object oriented structure of the GUIDO Notation Engine is introduced and it is shown how the GNE converts arbitrary GUIDO descriptions into conventional music notation; this is by far a non-trivial task: depending on the degree of specification of the GUIDO data, a multitude of notational information must be inferred automatically. This includes, among others, automatic beaming, automatic inference of bar lines (including automatically splitting notes and rests), automatic detection of tuplets, automatic detection of system and page breaks, etc. The object oriented design of the GNE allows for a flexible exchange of any integrated algorithm, so that, for example, a different spacing model can be embedded easily. Furthermore, it is always possible to explicitly specify all or some notational aspects directly in the GUIDO description, thus overriding the internal algorithms. Among others, these features distinguish the GUIDO Notation Engine from any other notation software.

Using the GUIDO Notation Engine

The general architecture of the GUIDO Notation Engine is shown in Figure 1. As can be seen, the GNE consists of two parts: a platform independent

\(^1\)The GUIDO NoteServer is accessible at http://www.informatik.tu-darmstadt.de/AFS/GUIDO
score renderer realized using C++ and a JAVA based front end for displaying and editing the score. Inbetween the renderer and the front end, information is transmitted using the GUIDO Graphic Stream (GGS) protocol; this protocol will soon be published, so that other front ends for the renderer can be realized by interested parties. The general idea behind GGS is the ability to develop lean music notation front ends that do not require score rendering intelligence themselves but rely on a portable renderer therefor. First a device independent score is assembled by the renderer, as will be described in the next section. Then all graphical elements of the score are transmitted using GGS: each object transmits a unique id, a position and additional information (like size, color, etc.). Some graphical elements (e.g. beams or slurs) additionally provide their coordinates. Because each object has a unique id, the front end can later interact with the renderer to obtain detailed information on any object (for example how it can be moved, what other objects need to be moved along with it). Note that the renderer and the front end do not have to reside on the same machine; communication between the renderer and the front end using GGS can be done in a local or global network.

The GNE provides a platform independent interface for other applications to use and can be accessed like a notation server in a client server model. Because of its platform independent design, the GNE can either be run on the same machine as the client application, or it can be run on a dedicated server machine, servicing a number of different clients concurrently. It is also possible to embed the GNE directly into an existing application using a dynamically loadable library. Input to the GNE consists of a GUIDO description and further formatting instructions (like e.g. the zoom-factor, the page number, etc.). The renderer then completely renders the score using algorithms described in the next section. The actual display and editing of the score is done by a graphical front end. We have realized a prototypical front end called JNView based on JAVA, so that platform independence is maintained. It is nevertheless always possible to provide platform dependent customized front ends using GUIDO Graphic Stream. JNView has been developed using Swing and JDK 1.1.8; it can therefore be run on almost any operating system.

Figure 2 demonstrates the truly platform independent nature of the GNE: (a) shows the (still prototypical) interface of JNView as a standalone application running on a Linux machine and (b) is screen-shot of a JAVA applet inside Internet Explorer on Mac OS. For both examples, identical source code was used, thus demonstrating how the GNE can easily be integrated into different applications and operating systems.

Notation Algorithms

To convert arbitrary GUIDO descriptions into nicely formatted scores, a number of data structures have to be build and algorithms have to be applied. Because...
The details of this process are too complex to be discussed here in detail, only some of the algorithms involved in it are discussed in the following. Note that all applied algorithms can be described as GUIDO-to-GUIDO transformations: input and output of each algorithm is a complete GUIDO description, where each algorithm adds specific information to its input if necessary.

The renderer itself is divided into an Abstract Representation (AR) that can be understood as an object oriented hierarchy of the general GUIDO syntax and into the Graphical Representation (GR) which is an object oriented framework of all visible objects of a score.

The rendering process first creates the AR of a given GUIDO description; then a number of algorithms are applied and finally the AR is converted into the GR.

The following algorithms are currently implemented by the renderer:

- **Automatic bar lines**: If meter information is provided in the GUIDO description, bar lines are inserted at the correct time positions within the AR. Upbeats and explicit bar lines are recognized. Notes and rests that overlap the new bar line positions are split and ties are inserted where needed.

- **Automatic beaming**: This algorithm finds groups of notes that can be beamed; it depends on the current meter and on note durations. Explicit beams in the GUIDO description override automatic beaming where they apply.

- **Automatic clef/key/meter**: If no clef information is given in the GUIDO description, a suitable clef is inserted. This algorithm also tracks key and meter changes and handles their graphical appearance.

- **Automatic splitting of note and rest durations**: Arbitrary GUIDO descriptions may contain notes or rests with durations that are not directly displayable as single graphical objects. This algorithm finds tuplet groups and splits notes and rests if necessary.

- **Determination of possible system- and page-breaks**: When trying to find optimal system- and page-breaks, a subset of all possible break locations has to be taken into account (this is somewhat similar to breaking paragraphs into lines in word processing (Knuth and Plass, 1981)). The algorithm determines possible positions for system-breaks by traversing all voices in parallel. Those time positions where notes or rests begin in all voices are marked as possible break positions. In addition, each position is marked with a *goodness-value*: if the time position coincides with the beginning of a measure, the break position gets a higher ranking. Therefore, break positions at measure boundaries are favored.

All off the algorithms described above work directly on the AR. The process of converting the AR into the GR handles mainly issues of spacing and collision detection. Currently spacing is implemented in the ren-
derer using a modified spring-rod-model (Haken and Blostein, 1995), where almost every graphical element is attached to a spring and each voice and staff has a set of rods. For each previously determined break position, the required force for stretching all springs to the system widths is determined. The break position with the optimum force is chosen as the system break. Using this approach, each chosen break point is locally optimal but no global factors apply. We plan to include global hints in future versions. The integrated spacing algorithm can be turned off by supplying explicit spacing information in the \texttt{Guido} description: the \texttt{\textbackslash space}-tag is directly converted into a voice rod. By additionally supplying \texttt{\textbackslash systemBreak}-tags, no automatic spacing and/or line breaking needs to be calculated.

**Conclusion and Future Work**

The \texttt{GUIDO} Notation Renderer is a big step towards platform independent interactive music notation. By supplying an easily portable renderer and a JAVA based, platform independent front end and interface, the GNE can be used by a large number of offline and online applications.

While it was shown that our general approach is feasible, there still remain a number of issues that will be resolved in future versions:

- **Speed**: Communication between the JAVA front end (JNView) and the renderer using \texttt{GUIDO} Graphic Stream has to be accelerated.
- **Open Source**: The source code of the GNE has to be published as open source to increase the number of active maintainers and to help with porting issues.
- **Interaction**: Currently, interaction in JNView is limited to selecting and moving notes. The renderer must implement a constraint model for graphical primitives so that intelligent interactions become possible.
- **Functionality**: The renderer must provide a clean interface for adding and/or exchanging the internal algorithms with user defined routines.
- **Score rendering**: The renderer is constantly being worked upon; almost all internal algorithms still need improvement and still many features found in conventional music notation are not yet supported. A great amount of our future work will be devoted to improving the algorithms of the renderer and adding new features. As all changes will immediately be available on a variety of operating systems, this might be a powerful incentive for other programmers to help developing future versions.

**Acknowledgment**  
I wish to acknowledge the work of Christian Trebing and Martin Friedmann, who have helped me in developing JNView. I would also like to thank Holger H. Hoos, Keith Hamel, and Jürgen Kilian without whom \texttt{GUIDO} would not be around.

**References**


