LA LIBRAIRIE GUIDO
UNE BOITE À OUTILS POUR LE RENDU DE PARTITIONS MUSICALES.

C. Daudin, D. Fober, S. Letz, Y. Orlarey
Grame
Centre national de création musicale
Lyon, France
{daudin, fober, letz, orlarey}@grame.fr

RÉSUMÉ

Le format GUIDO *Music Notation* (GMN) est un langage textuel et lisible de représentation de partitions musicales. Basée sur ce format, la GUIDOLib est une librairie multi-plateformes offrant une API de rendu graphique de partitions musicales. Cet article présente le format GMN et le moteur de rendu GUIDO. Un exemple d'application, le GUIDO *Scene Composer*, est ensuite décrit.

1. INTRODUCTION

Les logiciels d'édition de partitions musicales existent dans le commerce depuis plus d'une décennie, et fournissent des solutions de gravure sophistiquées, mais complexes. A côté de ces systèmes *fermés*, l'approche "boite à outils" a très tôt été explorée [1]. Cependant, peu d'entre eux sont arrivés à maturité : *Common Music Notation* [2] peut être considéré comme le plus abouti ; plus récemment, l'*Expressive Notation Package* (ENP) [3] a proposé une approche prometteuse ; ces deux systèmes sont basés sur Lisp.

Une autre solution est le design de compilateurs capables de produire des partitions depuis une description textuelle. MusiXTEX en est un : c'est un ensemble de macros TEX de notation musicale. MusiXTEX étant puissant mais complexe, des préprocesseurs comme PMX ou M-Tx ont été developpés pour en faciliter l'utilisation. Plus récemment, Lilypond [4] est un compilateur open-source, et partiellement implémenté par le language Scheme ; il possède des fonctions de mise en page automatique, et son format de représentation musicale est simple et intuitif. Ces deux systèmes peuvent générer des fichiers PostScript, EPS ou PDF.

Basé sur le format GUIDO *Music Notation* le projet GUIDOLib est une librairie C/C++ open source et cross-platforme qui fournit des services de mise en page et de rendu graphique de partitions aux applications clientes. Le moteur de rendu est issu d'un travail réalisé par Kai Renz dans le cadre de sa thèse de doctorat [**?**]. Le format de description de la musique est très proche du format Lylipond. A la différence de l'approche *compilateurs*, la librairie GUIDO permet d'embarquer des capacités de rendu de partition dans une application indépendante et de créer dynamiquement des partitions.

Cet article présente d'abord le format GUIDO *Music Notation* (GMN), puis le moteur de rendu GUIDO et l'API de la GUIDOLib. Un exemple concret d'application combinant la GUIDOLib avec Qt est ensuite décrit.

2. LE FORMAT GUIDO MUSIC NOTATION

Le format GMN [5] [6] a été créé par H. Hoos et K. Hamel il y a plus de dix ans. C'est un langage textuel et lisible de représentation de partitions musicales. Il est basé sur un formalisme simple mais puissant, se concentrant sur les grands concepts musicaux (en opposition aux caractéristiques graphiques). Une caractéristique clé de la GMN est l'idée que des concepts musicaux simples doivent être notés simplement, et que seulement les notions complexes doivent nécessiter des notations complexes.

2.1. Concepts de base

La notation GMN de base recouvre les notes, silences, altérations, voix indépendantes, et les concepts les plus courants de la notation musicale comme les clefs, metriques, armures, articulations, liaisons, etc. Les notes sont représentées par leur nom `(a b c d e f g h)`, une altération optionnelle ('#' et '&' pour dièse et bémol), un numéro d'octave optionnel et une durée optionnelle.
La durée est spécifiée par l'une des formes suivantes :

```
* enum / denom dotting
* enum dotting
/ denom dotting
```

où $enum$ et $denom$ sont des entiers positifs et $dotting$ est soit vide, '.', ou '..', avec la même sémantique que dans la notation musicale. 1 est la valeur par défaut si $enum$ ou $denom$ sont omis. La durée est exprimée en fraction de ronde.

Si omis, les descriptions optionnelles sont celles utilisées pour la note précédente de la séquence.

Les accords sont décrits par des notes entre accolades séparées par des virgules, i.e. {c, e, g}

2.2. Les tags GUIDO

Les tags sont utilisés pour donner des informations musicales supplémentaires, comme les liaisons, clés, armures, etc. Un tag simple a l'une des formes suivantes :

```
\tagname
\tagname<param-list>
```

où `param-list` est une liste de chaînes de caractères ou de nombres, séparés par des virgules. De plus, un tag peut ne s'appliquer qu'à un ensemble de notes (comme les liaisons etc.) ; la forme correspondante est :

```
\tagname(note-series)
\tagname<param-list>(note-series)
```

Le code GMN suivant illustre la concision de la notation ; la figure 1 en représente le rendu par le moteur GUIDO.

```
[ \meter<"4/4"> \key<-2> c d e& f/8 g ]
```

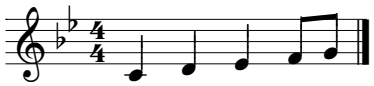

Figure 1. Un exemple simple de code GMN

2.3. Séquences et segments de notes

Une séquence de notes a la forme `[tagged-notes]` où `tagged-notes` est une série de notes, tags, tags temporels séparés par des espaces. Les séquences de notes représentent une seule voix. Les segments de notes représentent plusieurs voix ; ils sont notés par `{seq-list}` où seq-list est une liste de séquences de notes séparées par des virgules, comme dans l'exemple suivant :

```
{
 [\staff<1> \stemsUp \meter<"2/4">
  \beam(g2/32 e/16 c*3/32) c/8
  \beam(a1/16 c2 f)
  \beam(g/32 d/16 h1*3/32) d2/8
  \beam(h1/16 d2 g)],

 [\staff<1>\stemsDown g1/8 e
  g/16 d f a a/8 e a/16 e g h],

 [\staff<2> \stemsUp \meter<"2/4"> a0 f h c1],
 [\staff<2>\stemsDown f0 d g a]
}
```

Le rendu donné par le moteur GUIDO est visible sur la figure 2.

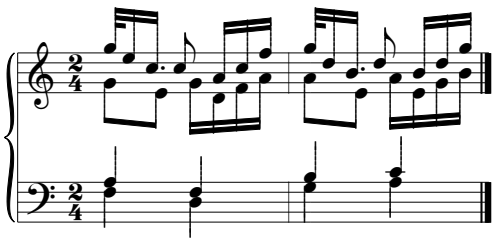

Figure 2. Un exemple avec plusieurs voix

De plus, la spécification *avancée* de GUIDO (non présentée dans cet article) permet un formatage en détail de la partition.

3. LE MOTEUR GUIDO

Basé sur le format GUIDO *Music Notation* et créé initialement par Kai Renz, le moteur GUIDO permet le rendu graphique de partitions musicales, avec mise en page automatique. A l'initiative du GRAME, le moteur a été remodelé en une librairie multi-platformes et est devenu open source en 2002, sous license GNU LGPL et hébergé sur SourceForge.

Le moteur GUIDO utilise une réprésentation mémoire du format GMN : la GUIDO *Abstract Representation* (GAR). Cette représentation est transformée pas à pas pour produire les pages de la partition. Deux types de traitements sont d'abord appliqués à la GAR :

- des transformations de GAR vers GAR, qui représentent une transformation du rendu logique : certaines parties du rendu (comme les barres liant les croches) peuvent aussi bien être produites d'après la GAR qu'exprimées dans la GAR,
- la GAR est convertie en GUIDO *Semantic Normal Form* (GSNF). La GSNF est une forme canonique telle que des expressions sémantiquement équivalentes ont la même GSNF.

Cette GSNF est finalement convertie en GUIDO Graphic Representation (GGR), qui contient les informations nécessaires au rendu, et est directement utilisée lors de celui-ci. Cette dernière étape inclut notamment les algorithmes d'espacement de notes et de séparation des pages [7].

Notez que bien que le format GMN permette un formatage graphique précis (grâce à la GMN *avancée*, dont la figure 3 montre les possibilités), le moteur GUIDO peut effectuer une mise en page automatique perfectionnée.

4. LA GUIDO LIB

4.1. Rendu de partition

La librairie permet de lire des fichiers GMN et de créer les GAR et GGR correspondantes. Les GAR et GGR sont référencées par des pointeurs opaques, et sont utilisées par toute fonction opérant sur la partition. Par exemple : `GuidoParse (const char * filename)` renvoit la GGR construite à partir du fichier GMN, GGR qui pourra par la suite être utilisée pour dessiner la partition avec la fonction `GuidoOnDraw`.

Un exemple de code pour dessiner une partition à partir d'un fichier GMN est donné ci-dessous (voir la section 4.6 pour des informations sur le `VGDevice`) :

```
void DrawGMNFile (char* filename, VGDevice* device)
{
  // Structure pour l'initialisation du moteur
  // avec les polices "guido2" et "times"
  GuidoInitDesc gd = { device, 0, "guido2", "times" };
  // Initialisation du moteur
  GuidoInit (&gd);

  // Déclaration d'une structure de données de dessin
  GuidoOnDrawDesc desc;
  // Obtention de la GGR du fichier GMN,
  // GGR stockée dans la structure dessin
  desc.handle = GuidoParse (filename);

  // Parametrage du dessin
  // (voir la doc pour plus de détails)
  desc.hdc = device;  // le device de sortie
  desc.page = 1;      // index de la page à dessiner
  desc.updateRegion.erase = true;
  desc.scrollx = desc.scrolly = 0;
  desc.zoom = 1;
  desc.sizex = desc.sizey = 0;
```

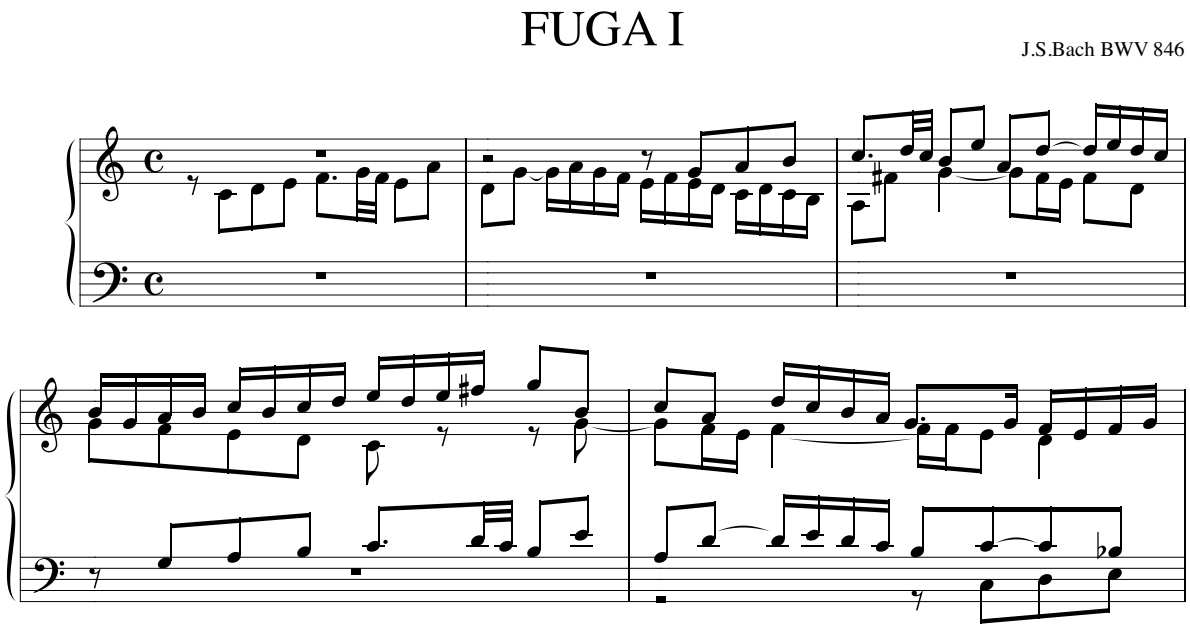

Figure 3. Une partition complexe

```
  // Dessin de la partition
  GuidoOnDraw (&desc);
}
```

4.2. Accès aux pages

Le résultat du rendu est un ensemble de pages qui peuvent être redimensionnées dynamiquement selon le contexte de l'application ou des besoins de l'utilisateurs. La librairie fournit le nécessaire pour changer la taille de la page, obtenir le nombre de pages, ou l'index de page correspondant à une date musicale donnée. Notez que la fonction `GuidoOnDraw` ne dessine qu'une page à la fois.

4.3. Paramètres du moteur

Les algorithmes de mise en page sont contrôlés par des paramètres généraux du moteur GUIDO. La librairie fournit une API pour obtenir et modifier ces paramètres ; ils comprennent la force des ressorts d'espacement, la distance inter-systèmes, la répartition des systèmes et le remplissage optimal de page.

4.4. La GUIDO Factory

Le moteur GUIDO peut aussi être alimenté avec de la musique générée par la GUIDO Factory. L'API de la GUIDO Factory fournit un ensemble de fonctions pour créer une GAR à la volée et pour la convertir en GGR. La GUIDO Factory est une machine à état qui opère sur des éléments courants implicites : par exemple, une fois une voix ouverte (`GuidoFactoryOpenVoice()`), elle devient la voix courante, et dès lors, tous les évènements créés y sont implicitement ajoutés.

L'état de la GUIDO Factory inclut les partitions, voix, accords, notes (ou silences) et tags courants. Certains éléments de l'état de la factory reflète la spécification formelle de GUIDO ; sauf spécifié, les notes créées auront implicitement les durées et octaves courantes.

La construction dynamique de partition musicale est très proche de la description textuelle GUIDO : chaque GAR de la factory correspond à un unique élement de notation. Une fois la partition dynamiquement construite, un appel à `GuidoFactoryCloseMusic()` retourne un pointeur de GAR, directement utilisable avec `GuidoFactoryMakeGR()`, qui retourne elle un pointeur de GGR, lui-même directement utilisable avec le reste de l'API. Le rendu logique est effectué au moment de retourner la GAR, et le rendu graphique au moment de retourner la GGR.

4.5. Mappings graphiques

En parallèle à la GGR, le moteur GUIDO maintient un arbre d'éléments graphiques pour chaque page de la partition, comme illustré par la figure 4. Chaque élément définit un rectangle englobant et une date musicale. Les positions sont stockées en pixels ; l'origine (0,0) est en haut à gauche.

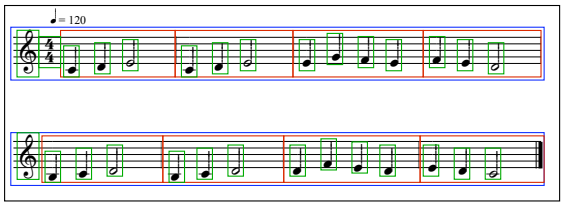

Figure 4. *Bounding boxes* des éléments de la partition

La librairie GUIDO fournit une API spécifique pour obtenir le mapping graphique de la partition et rechercher des éléments par type, position et date.

4.6. Le système graphique virtuel

L'objectif du *système graphique virtuel* est de fournir une couche d'abstraction pour les dépendences graphiques de chaque plateforme. Il est constitué d'un ensemble de classes abstraites recouvrant les besoins de bases d'une application : écrire du texte, dessiner sur l'écran ou en mémoire, etc. Ces classes sont :

- une classe `VGDevice` : spécialisée dans le dessin sur et hors écran ;
- une classe `VGFont` : abstraction de la police ;
- une classe `VGSystem` : pour créer les `VGDevice` et `VGFont` spécifiques à la plateforme.

Ces classes sont implémentées pour différentes plateformes : GDI (Windows), Quartz (Mac OS X), GTK (GNU Linux), OpenGL et plus récemment Qt.

5. SUPPORT DE QT

Qt est un framework de développement multi-plateformes [8], largement répandu pour le développement d'applications avec GUI [1] .

5.1. Classes GuidoQt

Un ensemble de classes permet une utilisation simple la librairie GUIDO avec Qt ; elles sont organisées en 3 couches (figure 5) :

- *basse* : GDeviceQt, GFontQt & GSystemQt : implémentation pour Qt du Virtual Graphic System ;
- *moyenne* : QGuidoPainter : une classe qui utilise les GDeviceQt, GFontQt & GSystemQt, et offre une interface de plus haut niveau ;
- *haute* : des objets Qt "prêt à l'emploi", qui utilisent le QGuidoPainter pour dessiner des partitions GUIDO :
  - QGuidoWidget, un QWidget ;
  - QGuidoSPageItem & QGuidoMPageItem, deux QGraphicsItem qui affichent, respectivement, une page à la fois, ou toutes les pages de la partition ;
  - Guido2Image, pour exporter des partitions GUIDO vers différents formats d'image.

Voici un programme simple utilisant la classe QGuidoPainter pour exporter une partition en image :

```
#include <QApplication>
#include <QPainter>
#include <QImage>
#include "QGuidoPainter.h"
int main(int argc , char* argv[])
{
   QApplication app( argc , argv );

   //Initialisation du moteur Guido
   QGuidoPainter::startGuidoEngine();

   //Création d'un QGuidoPainter...
   QGuidoPainter* p = QGuidoPainter::create();
   //...auquel on donne du code GMN
   p->setGMNCode( "[c/8 d e g e d c]" );

   //Obtention de la taille de la 1ère page de la partition
   int pageIndex = 1;
   QSizeF s = p->pageSizeMM( pageIndex );

   //Création d'une image vide, de la taille de la partition
   QImage image(s.toSize()*10 , QImage::Format_ARGB32);
   image.fill( QColor(Qt::white).rgb() );

   //Le QGuidoPainter dessine la partition, via QPainter.
   QPainter painter( &image );
   p->draw( &painter , pageIndex , image.rect() );

   QGuidoPainter::destroyGuidoPainter( p );

   //Arrêt du moteur Guido
   QGuidoPainter::stopGuidoEngine();

   image.save( "myScore.png" );
   return 0;
}
```

Et en voici un autre affichant une partition avec la classe QGuidoWidget :

```
#include <QApplication>
#include "QGuidoWidget.h"
#include "QGuidoPainter.h"
int main(int argc, char *argv[])
{
   QApplication app(argc, argv);

   //Initialisation du moteur Guido
   QGuidoPainter::startGuidoEngine();

   //Création d'un QGuidoWidget...
   QGuidoWidget w;
   //...auquel on donne du code GMN.
   w.setGMNCode( "[e d c]" );
   w.show();

   // C'est tout !
   int result = app.exec();

   //Arrêt du moteur Guido
   QGuidoPainter::stopGuidoEngine();
   return result;
}
```

5.2. Le GUIDO Scene Composer

Le GUIDO *Scene Composer* est un IDE graphique pour GUIDO, permettant de manipuler des partitions GUIDO dans une scène graphique.

5.2.1. Un outil d'apprentissage du format de notation GUIDO

Le GUIDO *Scene Composer* est un environnement efficace pour apprendre la GMN :

- en tapant du code GMN, la partition correspondante est immédiatement mise à jour, facilitant les essais de différentes notations ;
- une palette d'aide (figure 6) liste un large panel d'expressions et de tags GUIDO, permettant de se passer de manuel externe ;
- l'éditeur de code GMN possède une coloration syntaxique, rendant la lecture du code GMN plus facile ;
- import de partitions MusicXML et MuseData (support limité) ;
- export des partitions en PDF ou image.

1 . see : http ://www.qtsoftware.com

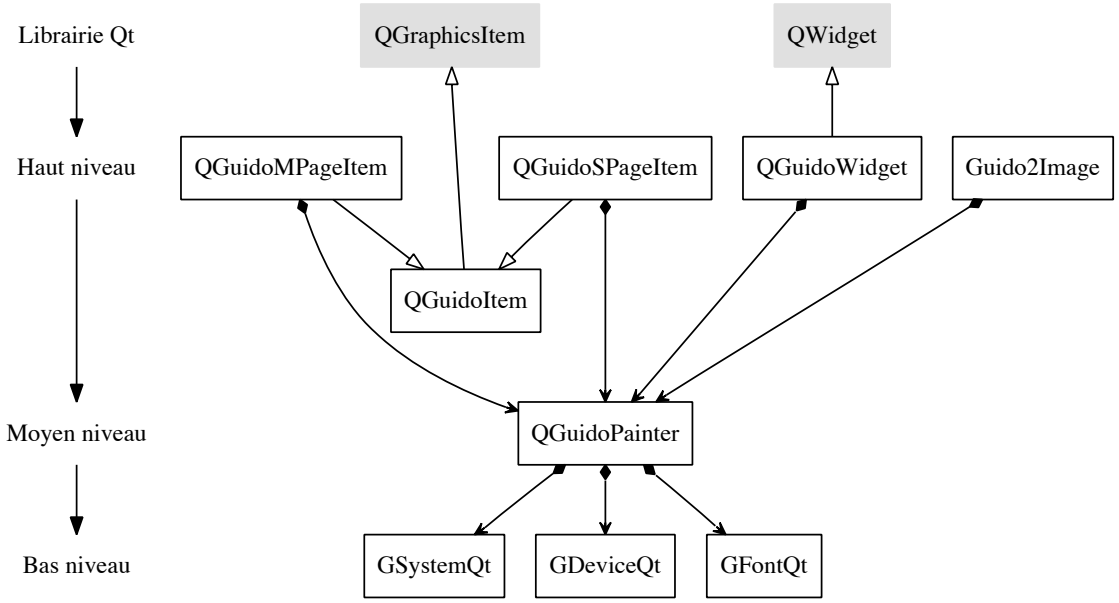

Figure 5. Diagramme des classes GuidoQt

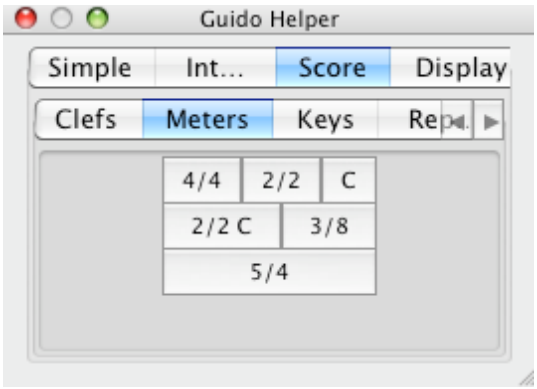

Figure 6. La palette d'aide GUIDO

5.2.2. Un gestionnaire de scènes graphiques

Le GUIDO *Scene Composer* permet de composer des scène graphiques comportant plusieurs partitions :

- bougez, redimensionnez, copiez/collez les partitions, pour créer une scène graphique musicale complexe,
- importez des images (de différents formats) pour enrichir la scène,
- ajoutez des annotations textuelles,
- exportez la scène en PDF ou en image.

6. CONCLUSIONS

Basée sur le format GUIDO *Music Notation*, la GUIDOLib est une librairie multi-plateformes unique en son genre, permettant d'embarquer des services de rendu de partition dans une application. Son récent support de la très populaire librairie Qt la rend simple d'accès à un grand nombre de développeurs.

Dans le futur, nous souhaiterions étendre les possibilités graphiques du GUIDO *Scene Composer*, en y ajoutant d'autres outils de dessin ; il deviendrait alors un outil intéressant pour l'écriture de certains types de partitions, notamment en musique contemporaine. Le support d'autres formats de partitions en import/export fait aussi partie de nos objectifs.

Pour finir, nous pensons ajouter des outils pour assembler ou décomposer les partitions, par exemple les mettre en séquence ou en parallèle, supprimer le début ou la fin, supprimer des voix, transposer, contraindre en durée, et ce en utilisant une approche homogène, où à la fois les arguments et le résultat d'une opération sont des partitions.

Le projet GUIDOLib est disponible sur Sourceforge à l'adresse `http ://guidolib.sourceforge.net`.

7. REFERENCES

[1] Assayag G. and D. Timis. A ToolBox for Music Notation. In *Proceedings of the International Computer Music Conference*, pages 173–178, 1986.

[2] Bill Schottstaedt. Common Music Notation. In Selfridge-Field E., editor, *Beyond MIDI, The handbook of Musical Codes.*, pages 217–222. MIT Press, 1997.

[3] Mika Kuuskankare and Michael Laurson. ENP - Music Notation Library based on Common Lisp and CLOS. In *Proceedings of the International Computer Music Conference*, pages 131–134. ICMA, 2001.

[4] Han-Wen Nienhuys and Jan Nieuwenhuizen. LilyPond, a system for automated music engraving. In

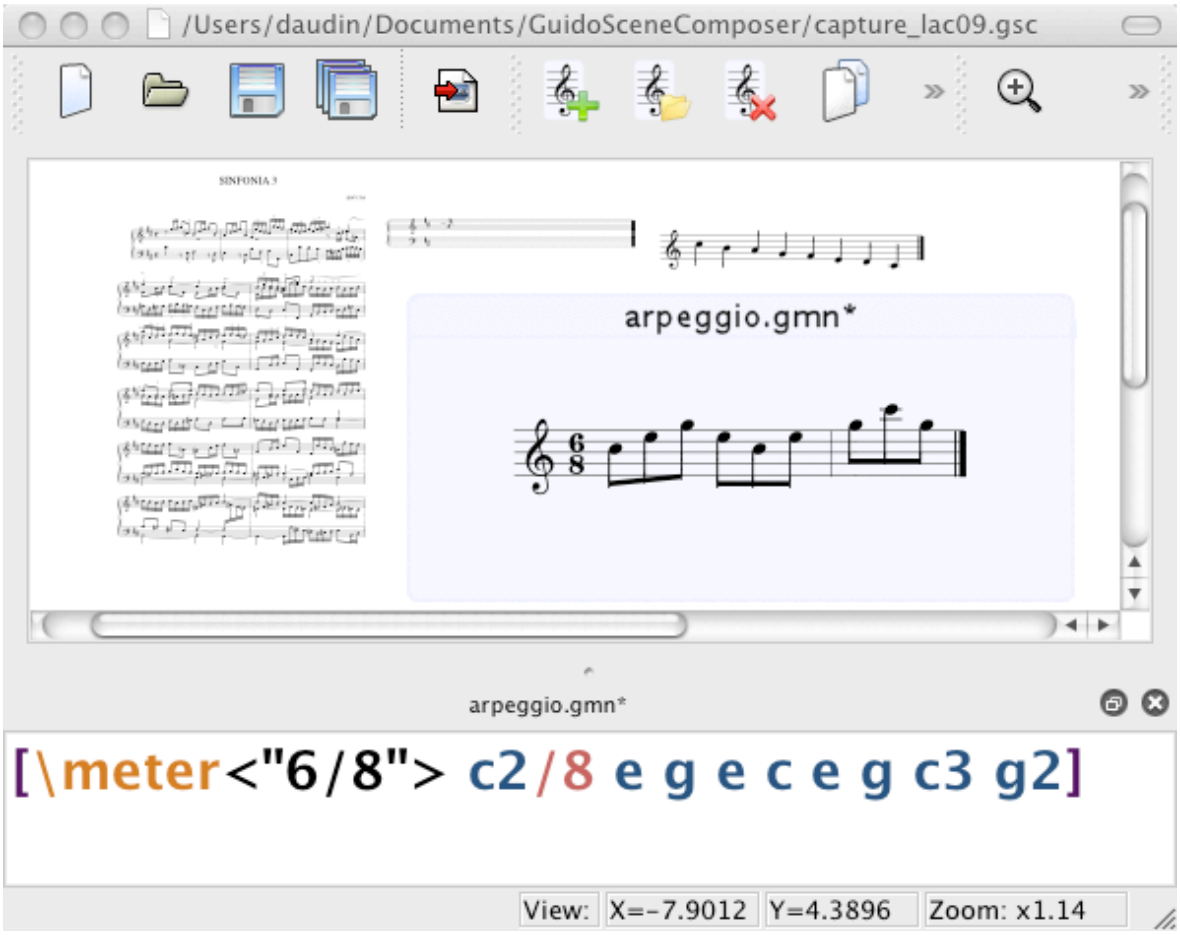

Figure 7. Le GUIDO Scene Composer

Proceedings of the XIV Colloquium on Musical Informatics (XIV CIM 2003), May 2003.

[5] Hoos H., Hamel K. A., Renz K., and Kilian J. The GUIDO Music Notation Format - a Novel Approach for Adequately Representing Score-level Music. In *Proceedings of the International Computer Music Conference*, pages 451–454. ICMA, 1998.

[6] H. H. Hoos and K. A. Hamel. The GUIDO Music Notation Format Specification - version 1.0, part 1 : Basic GUIDO. Technical report TI 20/97, Technische Universitat Darmstadt, 1997.

[7] Kai Renz. *Algorithms and Data Structures for a Music Notation System based on GUIDO Music Notation*. PhD thesis, Technischen Universität Darmstadt, 2002.

[8] Jasmin Blanchette and Mark Summerfield. *C++ GUI Programming with Qt 4 (2nd Edition)*. Prentice Hall, 2008.