
ENSEIRB-MATMECA

Filière informatique, $2^{eme}$ année

---

# Rapport
# Projet de Fin d'Année : Braille Music Compiler

---

*Auteurs :*
Pierrick Baudet
Pierre-Yves Canneill
David Dallago
Lun Jiang
Benjamin Lux
Sara Rahhali
Ali Slimani Houti

*Encadrant :*
Vinh-Thong Ta

5 avril 2012

# Table des matières

# Introduction

## Contexte

Braille Music Compiler (BMC) est un outil de conversion du braille musical principalement destiné à l'usage de personnes aveugles ou malvoyantes. Il est capable d'interpréter une partition écrite en braille pour la convertir en une sortie audio qui peut par la suite être transformée en un fichier MIDI ou MusicXML.

BMC est actuellement à l'état de prototype, il est en cours de développement par Mario Lang, un autrichien non-voyant passionné par la musique et l'informatique.

L'idée de développer un tel outil lui est venue lorsqu'il voulu jouer des partitions braille de musique classique. Souvent, il n'était pas sûr de bien interpréter certaine notes et s'est vite rendu compte que cette notation peut parfois porter à confusion. La notation Braille musicale est, en effet, assez complexe et peut parfois sembler ambiguë d'où l'utilité d'un programme qui servirait de référence en permettant d'écouter la note jouée.

Seulement la plupart des programmes disponibles sur le marché qui vont dans ce sens sont assez chers - coût moyen de 600 euros - et ne sont en général compatibles qu'avec Windows, ce qui limite fortement leur utilisation.

M. Lang commença par travailler sur *FreeDots*, un programme capable de convertir des partitions *noires* de musique en partitions braille.

Encouragé par le succès que rencontra ce premier programme, Il décida ensuite de se lancer dans la transcription inverse du braille musical, ce qui donna naissance au Braille Music Compiler.

L'amélioration de l'utilisation du BMC notamment par implémentation d'une interface graphique nous a alors été proposée par Samuel Thibault, chercheur à l'INRIA et proche collaborateur du développeur du BMC.

## Problématique

Le Braille Music Compiler est, pour l'instant, toujours en cours de développement. En effet, l'exécution de sa version actuelle requiert une certaine connaissance des outils informatiques, chose qui rend difficile son utilisation par un usager lambda.

L'intérêt de ce projet est donc d'améliorer cet outil de façon à ce qu'il devienne accessible à un plus large public. La mise en oeuvre d'une interface graphique permettrait en effet de rendre son utilisation plus intuitive.

Étant principalement dédié à l'usage de personnes non(ou mal)-voyantes, l'interface graphique à implémenter doit donc répondre à un certain nombre de critères en termes d'accessibilité, elle doit être simple mais fonctionnelle. De plus, il serait intéressant que cette interface comporte aussi des fonctionnalités d'édition et de modifications de fichiers écrits en braille musical.

Le BMC a aussi pour ambition de faciliter la collaboration entre voyants et non-voyants, en ce sens, l'interface devrait pouvoir à la fois afficher les partitions de musique écrites en braille et en *noir* permettant ainsi à chacun des collaborateurs de pouvoir se repérer par rapport aux partitions de l'autre.

# 1 Organisation du travail

## 1.1 Méthode Scrum

Le responsable pédagogique de ce pfa, monsieur Ta, a préféré au classique cahier des charges une utilisation des méthodes agiles, plus précisément de la méthode *scrum*. Cette méthode semble être efficace puisque aujourd'hui, nous avons bien avancé le projet, et là où d'autres groupes n'ont fait qu'un cahier des charges, nous avons un début d'application. Ce travail concret nous motive d'autant plus à nous investir dans le projet. De plus, cette méthode prépare le groupe à l'entreprise puisque de nos jour, la plupart utilisent cette méthode.

### Le Product Backlog

La méthode *scrum* place tous les collaborateurs du projet sur un pied d'égalité. Il faut tout d'abord créer un Product Backlog qui correspond à toutes les tâches à effectuer pour la finalisation du projet. Ce document doit être validé avec le client, puis ce dernier note les tâches par ordre de priorité.

### Un Sprint Backlog

Ces dernières sont ensuite réparties en Sprint Backlog. Ce sont des périodes courtes durant lesquelles l'équipe travaille sur certaines tâches définies précédement. À la fin de chaque Sprint Backlog l'équipe présente sont travail sous forme concrète au client, sous forme d'un executable, de maquettes ou toute autre forme. Ceci est fait pour éviter au projet de partir dans une mauvaise direction et permet au client, dont les envies peuvent changer, de prendre part activement au projet. La fin d'un sprint laisse place à une nouvelle réunion où sont définis les objectifs du sprint suivant. Nous avons décidé de faire des Sprints d'une durée approximative de 3 semaines.

### Les moyens de communication

Afin de pouvoir travailler de façon efficace en groupe, il est nécessaire de disposer d'outils spécialisés dans la gestion de projet. Nous avons mis en place trois de ces outils. Pour faciliter la communication au sein du groupe nous avons créé la liste *pfa.bmc@listes.enseirb-matmeca.fr*. Cette liste est utilisable par tout le monde (membres du groupe, client, résponsable pédagogique) et envoie le mail aux seuls membres du groupe. Pour gérer la méthode *scrum* nous communiquons via *Google docs* où nous avons créé une série de fichiers. Les principaux étant le Product Backlog, le détail des Sprint Backlog, le compte rendu des réunions avec l'encadrant et un ficher servant de tableau blanc. Enfin nous avons mis en place un répertoire de travail sur *Github* accessible en écriture aux membres du projet et visible par tous. Celui ci est accessible à l'adresse *https ://github.com/ddallago/pfa.bmc*.

# 2 Spécification des besoins

Le braille Music Compiler est d'abord conçu pour pouvoir convertir en sortie audio un fichier existant écrit en braille musical ainsi que pour permettre à son utilisateur d'apporter des modifications à un tel fichier ou d'en éditer un nouveau. Le BMC a, de plus, pour objectif de faciliter le travail commun de personnes voyantes et non-voyantes. Le travail effectuer par l'équipe prend en compte ces deux aspects.

Les besoins du client nous ont été communiqués dans le document *README.txt* sections *User Interface(s)* et *TODO*.

Après avoir rencontré M. Thibault, notre client local, et avoir discuté avec notre encadrant pédagogique M. Ta, nous avons pu déterminer les tâches que nous sommes en mesure de réaliser.

Les sections qui suivent résument les actions que le délivrable devrait pouvoir accomplir.

## 2.1 Besoins fonctionnels

Le développeur du BMC souhaite que cet outil soit accessible au plus large public possible. Dans ce sens, le système d'exploitation sur lequel travaille l'utilisateur ne devrait pas être un obstacle à l'utilisation du Braille Music Compiler.

Nous décidons, dans un premier temps, de mettre en place une interface graphique sous Linux qui sera ensuite portée sous Windows.

Quelle que soit le système d'exploitation, la version finale de l'interface graphique devra pouvoir réaliser les actions suivantes, classées par ordre de priorité :

- **Ouvrir un fichier :** affichage sur le programme principal du contenu d'un fichier écrit en braille musical sélectionné via une fenêtre de sélection.

- **Enregistrer un fichier :** enregistrement sur le disque dur du fichier ouvert en écrasant sa copie existante (se trouvant au même emplacement).

- **Fermer l'application :** fermeture du fichier ouvert en demandant la confirmation de sauvegarde des modifications éventuelles.

- **Éditer un fichier braille :** ce point regroupe toutes les fonctionnalités habituellement proposées par un éditeur de texte comme les actions de copier, coller, couper un partie du texte ainsi que les actions de sélection de lignes et de paragraphes.

- **Écouter une sélection :** lecture des notes sélectionnées.

- **Colorer le texte braille :** coloration du texte afin de faciliter à la personne voyante (collaborant avec un non-voyant) de se repérer par rapport aux notes écrites en braille musical.

- **Afficher la note en cours :** déplacement du curseur au fur et à mesure que les notes sont jouées pour que l'utilisateur puisse savoir à quel son correspond quelle note.

- **Numéroter les mesures :** autre fonctionnalité pour faciliter la collaboration entre voyant et non-voyant, la numérotation des mesures permet à chacun de se repérer par rapport aux notes de l'autre.

En plus des besoins que doit satisfaire l'interface graphique, il faut aussi implémenter des modules d'export qui convertissent le langage abstrait renvoyé par le compilateur en des fichiers MIDI (sortie audio), musicXML ou lilypond. Ce point sera détaillé dans la section *BMC* .

## 2.2 Besoins non-fonctionnels

En surcroît des besoins fonctionnels auxquels doit répondre le projet, ce dernier doit satisfaire les point suivants :

- **Portabilité :** le code doit être écrit de façon à pouvoir en réutiliser la plus grande portion possible lors du portage sur d'autres systèmes d'exploitation, notamment lors du portage

sous windows.

- **L'extensibilité :** le délivrable est susceptible d'être modifié par M. Lang, ce critère doit donc être pris en compte afin de résister aux changements éventuels .

- **Autonomie :** l'utilisation du travail rendu, notamment l'interface graphique ne devrait pas nécessiter la mise en place d'installations lourdes, sa mise en oeuvre doit être facile et la plus intuitive posssible.

- **Modularité :** le code doit être réparti en modules, afin de garantir la possibilité de réutiliser le code par le client ainsi qu'une meilleure gestion des erreurs. De plus, l'injection de nouvelles classes ou méthodes dans le projet doit être le plus possible indépendante du code du client.

- **Documentation :** le code doit être documenté en anglais pour faciliter sa compréhension par le client.

# 3 Travail effectué

## 3.1 BMC

*Braille music compiler* est un programme qui se veut de transformer un fichier écrit en *Braille Music* en un autre langage utilisable par d'autres applications. La difficulté est que le langage *Braille Music* possède beaucoup d'ambiguïtés, en effet par exemple pour connaître la note exacte en cours, il est nécessaire de savoir quelles ont été les notes précédentes. À l'heure actuelle le programme arrive correctement à compiler le langage *Braille Music* en un langage Abstrait qui lui ne possède aucune ambiguïté. Ce langage abstrait peut être converti en n'importe quels formats de musique. Par exemple :

- MIDI (Format AUDIO)
- MusicXML
- lilypond (Equivalent au Latex, mais pour la musique)

Le schéma 1 résume le fonctionement de BMC.

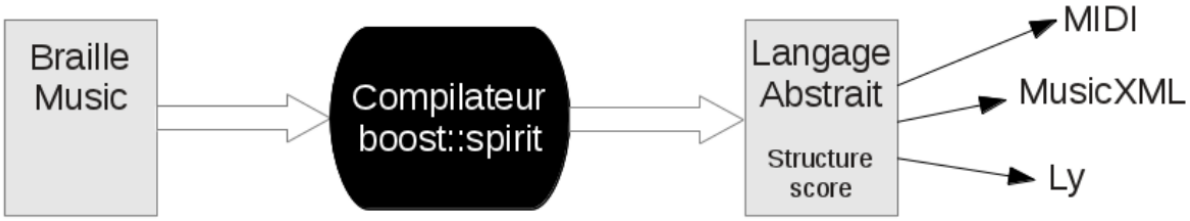


Figure 1 – Schéma visuel du fonctionement de BMC

Notre travail est donc d'implémenter ces modules de conversion. Pour cela il est nécessaire de comprendre comment est conçu le langage Abstrait. Après la compilation du *Braille Music*,

l'ensemble des données sont stockées dans une structure nommée "score". Cette structure est une multitude de vecteurs imbriqués les uns dans les autres. Voici le code de la strucure suivi d'une représentation visuel (2) :

```
typedef boost::variant<note, rest, chord, value_distinction, hand_sign, simile, barline>;
typedef std::vector<sign> partial_voice;
typedef std::vector<partial_voice> partial_measure;
typedef std::vector<partial_measure> voice;
struct measure : locatable
{
  boost::optional<unsigned> ending;
  std::vector<voice> voices;
};
typedef std::vector< boost::variant<measure> > staff;
typedef std::vector<staff> part;
struct score {
  boost::optional<time_signature> time_sig;
  std::vector<part> parts;
};
```

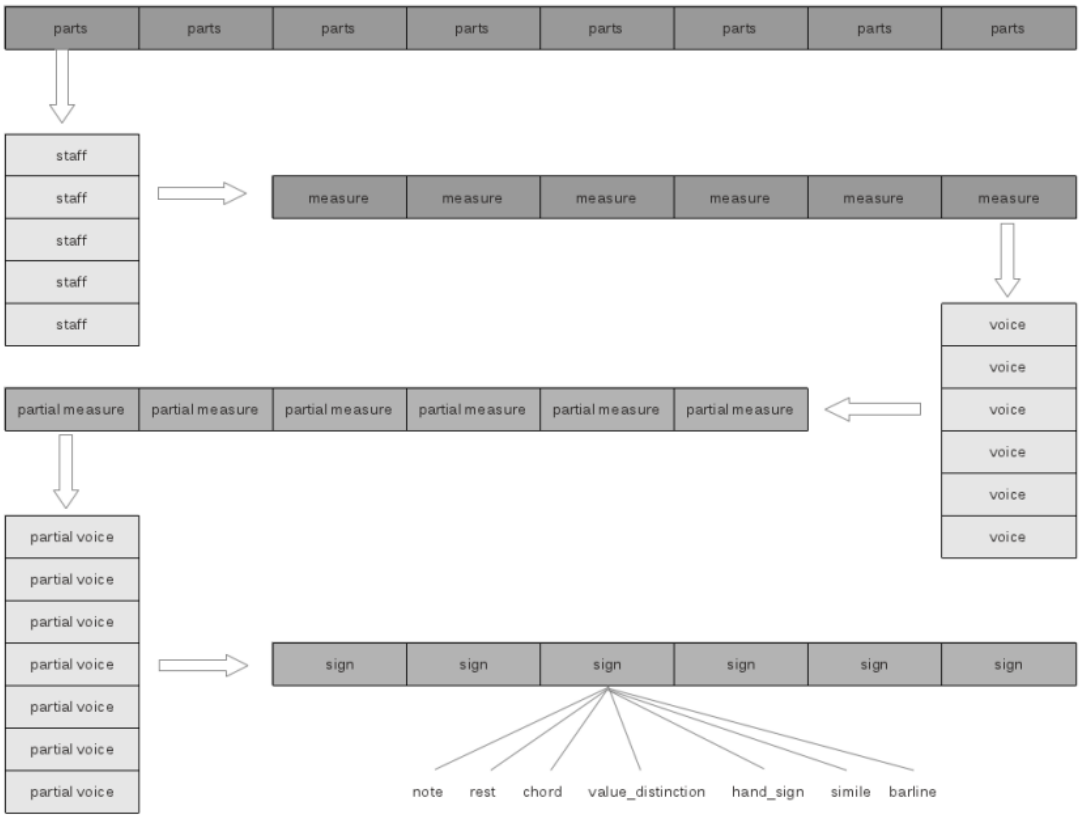


Figure 2 – Représentation de la structure score

Il est donc maintenant plus facile d'expliquer la structure *score*. Un fichier de musique braille peut donc se diviser en plusieurs parties (parts). Chaque partie possède un ou plusieurs

"staff". Un "staff" n'est autre que la portée, typiquement si le compositeur écrit une partition de musique pour piano il y aura deux portées. Chaque portée possède un nombre de mesures. Si la musique composée est polyphonique il y aura plusieurs voix ("voices"). Une voix est ensuite divisée en mesures partielles, ces mesures partielles pouvant de nouveau être divisées en voix ("partial_voices") qui elles sont divisées en un vecteur de *sign*. Un sign pouvant être une note, un repos, une barre ... La figure 3 représente sur une partition de musique ses différents termes.

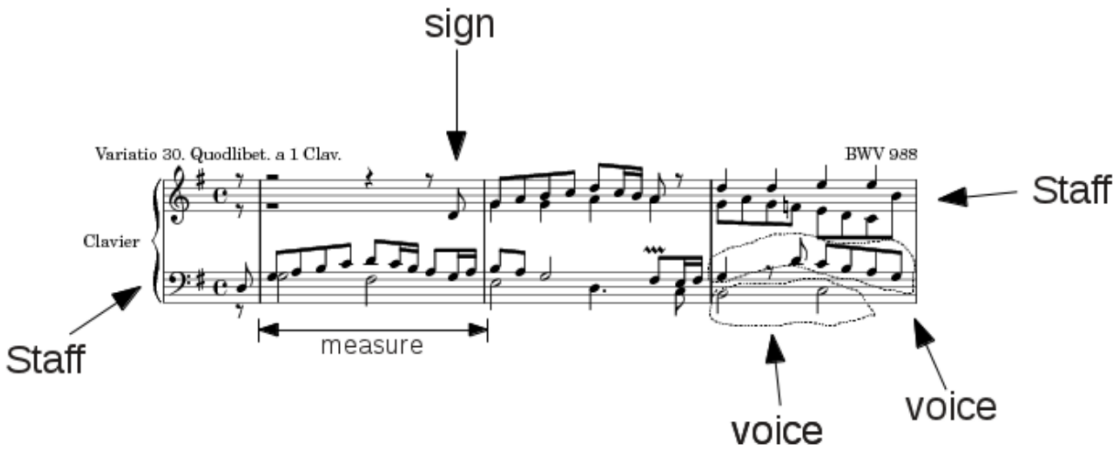


Figure 3 – Un petit exemple visuel

Avant de se mettre à faire la conversion du langage abstrait vers un format de musique conventionel nous avons d'abord créé les classes de base que nous devrons utiliser pour effectuer les différentes conversions. On procède dans un premier temps à un affichage sur le terminal.

```
./bmc < input/bwv988-v30.bmc
```

```
 r1/8  r1/8 >  |   r1/2  r1/4  r1/8 D 5 >   r1/1 >  |  G 5 A 5 B 5 C 6 D
6 C 6 B 5 A 5  r1/8 >  G 5 G 5 A 5 A 5 >  |  D 6 D 6 E 6 E 6 >  G 5 A 5 G
5 F 5 Natural  E 5 D 5 C 5 B 5 >  |  B 5 A 5 G 5 G 5 G 5 D 5 >  |   r1/2
 r1/4  r1/8 D 6 >  G 5 A 5 B 5 C 6 D 6 C 6 B 5 A 5 D 6 >  G 5 G 5 A 5 A 5 >
```

## 3.2 Interface graphique

L'un des critères essentiels de ce projet est l'accessibilité. Par conséquent, nous avons choisi d'utiliser la bibliothèque GTK+ pour l'interface graphique sur plateforme linux. En effet, cette bibliothèque est compatible avec les lecteurs d'écran des systèmes linux.

### Apparence

L'interface graphique est très classique : elle présente une barre de menu, une barre de raccourcis ainsi que les raccourcis naturels, encore en cours d'implémentation. La fenêtre se divise en deux parties : la première sert pour l'utilisateur malvoyant à éditer les partitions brailles. La seconde est destinée à afficher les partitions classiques avec les notes resultant de la conversion des partitions brailles. La figure 4 présente l'apparence de la fenêtre sous le système d'exploitation Ubuntu

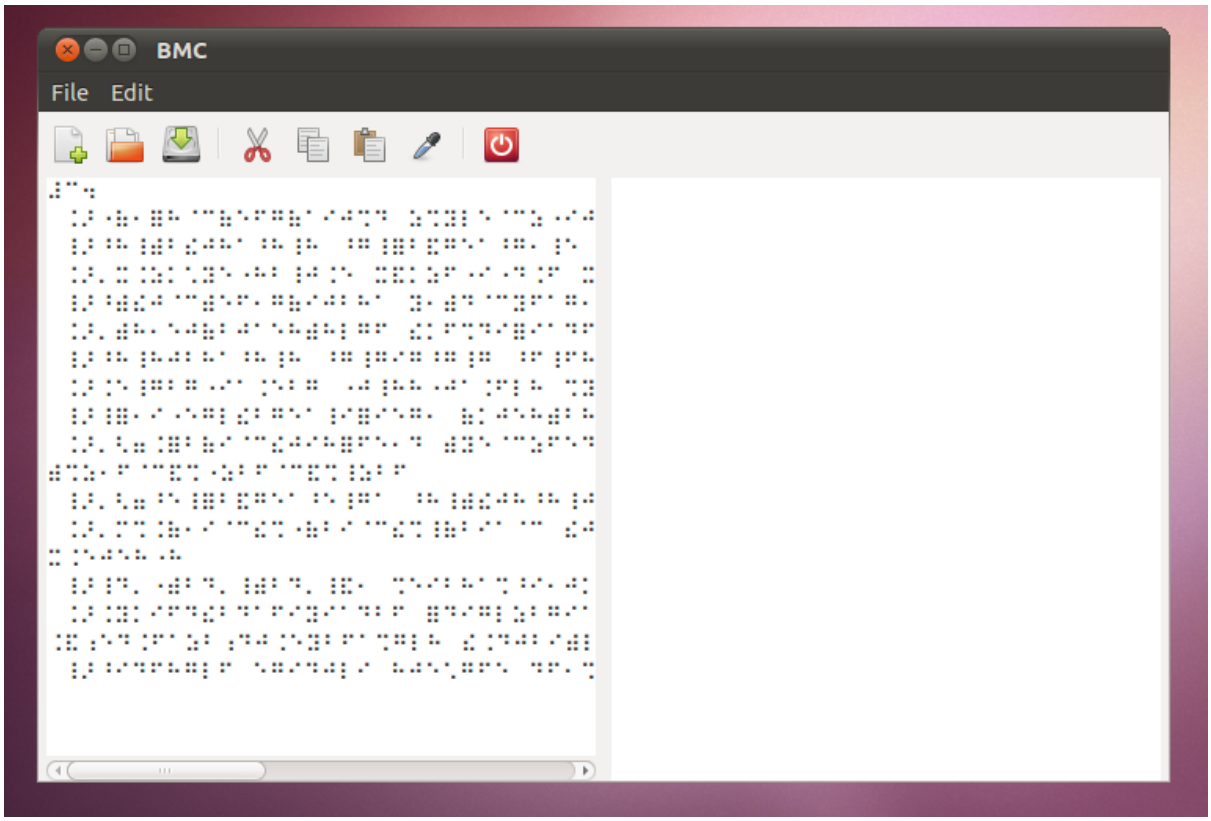


Figure 4 – Apparence de l'interface graphique

### Fonctionnalités

Le prototype réalisé comporte tous les fonctionnalités de base d'un éditeur de texte. Il permet de créer un nouveau fichier, d'ouvrir un fichier, d'enregistrer un fichier et d'enregistrer le fichier courant sous un autre nom. Il permet aussi d'éditer les textes avec les outils classiques comme copier, couper, coller, séléctionner du texte.
Non seulement toutes ces fonctions sont accessible depuis la barre de menu et/ou la barre d'outils, mais elles sont également toutes associées à des raccourci clavier. Les raccourcis utilisés sont très classiques. Par exemple, Ctrl+O pour ouvrir, Ctrl+C pour copier et Ctrl+V pour coller.
Le prototype possède aussi une procédure de sécurité qui consiste à vérifier si le fichier en cours d'édition est sauvegardé ou non et propose à l'utilisateur d'enregistrer son fichier lorsque l'utilisateur tente de créer un nouveau fichier, d'ouvrir un autre fichier ou encore de quitter le programme.

### Structure de l'interface graphique

L'interface graphique réalisée se base sur une structure centrale qui est *BrailleMusicEditor*. Cette structure est de la forme suivante :

```
typedef struct
{
```

```
  GtkWidget *window;
  GtkWidget *vbox;
  GtkWidget *hbox;
  GtkWidget *menubar;
  GtkWidget *toolbar;
  GtkWidget *scrollbar;
  GtkWidget *scrollbar2;
  GtkWidget *textview;
  GtkWidget *rightview;
  gchar *filename;
} BrailleMusicEditor;
```

La fenêtre obtenue est le résultat d'emboîtements des composantes (*GtkWidget*) présentes dans cette structure. La composante principale est le *window*, elle contient le *vbox* qui est un conteneur permettant de stocker une séquences de composantes de manière verticale. Dans le prototype réalisé , ce conteneur contient, dans l'ordre, les composants suivants :

- *menubar* : une barre de menu
- *toolbar* : une barre d'outils
- *hbox* : un conteneur horizontal

Le conteneur horizontal contient les deux barres de défilement, *scrollbar* et *scrollbar2*, qui contienent elles-mêmes chacune une composante de visualisation et d'édition de texte, *textview* et *rightview*. Un schéma illustratif est représenté sur la figure 5.

La liaison entre ces composantes et leurs fonctions s'effectue grâce à la programmation événementielle.

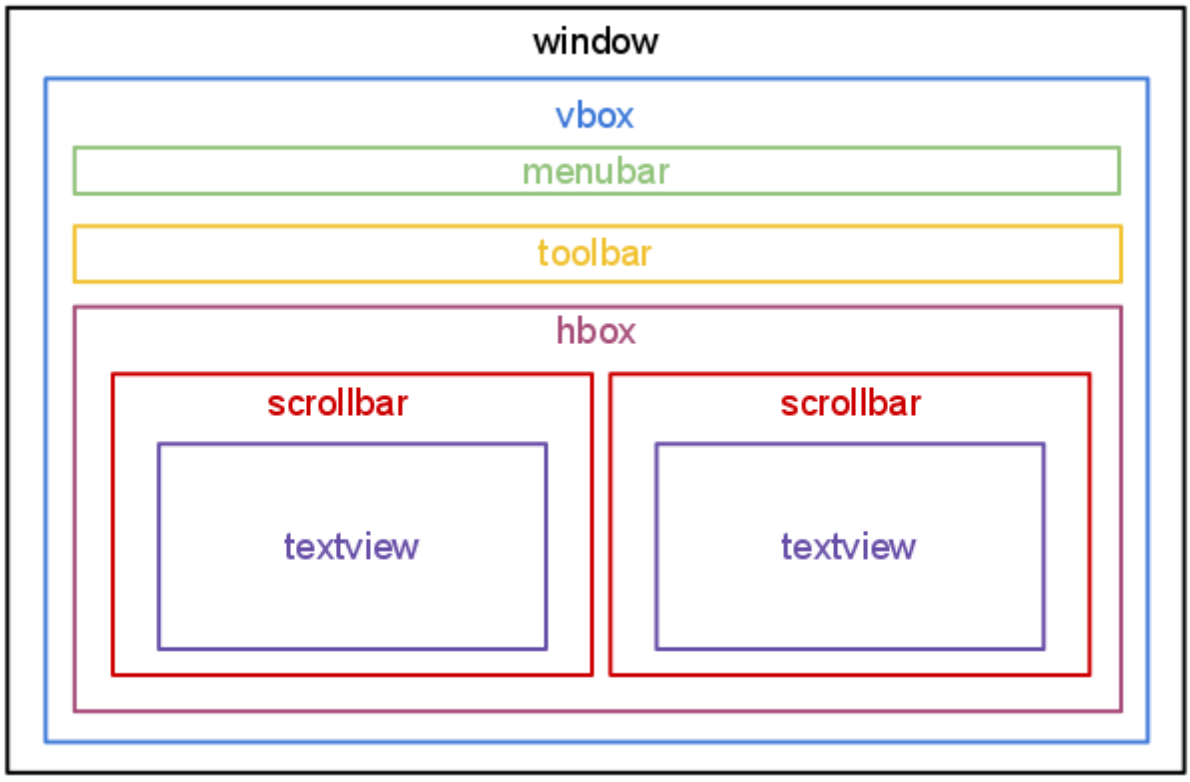


Figure 5 – Structure de l'interface graphique

**Programmation événementielle**

La programmation événementielle est la base de tout programme interactif. Il fonctionne sur le principe *action* → *reaction*. L'exemple immédiat est le clic sur un bouton (*action*) qui déclenche (*réaction*) l'appel à une fonction, dite *CALLBACK*.

GTK+ gère lui-même ces différentes actions (signaux) pour tous les widgets. Ceux-ci captent l'action de l'utilisateur (*e.g* le clic), émettent un signal, GTK+ reçoit le signal et sait alors quelle fonction *CALLBACK* appeller.

La tâche du développeur est grandement simplifiée par l'utilisation des bibliothèques GTK+ : la fonction à utiliser a pour prototype

```
gulong g_signal_connect(gpointer *object, const gchar *name, GCallback func, gpointer func_d
```

où

- *object* est un pointeur vers le widget contenant le bouton (*e.g* $G\_OBJECT(editor- > window)$) ;
- *name* est le nom de l'évènement (*e.g* "*destroy*") ;
- *func* est la fonction *CALLBACK* (*e.g* $G\_CALLBACK(gtk\_main\_quit)$) ;
- *func_data* est un paramètre donné à *func*.

Une autre méthode permet d'associer le clic d'un bouton à une action. Dans le cadre d'une boîte de dialogue proposant plusieurs choix (*e.g* : Voulez-vous sauvegarder avant de quitter ?), la création de ce *dialog* permet au développeur de décrire son fonctionnement. Suite à la création d'un nouveau widget *dialog* contenant les boutons OUI et NON (opération simplifiée par l'utilisation de la variable $GTK\_BUTTONS\_YES\_NO$), il suffit de lancer ce widget et stocker sa valeur de retour dans une variable :

```
gint resp=gtk_dialog_run (GTK_DIALOG (dialog));
```

Il ne reste plus qu'à dissocier :

```
if ( resp == GTK_RESPONSE_YES)
_save();
else if( resp == GTK_RESPONSE_NO)
_quit();
```

## 3.3 Bibliotèque BOOST

Lors de notre analyse du langage abstrait et ce en scrutant de plus près le code source du *Braille Music Compiler* nous étions amenés à étudier la bibliothèque BOOST qui est un ensemble de bibliothèques *C++* gratuites et portables. BOOST est très riche et fournit un large choix de bibliothèques, mais nous ne préciserons que celles utilisées pour le stockage de la musique sous forme de langage abstrait.

**Boost.variant**

Le Boost.variant est une sorte de type *somme*. Il s'agit en fait de décomposer un type donné en plusieurs sous-types. Une instance de ce type donné peut être obtenue par une valeur de ses sous-types, mais pas deux types à la fois, ce qui peut donc être assimilé à une *union*. Cependant, en *C* et *C++* le type *union* ne permet pas de gérer des classes dès qu'elles ont un constructeur ce qui est rendu possible grâce au Boost.Variant en *C++*.

Donc contrairement à un *std : :vector* en *C++* qui offre des éléments *multi-valeur, type unique*, le *boost : :variant* offre des éléments *multi-type, valeur unique*.

**Boost.rational**

Le langage *C++* offre plusieurs possibilités pour stocker des nombres, des entiers naturels aux réels et ce en les approximant par différents types : unsigned int, int, float... La bibliothèque boost.rational permet de représenter un nouveau groupe de nombres : les nombres rationnels.

Au niveau implémentation, le type Boost.rational est constitué de deux nombres de type entier, qui représente le numérateur et le dénominateur. De cette façon, cela permet d'avoir une meilleure précision de calcul et de faciliter l'implémentation.

## Conclusion

Ce premier rapport ainsi que le **Product Backlog** en annexe résultent des différents rendez-vous avec le client local Samuel Thibault ainsi que les multiples réunions que l'équipe a eu avec l'encadrant pédagogique.
Ces deux documents font état de l'avancement du projet à ce jour.

La réalisation du projet étant amenée à avancer, certains points de ce document sont susceptibles d'être modifiés. Cependant, les points fondamentaux de la conception du projet se doivent d'être respectés.

Vous trouverez dans les synthèses personnelles le sentiment de chacun vis à vis du projet.