

Heig-vd

VIRTUOSO

Travail de bachelor



Étudiant : Quentin Salomon
Professeur : François Birling
29/07/2016

1 Table des matières

2	Descriptif	4
3	Clause de confidentialité	6
4	Résumé	6
5	Introduction.....	7
6	Pré-étude.....	7
6.1	Cahier des charges.....	7
6.1.1	Problème	7
6.1.2	Démarche	7
6.1.3	Interface utilisateur	7
6.1.4	Programme de traitement des fichiers midi	10
6.2	Évaluation des risques.....	11
6.2.1	Communication I2C	11
6.2.2	Type de fichier converti pour le Virtuoso : MIDI.....	13
6.2.3	Windows IoT.....	14
6.2.4	Woopsa, HTML & JavaScript.....	16
6.2.5	Concept HMI.....	18
6.3	Comparatif des plateformes.....	19
6.3.1	Problème	19
6.3.2	Démarche	19
6.4	Choix du matériel	20
6.4.1	Problème	20
6.4.2	Démarche	20
6.5	UML	20
6.5.1	Problème	20
6.5.2	Démarche	21
6.5.3	Programme Virtuoso	21
6.5.4	Programme midi→XML	24
7	Projet	26
7.1	Schéma de la communication	26
7.2	Protocole de communication	26
7.2.1	Composition des messages	26
7.2.2	Types de messages	27
7.3	Réglages USB	27
7.3.1	Problème	27

7.3.2	Démarche	27
7.4	Communication USB.....	28
7.4.1	Partie NUC	28
7.4.2	Partie Arduino	29
7.4.3	Remarque	31
7.5	I2C.....	31
7.6	Programme de conversion	32
7.6.1	Problème	32
7.6.2	Démarche	32
7.7	Remote debugging	36
7.7.1	Problème	36
7.7.2	Démarche	36
7.7.3	Solution.....	39
7.8	Programme Virtuoso	39
7.8.1	Squelette des UserControls.....	39
7.8.2	UML final du Virtuoso.....	42
7.8.3	Problème	42
7.8.4	Démarche	42
7.8.5	Solution.....	46
7.8.6	Splash screen et pop-up	46
7.8.7	Défilement des notes de la partition courante	48
7.9	Visuels des interfaces	51
7.9.1	Problème	51
7.9.2	Démarche	51
7.9.3	Prototypes	51
7.9.4	Possibilités offertes par concept	52
7.9.5	Spécificité pour appliquer les styles	53
7.9.6	Styles créés pour le Virtuoso	54
7.9.7	Visuel final	58
7.10	Serveur WEB.....	60
7.10.1	ConceptWeb.....	60
7.10.2	Problème	60
7.10.3	Démarche	60
7.10.4	Rendu final.....	70
8	Gestion de projet.....	70
8.1	Planning	70

8.1.1	Problème	70
8.2	Démarche	70
9	Informations pratiques.....	77
9.1	Chargement de partition sur le NUC	77
9.2	Code pour l'accès au paramètre de l'application.....	77
10	Améliorations possibles.....	77
10.1	Métalophone	77
10.2	Édition des fichiers MIDI.....	77
10.3	Gestion des erreurs	78
11	Conclusion	78
12	Liste des références.....	78
13	Journal de travail	78
14	Annexes	78

2 Descriptif



Département TIN

Filière Génie électrique

Orientation Electronique et Automatisation industrielle

Candidat Salomon Quentin

Responsable Birling François

TRAVAIL DE BACHELOR 2015 - 2016

Virtuoso - Automatisation d'un xylophone

Domaine de recherche : Automatisme, Micro contrôleur, Développement logiciel

Institut iAi

Énoncé

La HEIG-VD a déjà réalisé un robot Xylophoniste anthropomorphe, appelé Xylobot. Du fait de sa structure, il peut jouer à une vitesse proche de l'humain.

Pour pouvoir jouer la musique à une rapidité absolument impressionnante, ce nouveau travail de diplôme a pour but d'automatiser un autre xylophone de 37 touches en l'équipant d'un actionneur électromécanique en face de chaque touche, et en pilotant l'exécution musicale par un logiciel embarqué d'une part, et un logiciel PC d'autre part permettant de préparer et transmettre les partitions.

En complément, la réalisation d'une interface utilisateur permettant de contrôler le système depuis un smartphone et de sélectionner le morceau à jouer peut s'inscrire dans ce travail de Bachelor.

Cahier des charges

Le projet recouvre les activités suivantes :

- Evaluation et dérisquage de la technologie Windows IOT pour l'application visée, sélection d'un système d'exploitation pour l'application finale.
- Analyse de la fonctionnalité et conception de l'architecture logicielle avec UML.
- Développement du logiciel embarqué de commande et d'interface graphique pour la pilotage du Xylophone.
- développement d'un logiciel de préparation des séquences de jeu du xylophone avec capacité d'importation d'un format de fichier musical répandu.
- Développement d'une interface Web de supervision de l'exécution.
- Démonstration finale, optimisation pour améliorer la musicalité.



Bibliographie

Candidat

Salomon Quentin Date: _____ Signature: _____

Responsable

Birling François Date: _____ Signature: _____

Chef du département TIN

Frosio Guido Date: Yverdon-les-Bains, le 28.07.2016 Signature:

3 Clause de confidentialité

4 Résumé

Au début, le Virtuoso n'était qu'un simple xylophone avec ses 37 touches. Il a alors été équipé d'un système électronique ainsi que d'actuateurs pour pouvoir piloter les touches. Il ne manquait plus que le logiciel dont le développement a fait l'objet de ce travail de bachelor. Avec ses objectifs ambitieux, tel que pouvoir jouer plus vite que l'homme tout en gardant une belle musicalité tout en combinant plusieurs technologies, il est devenu un défi pour le diplômant devant le programmeur. Son but final est d'être présenté aux évènements promotionnels de la Heig-vd.

À travers une étude fonctionnelle et technologique approfondie , un chemin pour arriver aux buts de ce travail de bachelor a été imaginé. Le Virtuoso est finalement composé de son électronique de base avec un Arduino la pilotant en I2C. Afin d'offrir une interface homme-machine, un PC embarqué de type Intel-NUC a été implanté. Le logiciel conçu avec UML a été développé tout spécialement dans le cadre de ce travail. Il est hébergé sur le NUC et gère la communication USB avec l'Arduino ainsi qu'une interface graphique attrayante en WPF, adaptée à un écran tactile. L'interface, dans sa symphonie de couleurs, permet de gérer une liste de lecture en ajoutant ou enlevant des partitions. Il est aussi possible de jouer facilement cette liste de lecture avec ses fonctions Play/Pause/Stop/Next, les notes défilant à l'écran au fur et à mesure de la progression. Sur l'interface apparaît également une page d'accueil moderne informant les utilisateurs sur certains points du Virtuoso, par exemple le nom du wifi. Afin d'avoir une bonne musicalité, le Virtuoso dispose d'une section paramètres dans laquelle il est possible de calibrer l'intensité des notes en réglant le temps de frappe. Pour accéder à ces paramètres, un mini-code pour écran tactile doit être tapé pour éviter un malencontreux déréglage venant d'une tierce personne.

Un serveur Web, développé avec un tout nouveau framework, nommé ConceptWeb, venant d'un autre travail de bachelor, a également été intégré à ce riche ensemble de technologies. Permettant à des visiteurs mélomanes d'interagir avec Virtuoso grâce à leurs smartphones, le serveur offre ainsi une interface ludique affichant le titre de la partition jouée et sa progression. En plus de cela il est possible de voir la liste de lecture, et même de demander l'ajout à la liste de lecture d'une partition choisie dans le catalogue.

En complément, un programme de conversion a également été développé, permettant d'obtenir des partitions exploitables par le Virtuoso à partir de n'importe quel fichier midi. Ce programme de conversion facile d'emploi et rapide a permis de générer un riche répertoire musical pour le catalogue du Virtuoso.

Arrivé au terme du temps imparti, ce projet aboutit à un démonstrateur fonctionnel fascinant, combinant art et technologies. Il a été passionnant et très formateur d'implémenter toutes ces technologies, et la satisfaction est grande d'arriver à un résultat probant.

5 Introduction



Dans le cadre des travaux de bachelor à la Heig-vd, nous avons dû choisir un projet.

Le projet choisi consiste en un xylophone automatisé nommé Virtuoso. Son but est de pouvoir jouer, à peu de choses près, n'importe quelle partition et que la musicalité soit bonne. Gérant ainsi les tempos de la musique ainsi que l'intensité de chaque note. Dans cette optique, les partitions doivent pouvoir être modifiées et converties afin de les adapter au xylophone et ses 3 octaves. Son autre objectif est de faire la promotion de l'Heig-vd aux évènements promotionnels. Pour que les visiteurs aient une interactivité avec le projet, un wifi avec un page Web sont prévus.

Dans l'état, le Virutoso est composé d'un boitier électronique gérant 37 actuateurs correspondant chacun à une touche du xylophone. Sur le dessus des actuateurs, il est équipé de LEDs offrant une meilleure visibilité. Du boitier électronique, il en sort 4 fils. Deux d'entre eux sont la masse et le 5 volts et les deux autres donnent accès à une communication en i2c.

Pour que les objectifs soient respectés, ce projet intègre beaucoup de technologies qui doivent être mises en place et par conséquent engage beaucoup de travail.

6 Pré-étude

6.1 Cahier des charges

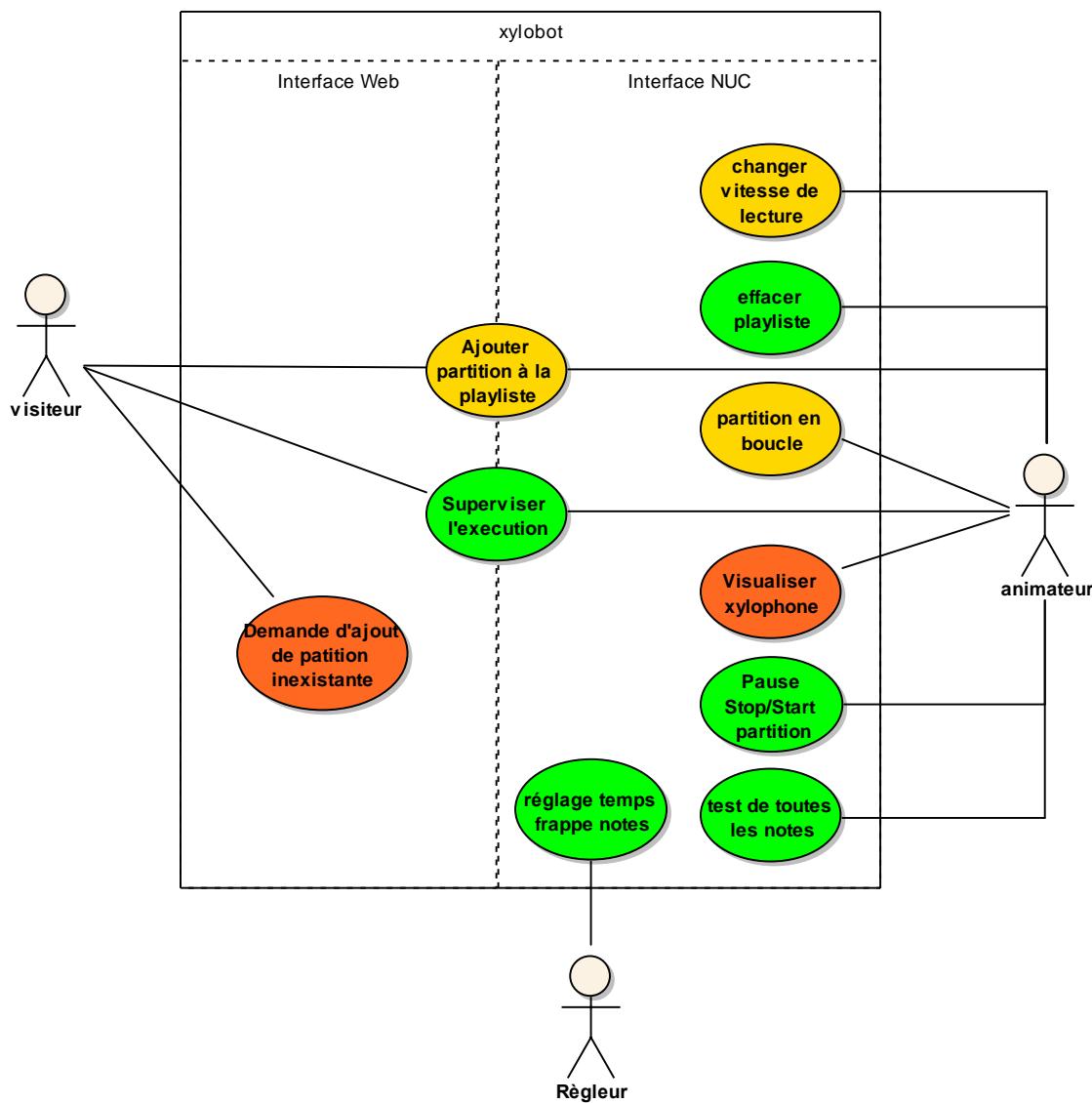
6.1.1 Problème

Pour la réalisation de ce projet qui engage beaucoup de travail, nous avons dû faire des choix concernant un cahier des charges. À l'aide des charges définies dans le cahier, le planning du projet pourra être défini en grande partie.

6.1.2 Démarche

Afin de réaliser le cahier des charges, un diagramme des cas d'utilisations a tout d'abord été fait et modifié afin de correspondre aux attentes voulues. Pour bien séparer le Virtuoso du programme pour la conversion des fichiers, nous avons réalisé deux diagrammes distincts. Puis à l'aide de ces diagrammes, pour chaque cas d'utilisation, nous avons défini des charges que nous avons assemblées pour créer le cahier.

6.1.3 Interface utilisateur



6.1.3.1 Interface Web

Superviser l'exécution

- 10.10 L'interface Web doit permettre de suivre l'exécution sur l'écran en visualisant le titre du morceau joué ainsi que sa position dans la playlist.
- 10.20 L'interface Web devrait permettre de voir un aperçu du Virtuoso avec chaque note et montrer quelles notes sont jouées en direct.

Demande d'ajout de partition inexistante

- 20.10 L'interface Web devrait permettre de faire une demande d'ajout d'une partition qui n'est pas dans la liste des partitions enregistrées.

6.1.3.2 Interface NUC

Superviser l'exécution

- 10.10 L'écran tactile doit permettre de suivre l'exécution sur l'écran en visualisant le titre du morceau joué ainsi que sa position dans la playlist.
- 10.20 L'écran tactile devrait permettre de voir un aperçu du Virtuoso avec chaque note et montrer quelles notes sont jouées en direct.

Effacer playlist

- 20.10 Il doit être possible de supprimer la playlist.

Pause Start/Stop partition

- 30.10 L'écran tactile doit permettre de mettre sur pause la partition qui est jouée.
- 30.20 L'écran tactile doit permettre la lecture d'une partition.
- 30.30 Le Start permettra de jouer, depuis le début, la partition de la playlist qui a été stoppée.

Test de toutes les notes

- 40.10 L'écran tactile doit permettre de lancer un programme test qui actionne chaque note à la suite.

Réglage du temps de frappe des notes

- 50.10 L'interface doit permettre de régler le temps de frappe de chaque note indépendamment.

Changer la vitesse de lecture

- 60.10 Il devrait être possible de changer la vitesse de lecture du Virtuoso.

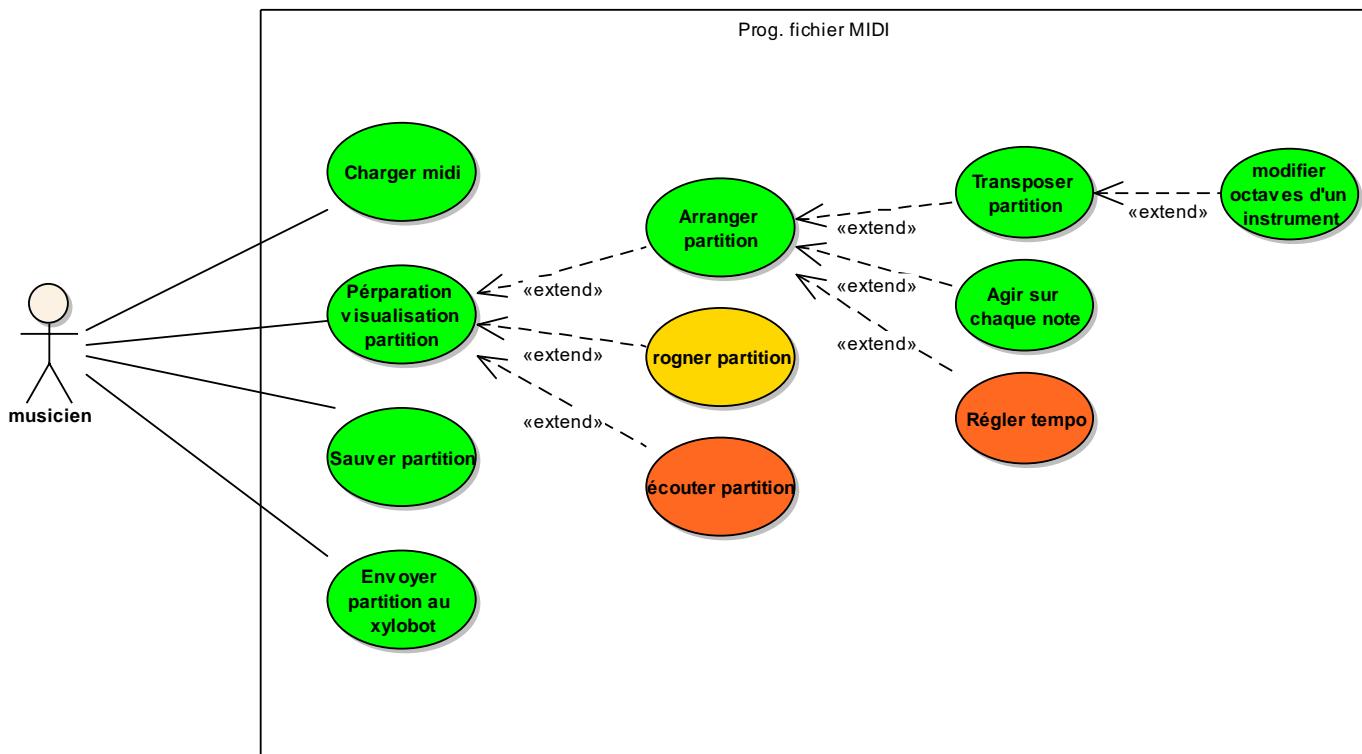
Partition en boucle

- 70.10 L'interface devrait permettre de jouer une partition en boucle.

Visualiser xylophone

- 80.10 Il devrait être possible de voir chaque note du xylophone en direct (voir les notes qui s'activent).

6.1.4 Programme de traitement des fichiers midi



Charger un fichier midi

- 10.10 Le programme doit permettre de charger un fichier au format midi et d'en extraire les notes.

Visualisation la partition

- 20.10 Le programme doit pouvoir visualiser les notes, avec leur hauteur et leur tic, de la partition chargée. Les notes venant de différents canaux (instruments) doivent être distinguées.

Arranger la partition

- 30.10 Le programme permettra de transposer toute la partition d'un nombre de ton(s) choisi(s).
- 30.11 Il doit pouvoir transposer chaque instrument d'un nombre de ton(s) choisi(s).
- 30.20 Il pourra agir sur chaque note indépendamment pour modifier sa hauteur ou son tic.
- 30.30 Le programme devrait permettre de régler le tempo de la partition.

Rogner la partition

- 40.10 Il devrait être possible de sélectionner la zone que l'on va garder sur une partition.

Écouter la partition

50.10 Le programme devrait pouvoir écouter la partition.

Sauver la partition

60.10 Le programme doit pouvoir sauver une partition sous forme XML.

Envoyer la partition au Virtuoso

50.10 Le programme devrait permettre de transmettre une partition au Virtuoso à l'aide de Woopsa.

6.2 Évaluation des risques

6.2.1 Communication I2C

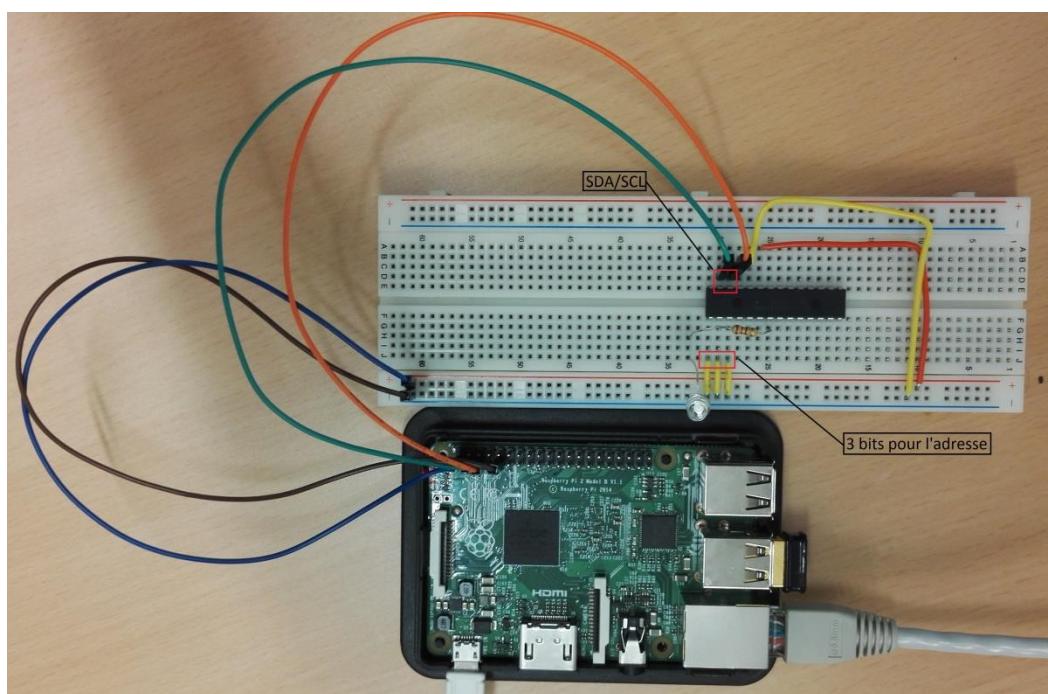
6.2.1.1 Problème

Comme la partie électronique du Virtuoso était implémentée avec des MCP23017 et prévue pour communiquer en I2C, le choix de la communication pour piloter les actuateurs s'est donc fait d'office. Il faut donc relever les risques de la communication I2C.

6.2.1.2 Démarche

Ainsi pour tester l'I2C, un programme a été codé sur un Raspberry Pi. Celui-ci fut créé juste pour tester l'I2C et savoir comment l'implémenter sur Windows IoT.

L'image suivante représente le câblage implanté pour le programme de test. Avec cette configuration, nous avons l'adresse 0x20 pour le MCP décrite juste après.



L'adressage pour le MCP23017 est sur 7 bits. Les 4 msb sont fixés et les 3 derniers peuvent être choisis. Dans notre cas, les 3 lsb valent 0, nous avons comme adresse $010\ 0000_2 = 0x20$.

L'utilisation de l'I2C est plutôt simple, car il suffit de trouver les bons registres à modifier. Pour le MCP23017, le registre des directions IO A est 0x00 et celui des valeurs IO A est 0x14. Ces registres nous permettent d'initialiser l'I2C puis de communiquer. Le programme initialise l'I2C :

```
string deviceSelector = I2cDevice.GetDeviceSelector();
var i2cDeviceControllers = await DeviceInformation.FindAllAsync(deviceSelector);
if (i2cDeviceControllers.Count == 0)
    return;
var i2cSettings = new I2cConnectionSettings(MCP_I2C_ADDR);
//i2cSettings.BusSpeed = I2cBusSpeed.FastMode;
I2CMCP = await I2cDevice.FromIdAsync(i2cDeviceControllers[0].Id, i2cSettings);
if (I2CMCP == null)
    return;
```

En donnant l'adresse du MCP (« MCP_I2C_ADDR » qui est 0x20), on obtient une variable « I2cDevice » qui nous permet d'écrire au MCP.

Puis le programme initialise les registres du MCP :

```
I2CMCP.WriteRead(new byte[] { MCP_REG_IODIRA }, i2CReadBuffer);
iodirRegister = i2CReadBuffer[0];

I2CMCP.WriteRead(new byte[] { MCP_REG_GPIOA }, i2CReadBuffer);
gpioRegister = i2CReadBuffer[0];

//Met tout les pins du port A comme output
byte bitMask = (byte)(0xFF ^ LED_GPIO_PIN); // set the LED GPIO pin mask bit to '0', all other bits to '1'
iodirRegister &= bitMask;
byte[] i2CWriteBuffer = new byte[] { MCP_REG_IODIRA, iodirRegister };
I2CMCP.Write(i2CWriteBuffer);
```

Avec les registres du MCP MCP_REG_IODIRA = 0x00 et MCP_REG_GPIOA = 0x14.

Et ensuite la communication est simple, on donne l'adresse et la valeur du registre que l'on veut écrire sous la forme d'un tableau de bytes :

```
private void FlipLED()
{
    byte bitMask;
    if (isLedOn == true)
    {
        // turn off the LED
        isLedOn = false;
        gpioRegister |= LED_GPIO_PIN;
        I2CMCP.Write(new byte[] { MCP_REG_GPIOA, gpioRegister });
    }
    else
    {
        // turn on the LED
        isLedOn = true;
        bitMask = (byte)(0xFF ^ LED_GPIO_PIN);
        gpioRegister &= bitMask;
        I2CMCP.Write(new byte[] { MCP_REG_GPIOA, gpioRegister });
    }
}
```

6.2.2 Type de fichier converti pour le Virtuoso : MIDI

6.2.2.1 Problème

Pour « alimenter » le Virtuoso en musique, il faut trouver un type de fichier à convertir. Dans l'idéal, il faut un type qui aurait une grande « bibliothèque » de musique. De plus, nous avons besoin de fichiers qui sont sous forme de partitions pour récupérer facilement les notes et autres informations utiles.

6.2.2.2 Démarche

Après quelques recherches et propositions, le MIDI a été choisi pour les raisons suivantes :

Pour les fichiers sons, le MIDI allait parfaitement étant donné qu'il permet d'obtenir chaque note jouée et de les séparer par instrument (chaque instrument correspond à un canal dans le fichier MIDI). De plus, il y a énormément de fichiers MIDI téléchargeables sur internet, de quoi couvrir tous les goûts.

Après plusieurs recherches sur des librairies C# donnant la possibilité de convertir les fichiers MIDI, nous sommes tombés sur miditoolkit. Comme elle s'apprêtait bien à nos attentes et qu'elle était open source, nous avons essayé de produire un code récupérant les notes d'un fichier. Dus au manque d'exemples à disposition, pour l'utilisation que nous souhaitions en faire, après quelques heures, nous avons finalement trouvé un code permettant de faire le résultat recherché.

6.2.2.3 Test de la librairie miditoolkit

Afin de s'assurer que la librairie fonctionne bien et qu'elle offre les fonctionnalités nécessaires, un programme test a été réalisé par nos soins.

Ce programme commence par charger le fichier midi avec un chemin en brut dans le code :

```
sequence = new Sequence("ARRANGEMENT_brahms_hungarian_dance_5.mid");
```

Puis il parcourt le fichier et charge les données dans une ListBox :

```
foreach (Track t in sequence) {
    foreach (MidiEvent midiEvent in t.Iterator())
    {
        if (midiEvent.MidiMessage.MessageType == MessageType.Channel)
        {
            tick = midiEvent.AbsoluteTicks;
            ChannelMessage msg = ((ChannelMessage)midiEvent.MidiMessage);
            if (msg.Command == ChannelCommand.NoteOn)
            {
                partition.Add(NotesConvert.IdToNote(msg.Data1, tick));
                ListBoxNotes.Items.Add(partition[partition.Count - 1] + "\t"
                    + msg.MidiChannel.ToString() + "\t" + tick.ToString());
            }
        }
    }
}
```

Pour chaque note trouvée dans la partition, le code la charge dans une liste de « notes ». Puis l'affiche en l'insérant dans une « ListBox ». Avec la note, nous affichons aussi son canal et son tick.

MI 6	0	30458
SOL# 6	0	30462
MI 6	0	30463
MI 6	0	30652
SI 7	0	30840
RE 7	0	30840
SI 6	0	30840
SOL# 6	0	30840
MI 6	0	30845
RE 6	0	30848
LA 6	0	31048

Pour l'affichage, le programme implémente une interface graphique très rudimentaire permettant de visualiser les résultats avec 3 colonnes qui sont la note, le canal et le tick.

6.2.3 Windows IoT

6.2.3.1 Qu'est-ce que c'est ?

Windows IoT est une version de Windows 10 simplifiée pour les microcontrôleurs comme le Raspberry Pi. Il a l'avantage de pouvoir développer à l'aide de Visual Studio sur un PC et de compiler facilement sur un Raspberry en remote debugging.

6.2.3.2 Performances de Windows IoT

6.2.3.2.1 Problème

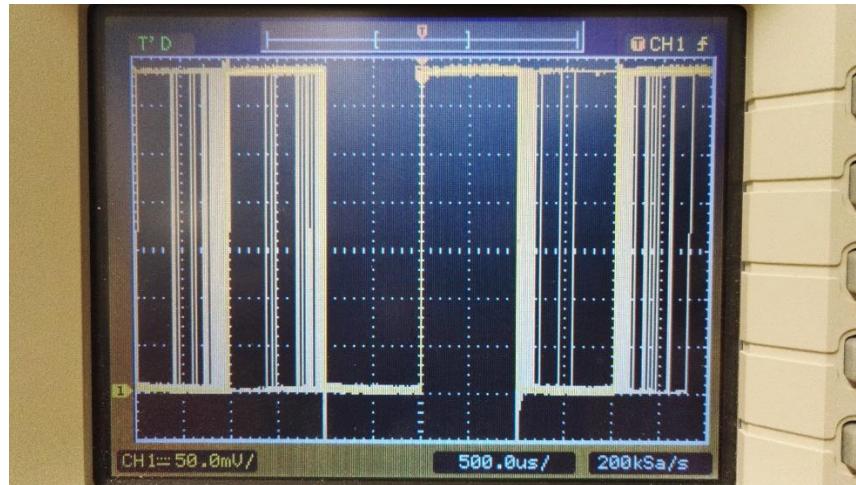
Pour savoir si les performances de Windows IoT sont concluantes pour le pilotage du xylophone, les risques doivent être relevés.

6.2.3.2.2 Démarche

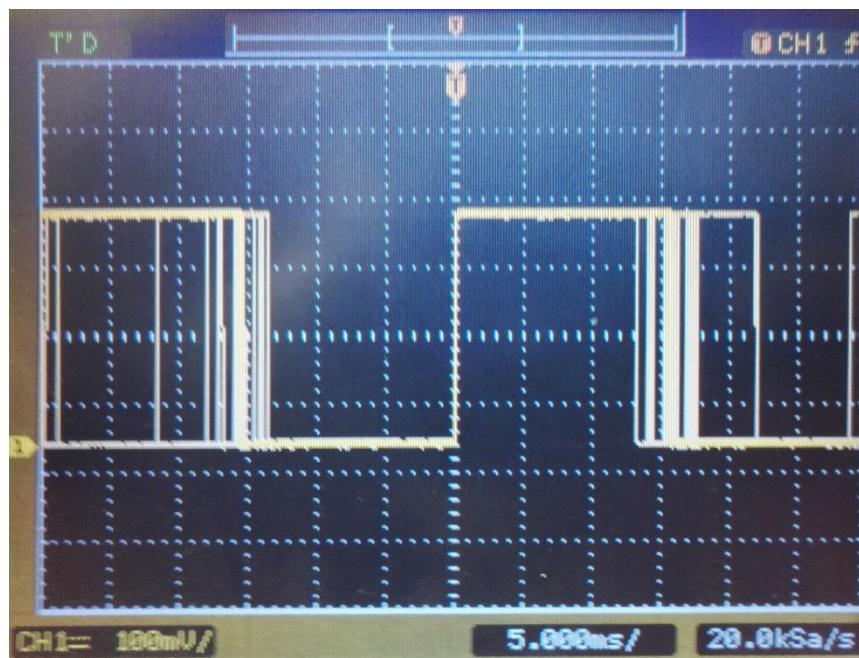
Afin de pouvoir piloter le Virtuoso comme il faut, nous avons besoin d'un temps de réponse précis à la milliseconde. Par conséquent, des tests ont été réalisés avec différente manière.

Les tests sont des clignotements d'une sortie du Raspberry pi qui ont été mesurés à l'oscilloscope avec une rémanence infinie dans le but d'observer la gigue.

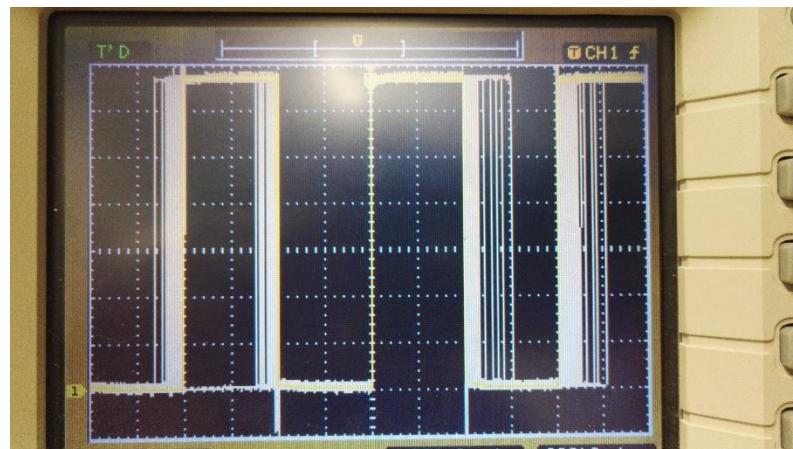
Le premier test est avec une application normale sans thread :



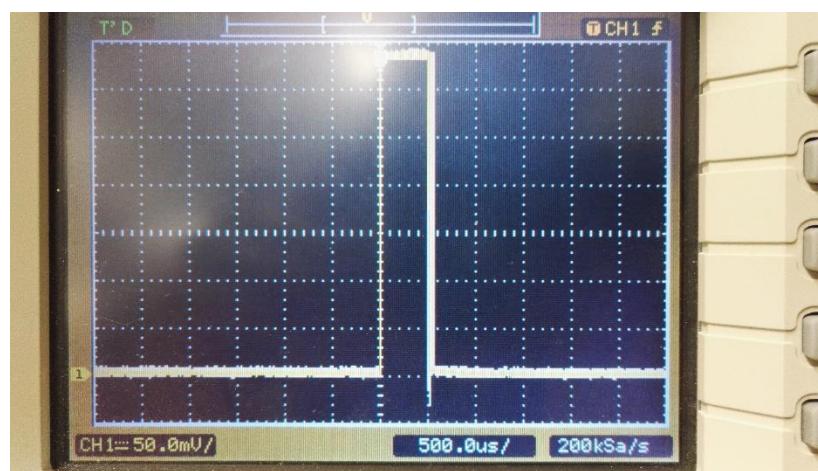
Le deuxième est avec un thread qui tourne avec des attentes passives :



Et le troisième test est avec un thread qui tourne à 100% avec une attente active :



Un autre test a été réalisé avec une application qui tourne en arrière-plan aussi avec un thread qui fonctionne à 100% :



Après ces tests, il a été constaté les faits suivants :

En attente active, Windows IoT permettrait de piloter le Virtuoso avec des temps de l'ordre de la milliseconde avec une gigue de 500 microsecondes. Malheureusement, l'attente active n'est l'objectif souhaité, car il fait tourner un thread à 100%.

Pour ce qui est de l'attente passive, Windows IoT ne tient pas la route, car il offre une précision de l'ordre des 15ms.

6.2.3.3 Solution

Étant donné que les performances de IoT ne nous satisfaisaient pas, nous avons fait le choix de partir sur une autre solution de matériel. Différentes solutions ont été explorées. Ces solutions sont décrites au point 6.3.

6.2.4 Woopsa, HTML & JavaScript

6.2.4.1 Qu'est-ce que c'est ?

Woopsa est un protocole de communication pour le Web qui est open source. Il fonctionne sur le principe de l'orienté objet ce qui le rend pratique pour des programmes codés en objet.

6.2.4.2 Problème

Woopsa fait partit des exigences pour le développement du Virutoso. Car le Virtuoso doit émettre un réseau wifi et gérer un serveur à l'aide de ce dernier. Afin de pouvoir créer un serveur avec une page web, des connaissances en HTML et JavaScript sont obligatoires.

6.2.4.3 Démarche

Dans le but de découvrir un minimum Woopsa, HTML et JavaScript, car ils étaient jusqu'à maintenant inconnus, nous avons généré un programme impliquant un serveur Woopsa et implémentant un page web grâce à l'HTML et au JavaScript.

6.2.4.3.1 Programme de test

Le programme en C# est simple, il implémente une instance de WoopsaServer qui prend une partition « p ». Puis il ajoute les routes avec les pages HTML dans le dossier les contenant.

```
s = new WoopsaServer(p);
s.WebServer.Routes.Add("/files", HttpMethod.GET, new RouteHandlerFileSystem(
    @"C:\Users\quentin\Desktop\TB\PreEtude\Projet\WoopsaTest\Page\"));
```

À l'aide du code ci-dessus, pour accéder aux pages HTML qui sont dans le dossier \Page\, il faut écrire <http://localhost/files/nomDeLaPage.html>

La classe partition est une classe « brouillon » avec une liste de notes.

```
public class Partition
{
    public Partition()
    {
        Notes = new List<Note>();
        a = 3;
    }
    public int a { get; set; }
    public List<Note> Notes { get; set; }
}
```

```
public class Note
{
    public Note(int o)
    {
        Octave = o;
        Name = o.ToString();
    }
    public string Name { get; set; }
    public int Octave { get; set; }
}
```

6.2.4.3.2 HTML/JavaScript

L'HTML est simple, car pour tester l'affichage d'une liste, tout ce qu'il implémente c'est une liste

```
<h1>Xylobot</h1>
<div>
    <h2>Liste</h2>
    <p id="p" style="overflow-y: scroll; height:100px; width:500px;"></p>
</div>
```

Remarque, il implémente aussi un titre pour la liste avec `<h1>Xylobot</h1>`.

En revanche le JavaScript est un peu plus complexe. Pour commencer, il crée une instance de la liste avec l'id « p » ainsi qu'une instance d'un client Woopsa.

```
var parent = document.getElementById("p");
var client = new WoopsaClient("http://localhost/woopsa/", jQuery);
```

Puis au chargement de la page web, il ajoute toutes les notes existantes dans la liste.

```
client.read("/Notes/Count", function(count){
    //Ajoute toutes les notes à la liste-->
    for (var i = 0; i < count; i++){
        AddNote(i);
    }
});
```

La fonction `AddNote(i);` crée un label avec le nom de la note ainsi qu'un champ éditable pour la valeur de l'octave. Le nom ainsi que l'octave sont lus avec l'instance client de Woopsa.

```
<!--Ajoute une note à la liste-->
function AddNote(i){
    <!--Lit la note concernée-->
    client.read("/Notes/Note["+ i +"]/Octave", function (value, name){
        parent.appendChild(document.createElement("br")); <!--crée une nouvelle ligne-->

        <!--Ajoute le nom de la note à la liste-->
        var new_name = document.createElement("label");
        new_name.appendChild(document.createTextNode(name+ " : "));
        parent.appendChild(new_name);

        <!--Ajoute un champ éditable avec la valeur de la note-->
        var new_in = document.createElement("input");
        new_in.value = value;
        new_in.type = "number";
        new_in.id = name;
        new_in.addEventListener("change", ValueChange, name); <!--Implémente la fonction lors d'un changement-->
        parent.appendChild(new_in);
    });
}
```

Et pour finir, lorsqu'un changement intervient dans le champ éditable, on appelle la fonction « ValueChange » avec le nom de la note impliquée.

Cette fonction récupère l'objet grâce au nom et renvoie au serveur la nouvelle valeur pour l'objet concerné à l'aide Woopsa.

```
<!--Lors d'un changement, envoie la nouvelle valeur au programme-->
function ValueChange(name){
    <!-- récupère l'objet qui a changé-->
    var o = document.getElementById(name.target.id);
    console.log(o.id);
    <!-- Le client écrit au programme la nouvelle valeur-->
    client.write(o.id, o.value, function(){});
}
```

6.2.5 Concept HMI

6.2.5.1 Qu'est-ce que c'est ?

Concept HMI est une librairie pour le C# permettant de faciliter et d'accélérer le codage d'un programme grâce aux fonctionnalités qu'elle implémente.

6.2.5.2 Problème

La question s'est posé de savoir s'il serait utile d'utiliser ou non concept pour les applications du Virtuoso et de la conversion des fichiers midi. Ainsi, des tests doivent être réalisés pour répondre à cette question.

6.2.5.3 Démarche

Pour expérimenté Concept HMI, un rendez-vous chez Objectis, qui est la boîte développant Concept, a été pris. Objectis nous a fourni des tutoriels pour l'utilisation de concept avec des licences pour le développement d'application.

Les premiers points du tutoriel ont été réalisés ainsi que ceux ciblant plus les objectifs du Virtuoso. Notamment la partie XML. À la fin des tests, la décision fut prise d'utiliser concept surtout pour la partie XML gérant la sauvegarde et les chargements automatiquement.

Par contre, pour la suite du projet, il reste à voir comment changer les styles de concept pour correspondre aux attentes de l'application.

6.3 Comparatif des plateformes

6.3.1 Problème

Après une évaluation des risques, il s'est avéré que l'idée de base qui était d'utiliser Windows IoT ne correspondait pas aux objectifs du Virtuoso. Il a donc fallu procéder à un choix d'une nouvelle plateforme.

6.3.2 Démarche

Afin de résoudre le problème de la plateforme IoT, un comparatif a été accompli. Il compare chaque possibilité de pilotage (ex. : Linux avec MONO) avec les éléments requis pour le bon déroulement du projet. Et permet de faire un choix rapide et efficace.

	Linux (MONO)	Windows IoT	Windows IoT + Arduino	NUC + Arduino
Interface graphique	- En HTML	+ En C#, langage maîtrisé	+ En C#, langage maîtrisé	+ En C#, langage maîtrisé
Lecture/écriture des fichiers	+ Fonctionne de base	Test exemple de msdn ne fonctionne pas	Test exemple de msdn ne fonctionne pas	+ Fonctionne sur Windows
Temps de réponse	+ suffisant pour le Virtuoso	Pour obtenir un temps de réponse satisfaisant, il faut faire tourner un thread à 100%	+ Gérer par l'Arduino, suffisant	+ Gérer par l'Arduino, suffisant
I2C	+ Implémenter directement de MONO au xylophone	+ Implémenter directement du Raspberry au xylophone	- Passe par l'intermédiaire de l'Arduino	- Passe par l'intermédiaire de l'Arduino
Woopsa	+ Aucune modification à reprendre	- Modification à faire	- Modification à faire	+ Aucune modification à reprendre
Accès hardware	- Inconnu	+ Déjà testé et fonctionne parfaitement	- Inconnu sur la partie Arduino	
Debugging				
Temps à investir	- Apprendre à utiliser MONO - Revoir certains points déjà connus sur IoT	- Apporter des modifications à Woopsa	- Apporter des modifications à Woopsa - Apprendre à utiliser Arduino	- Apprendre à utiliser Arduino + Interface sur C#, utilisation de concept possible

À la suite du tableau ci-dessus, le choix du NUC de Intel avec un Arduino est le choix optimal. L'autre choix qui pourrait être pris serait le Raspberry avec MONO, mais celui-ci n'offre pas la fonctionnalité importante du debugging. En revanche, pour la partie I2C, Raspberry avec MONO n'a pas besoin de traitement particulier pour déléguer la communication avec le xylophone.

Un inconvénient du NUC est le prix qui est facilement le triple des autres. Mais en contrepartie, il demande beaucoup moins de temps de développement, car il utilise certaines technologies déjà connues .En plus de cela, il sera possible d'utiliser concept HMI pour la partie Virtuoso.

6.4 Choix du matériel

6.4.1 Problème

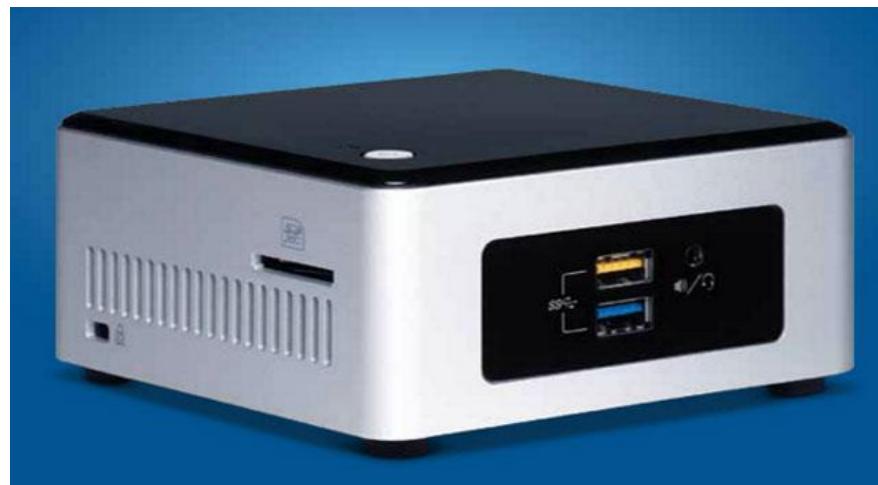
À la fin du comparatif des plateformes, après avoir fait le choix de partir avec un NUC combiné avec un Arduino, il a fallu faire un choix pour le matériel.

6.4.2 Démarche

Le choix pour l'Arduino a été vite fait, car nous avons repris celui déjà disponible avec le Virtuoso. (Un Arduino était fourni de base avec le Virtuoso pour le travail de bachelor)

Du côté du NUC, après des recherches, le NUC5PGYH fut le meilleur choix pour les raisons suivantes :

- Un disque dur SSD 32Go implémenté de base
- Windows 10 déjà installé
- 2Go ram DDR3 mis de base
- Un prix pas trop élevé (260.-)
- Disponible et en stock sur digitec.ch



6.5 UML

6.5.1 Problème

Afin d'avoir une idée de la direction dans laquelle part le projet, un diagramme UML est nécessaire avant le développement.

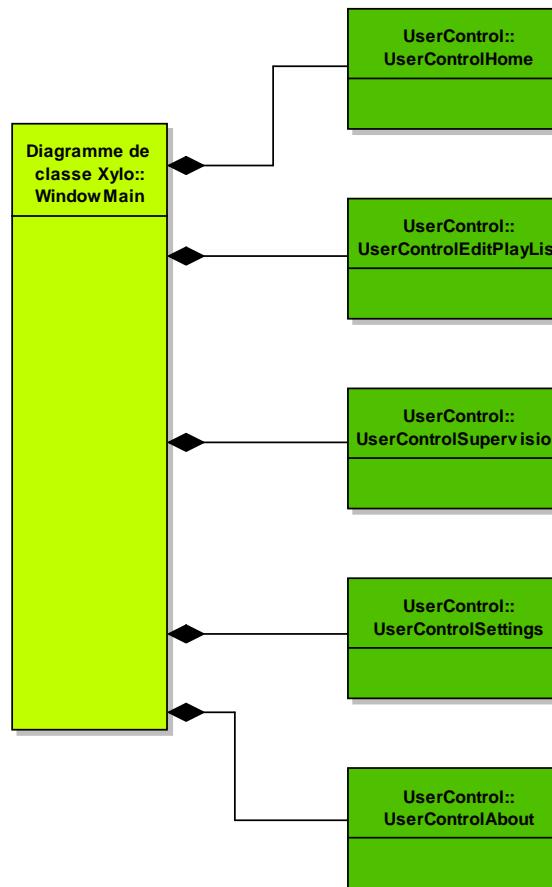
6.5.2 Démarche

Le diagramme UML décrit ici est le diagramme obtenu à la fin de la pré-étude. Ce diagramme nous sert, d'une part, comme base pour le planning du projet, car il décrit dans les grosses lignes ce que devra implémenter le projet. Et d'autre part, pour structurer le code des applications dès le début.

L'UML complet contient les UserControls ainsi que toutes les classes « prédictes » pour les codes du projet. Il en découle donc plusieurs diagrammes.

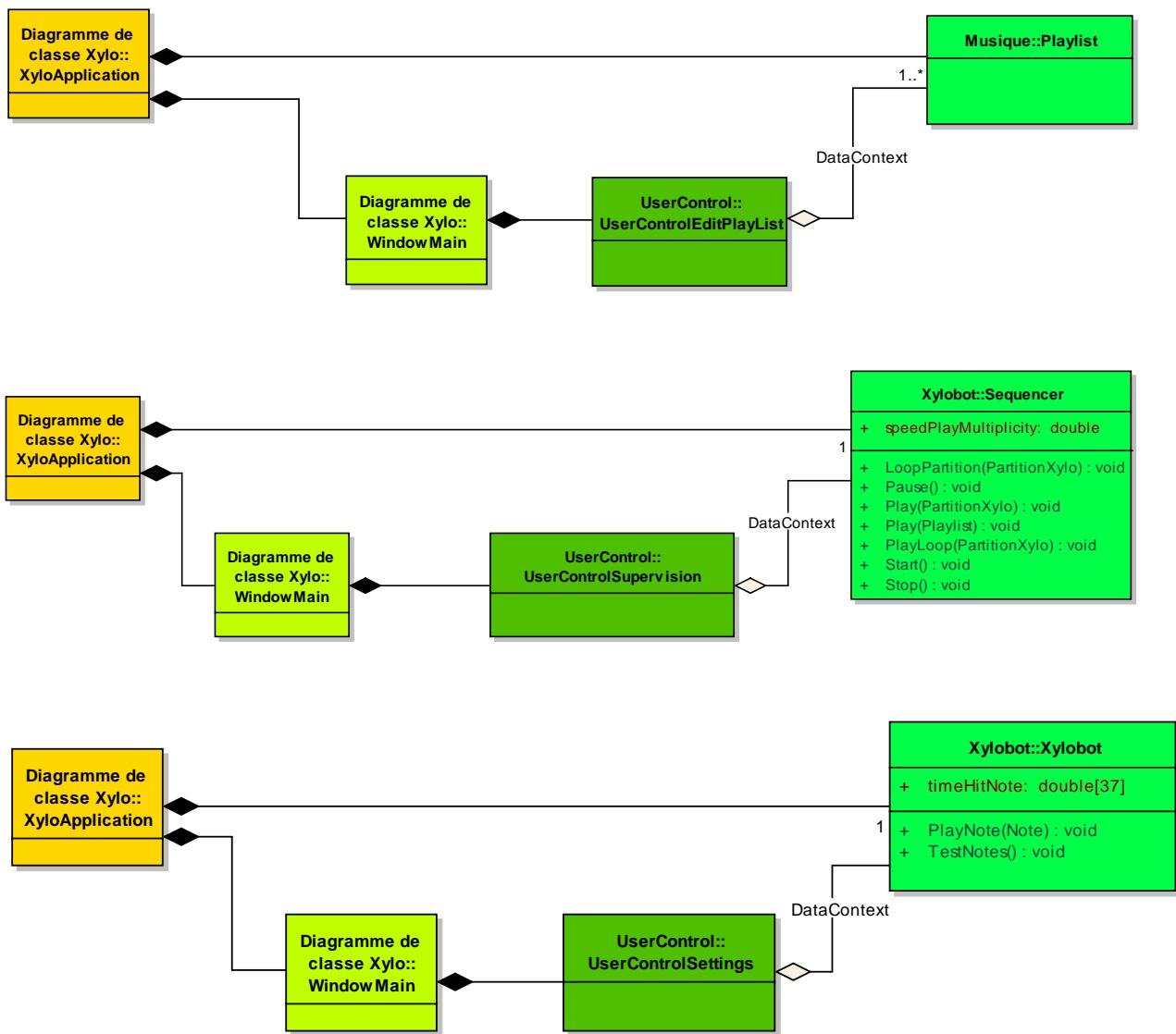
6.5.3 Programme Virtuoso

Pour la partie « Xylophone » : le diagramme HMI donne lieu à plusieurs « UserControl » :



Ces différents UserControls sont les vues offertes par l'application. Les vues « Home » et « About » n'utilisent aucune autre classe, elles font juste partie de l'interface utilisateur.

Par contre, les vues « EditPlaylist », « Supervision » et « Settings » utilisent d'autres classes comme montrées ci-dessous.

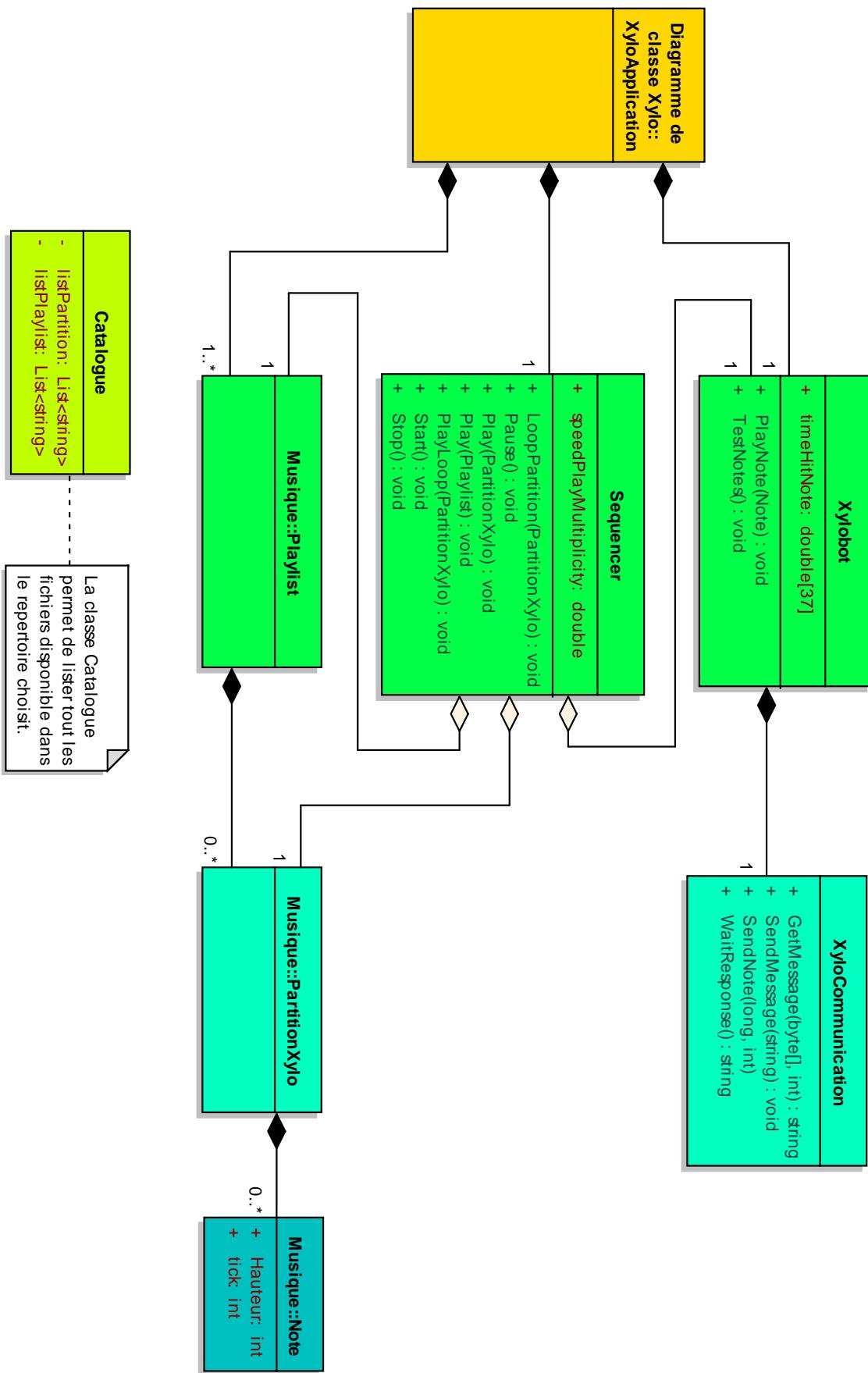


La partie « EditPlaylist » utilise la playlist pour accéder à ces propriétés et les modifier. Pouvant ainsi permettre à l'utilisateur de modifier la playlist du xylophone par le bief du UserControl.

La partie « Supervision » utilise le séquenceur offrant à l'utilisateur les fonctions play/pause, stop, partition précédente ou suivante.

Et enfin, la partie « Settings » est là pour les réglages du Xylobot comme par exemple le temps frappe de chaque note. Permettant ainsi l'ajustement de l'intensité.

Après cela, nous avons les classes que composera le programme Virtuoso :



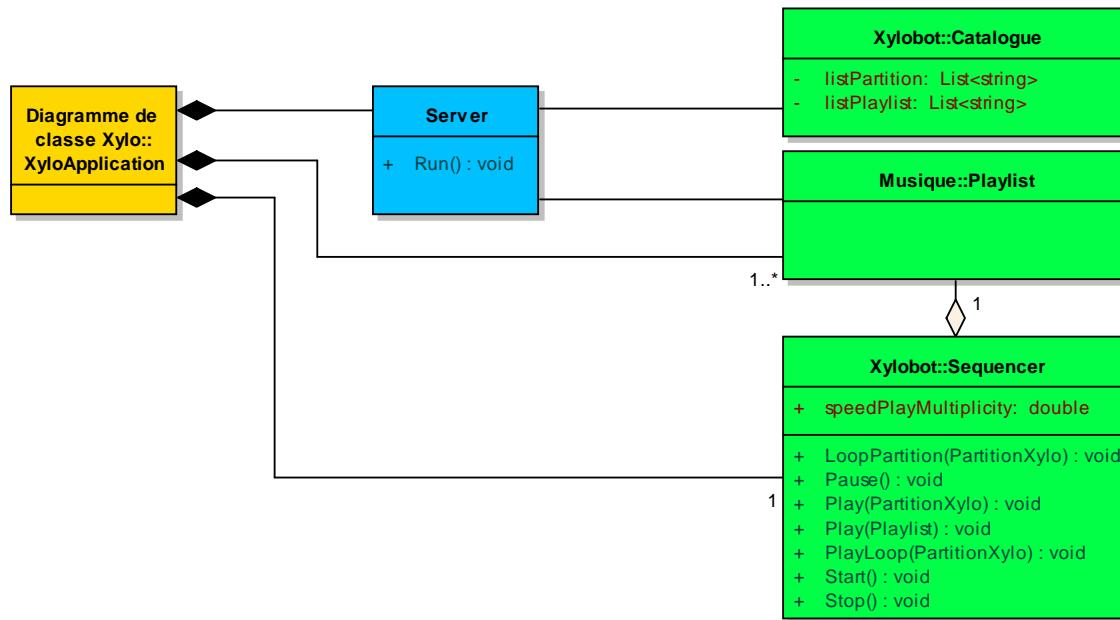
Nous n'allons pas décrire toutes les classes, mais en bref nous avons le Xylobot, qui représente le xylophone avec ces 37 notes et actuateurs, qui contient une gestion de la communication avec le XyoCommunication. Le Xylobot offre la possibilité de tester toutes les notes pour vérifier que tout soit opérationnel.

Ensuite vient la Playlist, qui naturellement contient plusieurs partitions (partition faite pour le xylophone) et chacune de ces partitions a des notes.

Et pour finir, le Sequencer, qui gère un peu tout derrière pour la lecture des partitions.

Le catalogue est là pour la gestion de la recherche. Il liste toutes les partitions disponibles.

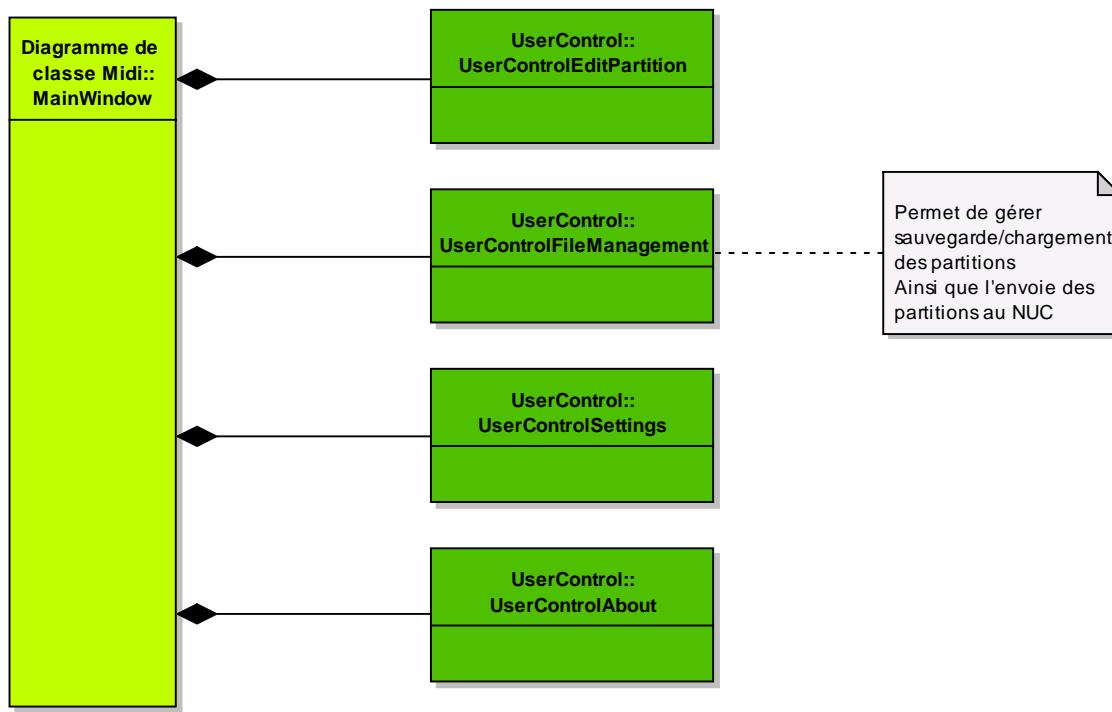
Il reste le diagramme pour le serveur Web :



Avec le Server qui utilise la Playlist, car les clients (au sens web du terme) se connectant dessus pourront seulement faire des propositions de partitions à jouer et voir la playlist en cours. Il inclut aussi le Sequencer, car il offrira la possibilité de voir l'avancement de la partition courante (comme décrit dans le cahier des charges).

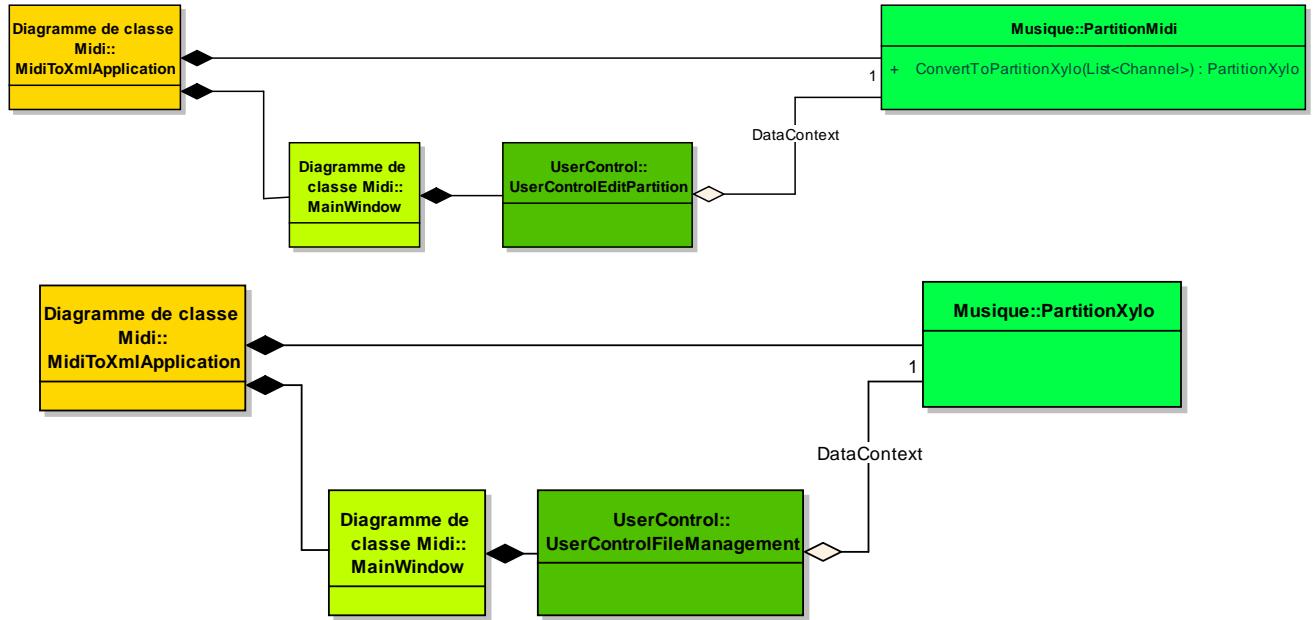
6.5.4 Programme midi→XML

Comme pour la partie Xylophone, nous avons le diagramme avec les UserControls :



La partie principale du programme est l'édition de partitions. Elle permet de modifier une partition midi pour ensuite l'enregistrer et l'envoyer au xylophone à l'aide du FileManagement.

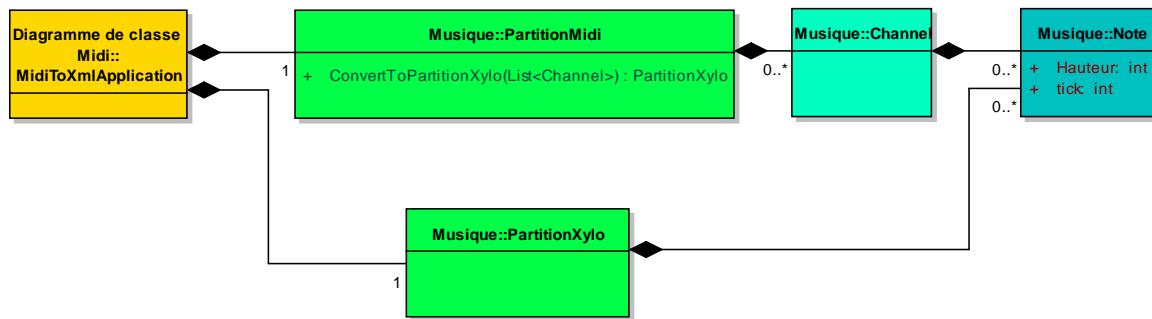
On obtient les deux diagrammes suivants :



Comme le Virtuoso utilise des PartitionXylo, le FileManagement utilise aussi une PartitionXylo, car c'est lui qui les envoie au Virtuoso.

L'édition de partition se fait avec une PartitionMidi obtenue directement depuis un fichier midi.

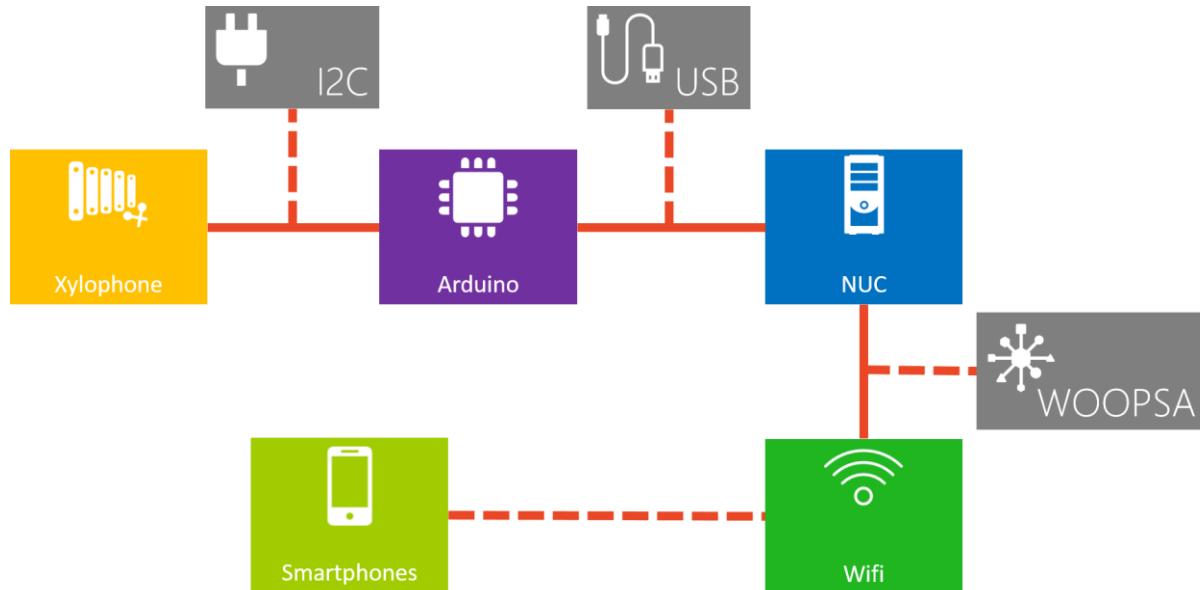
Les différences entre les PartitionXylo et PartitionMidi sont les canaux. Une PartitionMidi a plusieurs canaux représentant plusieurs instruments tandis que la PartitionXylo n'a pas de canal étant donné qu'elle ne compose qu'un instrument.



7 Projet

7.1 Schéma de la communication

Afin de mieux visualiser comment se combinent toutes les pièces du Virtuoso, nous avons fait un schéma représentatif :



Il décrit comme les appareils vont communiquer entre eux.

7.2 Protocole de communication

Afin de pouvoir communiquer entre l'Arduino et le NUC, un protocole de communication USB était nécessaire.

7.2.1 Composition des messages

Message envoyé par le NUC :

Byte de Start	Numéro de message	Type	Taille des données	Données
1 byte	1 byte	1 byte	2 bytes	X bytes

Message envoyé par l'Arduino:

Byte de Start	Numéro de message	Type	Taille disponible dans le buffer	Tick courant (Tick pour les notes)
1 byte	1 byte	1 byte	1 byte	4 bytes

Dans les deux sens, tous les messages commencent par un Byte de Start puis un numéro de message et un type. Ce Byte de Start vaut la valeur max d'un Byte qui est 255.

7.2.2 Types de messages

NUC	Arduino
Tempo	0000 0000 ₂
Notes	OK
Start	0000 0001 ₂
Stop	Too many data
Pause	Erreur Start Byte
	0000 0010 ₂
	Erreur Type
	0000 0011 ₂

Le NUC envoie un paquet à l'Arduino avec un numéro de message. L'Arduino traite le message et renvoie au NUC un message avec le même numéro et ainsi de suite.

Si le NUC ne reçoit pas le message de réponse après un délai, il renvoie une fois les données et après cela, s'il n'a toujours pas de réponse, il génère une erreur.

Quand l'Arduino reçoit un message qu'il a déjà reçu, il ne le traite pas, mais renvoie quand même une réponse. Si la taille des données qu'il reçoit est trop grande, l'Arduino renvoie « Too many data ».

Si le NUC ne reçoit pas de réponse avant un certain temps, il génère une erreur.

La vitesse de transmission des données n'est pas modifiable. Elle est définie dans le code.

7.3 Réglages USB

7.3.1 Problème

Afin de pouvoir communiquer en USB de l'Arduino au NUC, il faut régler les paramètres de la communication USB des deux côtés.

7.3.2 Démarche

Dans le but de réaliser la communication le plus simplement possible, les paramètres du côté Arduino n'ont pas été changés hormis la vitesse de communication (baud rate).

```
Serial.begin(BAUD_RATE_SERIAL);
```

Avec BAUD_RATE_SERIAL qui est une constante valant 115'200.

Pour ce qui est du programme du NUC, les paramètres sont ajustés en correspondance à ceux de l'Arduino.

```
_portName = _defaultPortName;
_serialPort = new SerialPort();
_serialPort.PortName = _portName;
_serialPort.BaudRate = BaudRate;
_serialPort.ReadTimeout = TimeOut;
_serialPort.WriteTimeout = TimeOut;
_serialPort.DataBits = 8;
_serialPort.StopBits = StopBits.One;
_serialPort.Parity = Parity.None;
_serialPort.Handshake = Handshake.None;
```

La BaudRate est réglée identiquement à l'Arduino, c'est-à-dire 115'200.

7.4 Communication USB

7.4.1 Partie NUC

La communication USB utilise le protocole décrit précédemment.

La partie NUC est gérée par la classe XyloCommunication. Cette classe implémente les fonctions suivantes permettant de gérer la communication :

```
private ReceiveTypeMessage Read()...
```

```
public void SendDatas(SendTypeMessage typeMessage, List<byte> datas)...
```

```
public void SendMessage(SendTypeMessage typeMessage)...
```

```
private byte[] HeaderMessage(ushort dataSize, byte type)...
```

Remarquez que les seules fonctions publiques sont celles pour l'envoi de messages. Ceci est dû au fait que lors de l'envoi d'un message, l'Arduino répond directement. La lecture des données est donc faite et contrôlée dans les fonctions « Send » à l'aide de la fonction « Read ».

Les fonctions de communication utilisent les fonctions de SerialPort, qui est une classe déjà implémentée par Visual Studio. Pour la gestion des erreurs, les fonctions d'envoi regardent si elles reçoivent un message de type « OK ». Si ce n'est pas le cas, elle renvoie le message jusqu'à un maximum de 10 fois. Finalement si le message reçu n'est toujours pas concluant, la fonction renvoie une erreur.

```

try
{
    int countSend = 0;
    do
    {
        int i = 0;
        byte[] msg = new byte[SizeHeadMessage + datas.Count];
        ushort dataSize = (ushort)(datas.Count);
        byte[] headMsg = HeaderMessage(dataSize, (byte)typeMessage);
        for (i = 0; i < SizeHeadMessage; i++)
            msg[i] = headMsg[i];
        foreach (byte data in datas)
            msg[i++] = data;
        //Envoie
        _serialPort.DiscardOutBuffer();
        _serialPort.Write(msg, 0, msg.Length);

        if (countSend++ > 9)
            throw new Exception("Send-receive message fail too many times.");

    } while (Read() != ReceiveTypeMessage.Ok);
}
catch (TimeoutException e)
{
    throw e;
}
_numMessage++;

```

7.4.2 Partie Arduino

L'implémentation pour l'Arduino a posé quelques problèmes. Car un premier code a été fait, basé sur une faute. Ce code envoyait des données à l'Arduino et celui-ci renvoyait les mêmes. Les données envoyées et reçues concordaient bien, mais plus tard il a été remarqué que l'Arduino traitait déjà des données sans en avoir reçu la totalité. À cause de cela, en checkant s'il y avait des données dans le buffer USB (fonction Arduino: Serial.available()) on les détectait, mais elles n'étaient pas encore toutes arrivées. Ainsi dans l'exemple suivant, le programme recevait les bytes de données et on les renvoyait un par un au lieu de les grouper en un bloc d'une taille de SIZE_MSG :

```

void loop() {
    int idx = 0 ;
    byte msg[SIZE_MSG] ;
    while(Serial.available())
    {
        msg[idx] = Serial.read();
        idx++ ;
        if (idx >= SIZE_MSG)
            break;
    }

    Serial.write(msg, idx);
}

```

Il a fallu réadapter le code en prenant compte cet effet. Ainsi on obtient un code avec la partie de réception qui est fragmenté en partie agissant comme une machine d'état.

```

switch (modeSerial)
{
    case 0: //entête du message
        if(ReadHeadMessage()) //Attend de recevoir tout l'entête du message avant de continuer
            modeSerial = 1;
        break;

    case 1: //Validité du message? et lecture des données
        if(CheckMessage())
            modeSerial = 2;
        else
            modeSerial = 3;
        break;

    case 2:
        ReadDataMessage();
        modeSerial = 3;
        break;

    case 3:
        while(Serial.available()) //Vide le Buffer d'entrée
            Serial.read();
        ResponseMessage(msgSendType);
        modeSerial = 0;
        break;

    default:
        break;
}

```

Le switch-case ci-dessus est dans la fonction de base « `void loop()` ».

À chaque passage dans `loop`, le buffer est checké et s'il y a des données qui sont arrivées, on regarde s'il est valide au prochain passage dans `loop`. Si le message est valide, on lit les données. Finalement, on répond au NUC et on attend le prochain message.

Un autre problème qui survenu et qu'il a fallu débugger est la vitesse de communication. Après quelque recherche rapide sur le site Arduino pour les vitesses disponibles pour l'Arduino méga et une vitesse a été choisie. Or il s'est avéré que cette vitesse de communication était en fait pas adaptés au méga que je possède. En conséquence, quand on exécutait le code de test (envoie des données à l'Arduino et celui-ci renvoie les mêmes) deux bytes en trop apparaissaient au début des messages reçus dans l'ordinateur.

Message envoyé :	byte 0	byte 1	byte 2	byte 3	byte 4
	255	0	2	128	0

Message reçu :	byte 0	byte 1	byte 2	byte 3	byte 4
	171	171	255	0	2

7.4.3 Remarque

L'Arduino offre une possibilité de débogage grâce au port USB par lequel la compilation du programme est faite. Or cette fonctionnalité n'a pas pu être utilisée étant donné que le port USB était déjà occupé par la communication NUC – Arduino. Ce point non négligeable a posé quelques problèmes notamment pour le débogage de la communication I2C.

7.5 I2C

La partie I2C a donné plus de fil à retordre que prévu. Ceci dut au mauvais choix de partir avec des interruptions. Et il s'est avéré qu'il n'était pas possible de communiquer en I2C à l'intérieur d'une interruption, car les fonctions de bases sont bloquantes.

Les interruptions étaient à première vue un bon choix, car pour jouer la musique, il intervient un tempo régulier donnant le rythme.

Le programme a donc été réorienté une première fois en utilisant la fonction `delay(ms)` faisant une attente passive d'un nombre de millisecondes donné. Ainsi, le programme préparait toutes les notes qui devait être jouée, puis activait les actuateurs, attendait un certain temps (ici 9ms) et pour finir, désactivait les actuateurs.

```
while(currentNote.Tick == currentTick)
{
    PreparePush(currentNote);
    currentNote.Next();
}
ApplyPush();
delay(9); //delay de 9ms
ReleasePush();
```

Le principal problème de cette méthode est qu'il n'est pas possible d'ajuster le temps de frappe de chaque note indépendamment des autres.

Le code a donc été adapté de manière à corriger ce problème. Pour ce faire, la fonction `delay(ms)` a été remplacée par la fonction `micros()` qui retourne le nombre de microsecondes depuis le lancement du programme.

```
if(play)
    Push();
else
    startTime = micros(); //Actualise le temps si on est en pause
ReleasePush();
```

Nous avons donc le code ci-dessus dans la fonction `loop()` qui regarde à chaque passage si le programme est en train de jouer ou non. Si le programme est en train de jouer, on appelle la fonction `Push()` qui active les actuateurs et met à jour le tick courant (qui correspond à la progression dans la partition).

Même si nous sommes plus en train de jouer, le programme désactive au moment voulu les notes qui ont été poussées. Le temps de frappe des notes est géré avec les fonctions Push/ReleasePush.

7.6 Programme de conversion

7.6.1 Problème

Dans le but de pouvoir exploiter les fichiers MIDI, il est nécessaire d'avoir un outil permettant de les convertir sous un format exploitable pour le xylophone. Afin de pouvoir adapter les partitions pour qu'elles puissent être jouées par le xylophone, il faut aussi pouvoir éditer le fichier midi ou le fichier converti.

7.6.2 Démarche

Dans un premier temps, seule la partie de conversion midi à XML fut réalisée. L'édition des fichiers est faite en midi grâce au logiciel « MidiEditor ». Avant la partie conversion, une partie obligatoire a été réalisée, c'est la partie de chargement.

7.6.2.1 Chargement des fichiers MIDI

Pour faciliter le chargement des fichiers, une librairie open source a été installé. Celle-ci se nomme miditoolkit.

7.6.2.1.1 Chargement

Pour charger les partitions MIDI, la classe PartitionMidi implémente la fonction Load qui prend en paramètre un nom de fichier.

```
public void Load(string filename)
{
    Clear();
    Sequence sequence = new Sequence(filename);
    string[] tabString = filename.Split('\\');
    string title = tabString[tabString.Length - 1];
    Title = title.Split('.')[0];
    foreach (Track t in sequence)
    {
        foreach (MidiEvent midiEvent in t.Iterator())
        {
            int tick = midiEvent.AbsoluteTicks;
            if (midiEvent.MidiMessage.MessageType == MessageType.Channel)
            {
                ChannelMessage msg = ((ChannelMessage) midiEvent.MidiMessage);
                if (msg.Command == ChannelCommand.NoteOn)
                    GetMidiNoteOn(msg, tick);
            }
            else if (midiEvent.MidiMessage.MessageType == MessageType.Meta)
            {
                MetaMessage msg = ((MetaMessage) midiEvent.MidiMessage);
                if (msg.MetaType == MetaType.Tempo)
                    GetMidiTempo(msg, tick, sequence);
            }
        }
    }
    Tempo = Tempos[0].Value;
}
```

Cette fonction efface l'ancienne partition chargée s'il y en avait une. Puis charge le fichier midi dans une instance de Sequence : `Sequence sequence = new Sequence(filename);`

En parcourant cette séquence, on prend tous les messages « NoteOn » et « Tempo » qui sont les seuls messages qui nous intéressent pour le moment.

7.6.2.1.2 Notes du midi

Lors du chargement du fichier midi, on a vu qu'on récupérait les messages « NoteOn ». Lorsqu'un de ces messages apparaît, on enregistre la note dans notre PartitionMidi avec la fonction « GetMidiNoteOn ».

```
private void GetMidiNoteOn(ChannelMessage msg, int tick)
{
    //Data1 = id de la note.
    //Data2 = intensité de la note.
    Note tmpNote = NotesConvert.IdToNote(msg.Data1, msg.Data2, tick);
    //ajout des canaux innexistants
    if (Channels.Count - msg.MidiChannel <= 0)
    {
        int nbChannel = Channels.Count;
        for (int i = 0; i <= msg.MidiChannel - nbChannel; i++)
        {
            Channel ch = new Channel();
            ch.Name = Channels.Count.ToString();
            Channels.Add(ch);
        }
    }
    tmpNote.Name = Title + "_" + msg.MidiChannel.ToString() + "_" +
        (Channels[msg.MidiChannel].Notes.Count).ToString();
    Channels[msg.MidiChannel].Notes.Add(tmpNote);
}
```

La première étape est la conversion de la note :

```
Note tmpNote = NotesConvert.IdToNote(msg.Data1, msg.Data2, tick);
```

Ceci nous retourne une note avec l'ID qui est « msg.Data1 », l'intensité qui est « msg.Data2 » et le tick.

De cette note, si elle fait partie d'un canal qui n'existe pas encore (numéro de canal inexistant), on ajoute des canaux jusqu'au canal de la note en question.

Puis on enregistre la note dans le canal lui correspondant

```
Channels[msg.MidiChannel].Notes.Add(tmpNote);
```

7.6.2.1.3 Tempos du midi

Comme il est possible que le tempo d'une musique change au cours du temps, on récupère tous les tempos ainsi que leurs ticks.

```

private void GetMidiTempo(MetaMessage msg, int tick, Sequence sequence)
{
    int tempo = 0;

    // If this platform uses little endian byte order.
    if (BitConverter.IsLittleEndian)
    {
        int d = msg.Length - 1;

        // Pack tempo.
        for (int i = 0; i < msg.Length; i++)
        {
            tempo |= msg[d] << (8 * i);
            d--;
        }
    }
    // Else this platform uses big endian byte order.
    else
    {
        // Pack tempo.
        for (int i = 0; i < msg.Length; i++)
        {
            tempo |= msg[i] << (8 * i);
        }
    }
    Tempo tmpTempo = new Tempo(tick, (double)tempo / sequence.Division / 1000);
    tmpTempo.Name = "Tempo" + (Tempos.Count).ToString();
    Tempos.Add(tmpTempo);
}

```

Comme les tempos sont en plusieurs bytes, on concatène tous ces bytes dans un entier :

```
tempo |= msg[i] << (8 * i);
```

Puis on enregistre le tempo avec son tick et sa valeur convertie en milliseconde.

7.6.2.2 Conversion midi → XML

7.6.2.2.1 Choix des canaux à convertir

Comme les fichiers midi peuvent contenir plusieurs canaux représentant différents instruments, pour la conversion, il faut que l'utilisateur choisisse les canaux qu'il veut garder. Car jouer un orchestre sur un xylophone ne serait pas très musical.

Il y a donc une fenêtre qui s'ouvre lors de la conversion, demandant à l'utilisateur de sélectionner les canaux désirés.

A l'appel de la fenêtre, on envoie une liste de tous les canaux disponibles de la partition midi.

```

List<Channel> channels = new List<Channel>();
PartitionXylo partitionXylo = new PartitionXylo();
WindowChannelsSelect window = new WindowChannelsSelect();

for (int i = 0; i < CurrentPartition.Channels.Count; i++)
    channels.Add(CurrentPartition.Channels[i]);

if (window.Execute(channels) == true)

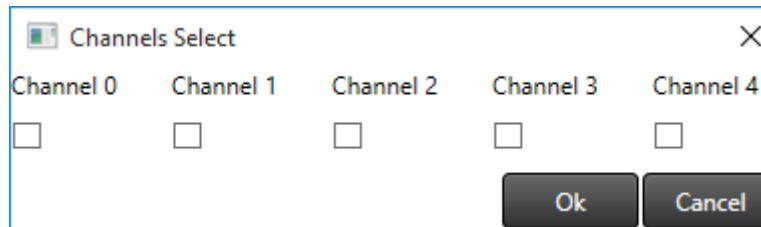
```

Puis la fenêtre génère dynamiquement des check box pour la sélection des canaux. Comme l'affectation d'un content au check box ne s'affiche pas, la fenêtre crée aussi un label pour chaque canal.

```
public bool? Execute(List<Channel> channels)
{
    for(int i=0; i< channels.Count; i++)
    {
        TextBlock txt = new TextBlock();
        txt.Name = "Label" + i.ToString();
        txt.Text = "Channel " + i.ToString();
        txt.Width = 80;
        txt.HorizontalAlignment = HorizontalAlignment.Center;
        StackPanelLabels.Children.Add(txt);

        CheckBox c = new CheckBox();
        c.Name = "CheckBox" + i.ToString();
        c.DataContext = channels[i];
        c.Width = 80;
        c.HorizontalAlignment = HorizontalAlignment.Center;
        StackPanelCheckBoxs.Children.Add(c);
    }
    this.Width = channels.Count * 80;
    this.ResizeMode = ResizeMode.NoResize;
}
```

On obtient alors ceci :



7.6.2.2.2 Conversion et sauvegarde

Une fois les canaux validés, si l'utilisateur a confirmé son choix, on convertit la partition midi en partition xylophone et on demande le nom pour la sauvegarde du fichier.

```
if (window.Execute(channels) == true)
{
    partitionXylo = CurrentPartition.ConvertToPartitionXylo(channels);

    SaveFileDialog dlg = new SaveFileDialog();
    dlg.InitialDirectory = FrameworkController.Instance.FileManagement.PathSaveFile;
    dlg.Filter = "xml files (*.xml)|*.xml";
    if (dlg.ShowDialog() == System.Windows.Forms.DialogResult.OK)
        partitionXylo.SaveToFile(dlg.FileName);
}
```

Pour la partie de conversion, la fonction demande les canaux à convertir puis pour chaque canal, elle ajoute les notes comprises dans les octaves du xylophone dans une seule et même liste : `List<Note> notes`.

```
public PartitionXylo ConvertToPartitionXylo(List<Channel> channels)
{
    PartitionXylo partitionXylo = new PartitionXylo();
    List<Note> notes = new List<Note>();
    foreach (Channel ch in channels)
        foreach (Note n in ch.Notes)
    {
        if ((n.Octave >= PartitionXylo.minOctave && n.Octave <= PartitionXylo.maxOctave)
            || (n.Octave == PartitionXylo.lastNoteOctave && n.High == PartitionXylo.lastNoteHigh))
            notes.Add(new Note(n));
    }
    notes.Sort(CompareNoteByTick);

    AdjustingMonoTempo(notes);

    foreach (Note n in notes)
        partitionXylo.Notes.Add(n);
    partitionXylo.Tempo = Tempo;
    partitionXylo.Title = Title;
    return partitionXylo;
}
```

Comme les notes proviennent de différents canaux et qu'elles sont triées par canal, il faut les retrier par « tick ». Pour cela, la classe « list » offre la fonction « Sort » qui prend en paramètre une fonction de tri.

Enfin, avant de ranger les notes dans la « PartitionXylo », il faut ajuster le tempo pour en obtenir un seul à l'aide de « AdjustingMonoTempo ». Cette fonction ajuste le tick des notes pour obtenir un seul et même tempo qui est le premier.

7.7 Remote debugging

7.7.1 Problème

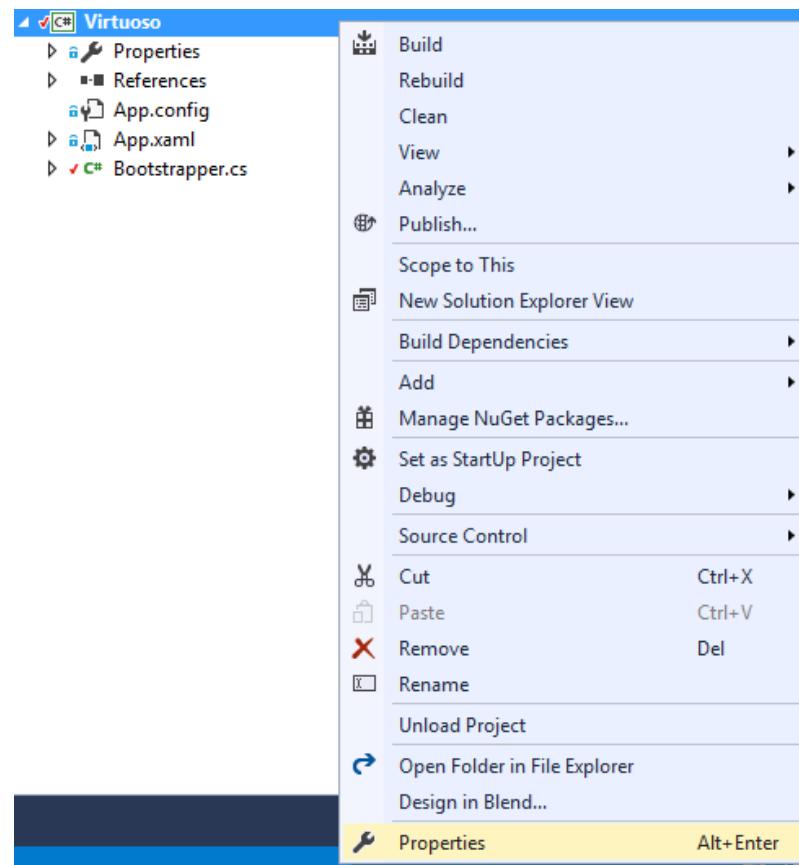
Pour éviter de devoir installer Visual Studio sur le NUC, nous avons choisi de partir sur le remote debugging. Et comme nous pouvions aussi développer directement sur un PC lambda avant de recevoir le NUC, cela simplifiait le développement.

7.7.2 Démarche

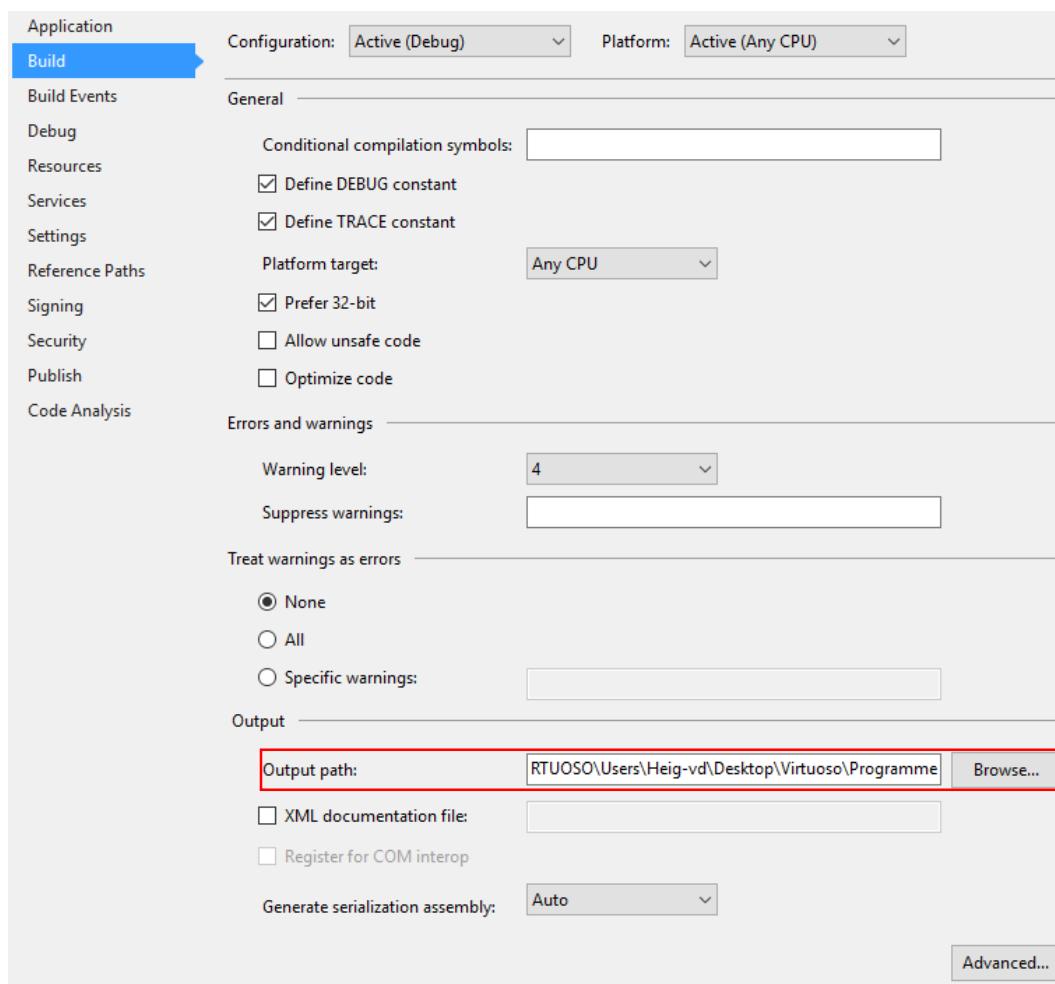
Après plusieurs recherches sur internet, nous avons installé l'outil « Visual Studio 2015 Remote Debugger » disponible sur le site Microsoft. Après plusieurs essais, nous n'obtenions aucun résultat. Nous n'arrivions pas à établir la connexion entre le pc et le NUC. Nous avons essayé le remote debugging sans authentification, à ce moment-là la connexion se faisait, mais ensuite il était impossible de lancer l'application.

Pour finir, nous avons trouvé le problème. Celui-ci était dû au fait que nous prenions un utilisateur sans mot de passe. Après l'application d'un mot de passe, un autre problème bloquait le déploiement de l'application. Nous n'utilisions pas de dossier partagé dans le NUC et par conséquent, le projet ne pouvait pas s'installer dans un dossier dans le NUC.

Finalement après ces deux problèmes qui ont pris un peu de temps à corriger, le remote debugging fonctionne. Et pour le paramétrier, il faut aller dans les propriétés du projet :

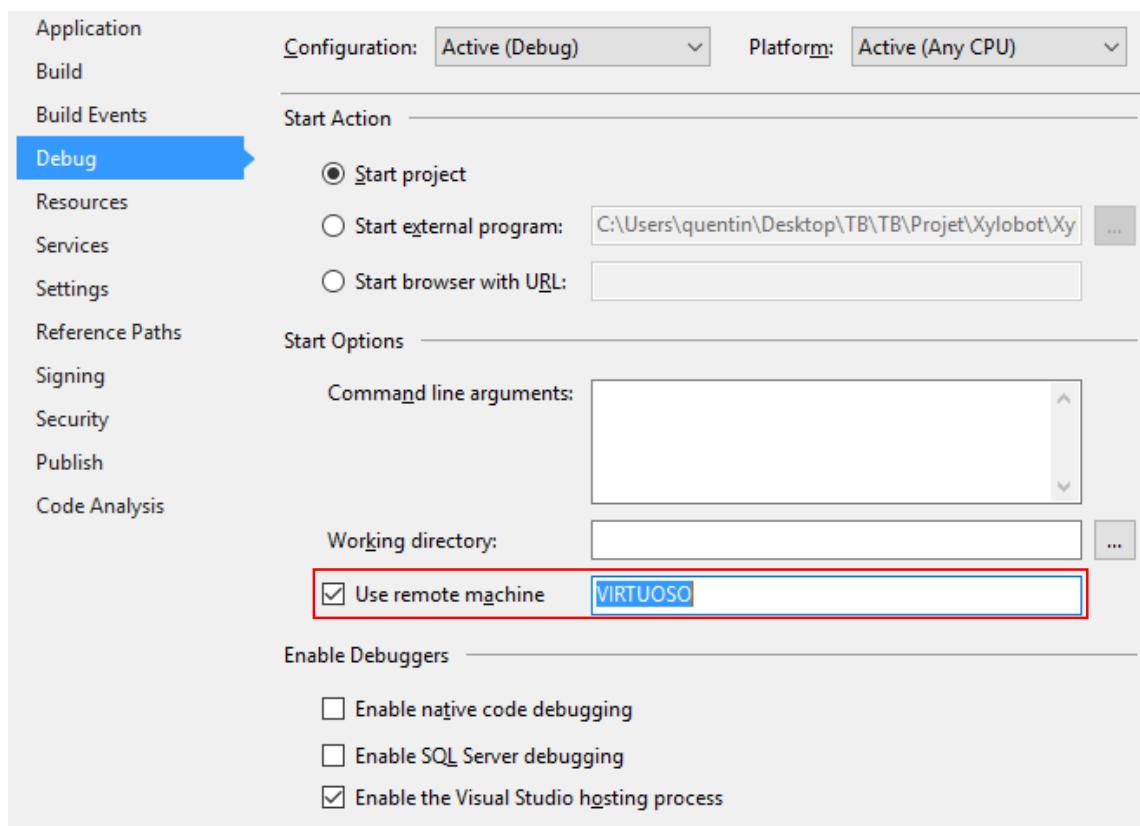


Et modifier l' « Output path » dans build :



Le chemin doit être celui du dossier partagé.

Il faut cocher la case « Use remote machine » et préciser le nom du NUC :



Et finalement, il ne faut pas oublier de lancer « Visual Studio 2015 Remote Debugger » sur le NUC avant de starte le projet.

7.7.3 Solution

Finalement le remote debugging fonctionne très bien, malgré les petits soucis rencontrés. Il aura quand même pris un peu plus de temps que prévu.

7.8 Programme Virtuoso

7.8.1 Squelette des UserControls

Le squelette de base de l'application utilise le composant NavigationControl de Concept HMI qui permet de switcher entre plusieurs UserControl. Le navigation control est implémenté dans la fenêtre MainWindow.xaml.

```
<Window x:Class="Framework.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:concept="http://www.objectis.ch/concept"
        xmlns:local="clr-namespace:Framework"
        Title="Virtuoso"
        WindowStyle="None" ResizeMode="NoResize"
        WindowStartupLocation="CenterScreen" WindowState="Maximized" BorderThickness="0">

    <Grid>
        <concept:NavigationControl NavigationBarDock="Top" Background="#FFFF0F0F0" Transition="Fade">
            <local:HomeView concept:NavigationControl.MenuTitle="Home"
                concept:NavigationControl.MenuImageSource="/Framework;component/Images/Home32x32.png"/>
            <local:SupervisionView concept:NavigationControl.MenuTitle="Supervision"
                concept:NavigationControl.MenuImageSource="/Framework;component/Images/Supervision32x32.png"
                DataContext="{Binding Path=SupervisionViewModel}"/>
            <local:CurrentPlaylistView concept:NavigationControl.MenuTitle="Current Playlist"
                concept:NavigationControl.MenuImageSource="/Framework;component/Images/CurrentPlayay64x32.png"
                DataContext="{Binding Path=CurrentPlaylistViewModel}"/>
            <local>EditPlaylistView concept:NavigationControl.MenuTitle="Playlist"
                concept:NavigationControl.MenuImageSource="/Framework;component/Images/Playlist32x32.png"
                DataContext="{Binding Path=EditPlaylistViewModel}"/>
            <local:SettingsView concept:NavigationControl.MenuTitle="Settings"
                concept:NavigationControl.MenuImageSource="/Framework;component/Images/Settings32x32.png"
                DataContext="{Binding Path=SettingsViewModel}"/>
        </concept:NavigationControl>
    </Grid>
</Window>
```

Ainsi, nous avons nos cinq vues correspondant aux cinq UserControl implémentés. Chaque UserConrol à son propre DataContext provenant du DataContext de la MainWindow.

```
public class MainViewModel : BaseViewModel
{
    #region Constructor

    public MainViewModel()
    {
        CurrentPlaylistViewModel = new CurrentPlaylistViewModel();
        EditPlaylistViewModel = new EditPlaylistViewModel();
        SupervisionViewModel = new SupervisionViewModel();
        SettingsViewModel = new SettingsViewModel();
    }

    #endregion

    public CurrentPlaylistViewModel CurrentPlaylistViewModel { get; set; }
    public EditPlaylistViewModel EditPlaylistViewModel { get; private set; }
    public SupervisionViewModel SupervisionViewModel { get; private set; }
    public SettingsViewModel SettingsViewModel { get; private set; }
}
```

Remarque : le UserControl HomeView n'a pas de Model, car il n'en a pas besoin.

Ces modèles qui sont passés dans les DataContexts récupèrent les propriétés de FrameworkController qui est un singleton.

```
public class CurrentPlaylistViewModel : BaseViewModel
{
    public Playlist Playlist { get { return FrameworkController.Instance.Playlist; } }
    public Sequencer Sequencer { get { return FrameworkController.Instance.Sequencer; } }
}
```

(Les dépendances des DataContexts sont décrites dans les diagrammes UML.)

FrameworkController es là pour gérer les variables appartenant à l'application. Dans les diagrammes UML, il remplace la classe VirtuosoApplication.

```
public sealed class FrameworkController
{
    #region Singleton

    Load / Unload

    #region Public Properties

    public Playlist Playlist { get; private set; }
    public Settings Settings { get; private set; }
    public Sequencer Sequencer { get; private set; }
    public VirtuosoWebServer WebServer { get; private set; }

    #endregion

    Load / Save Configuration
}
```

7.8.2 UML final du Virtuoso

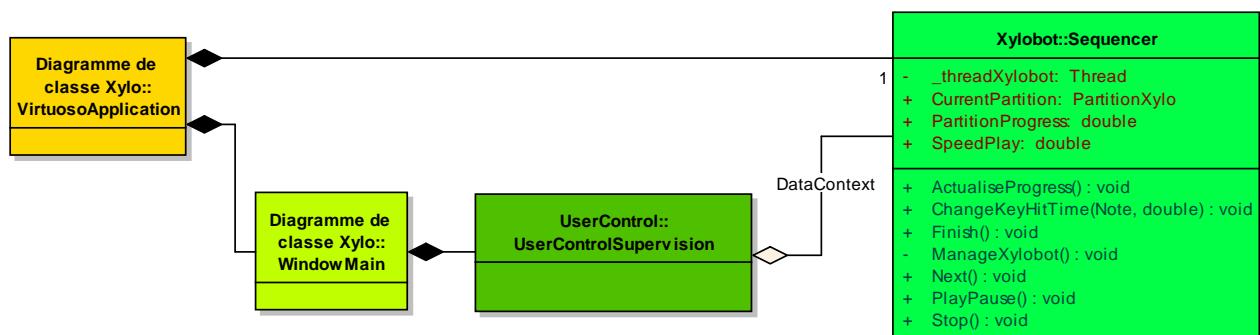
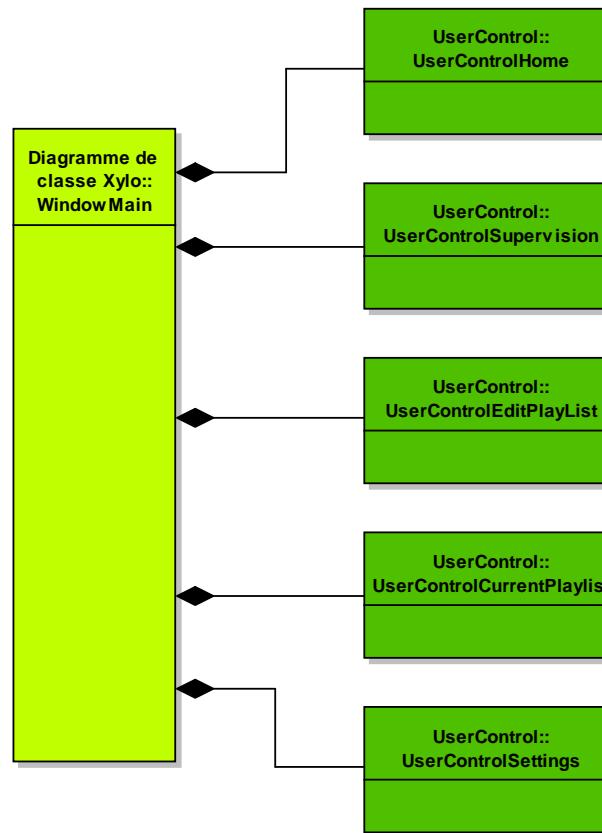
7.8.3 Problème

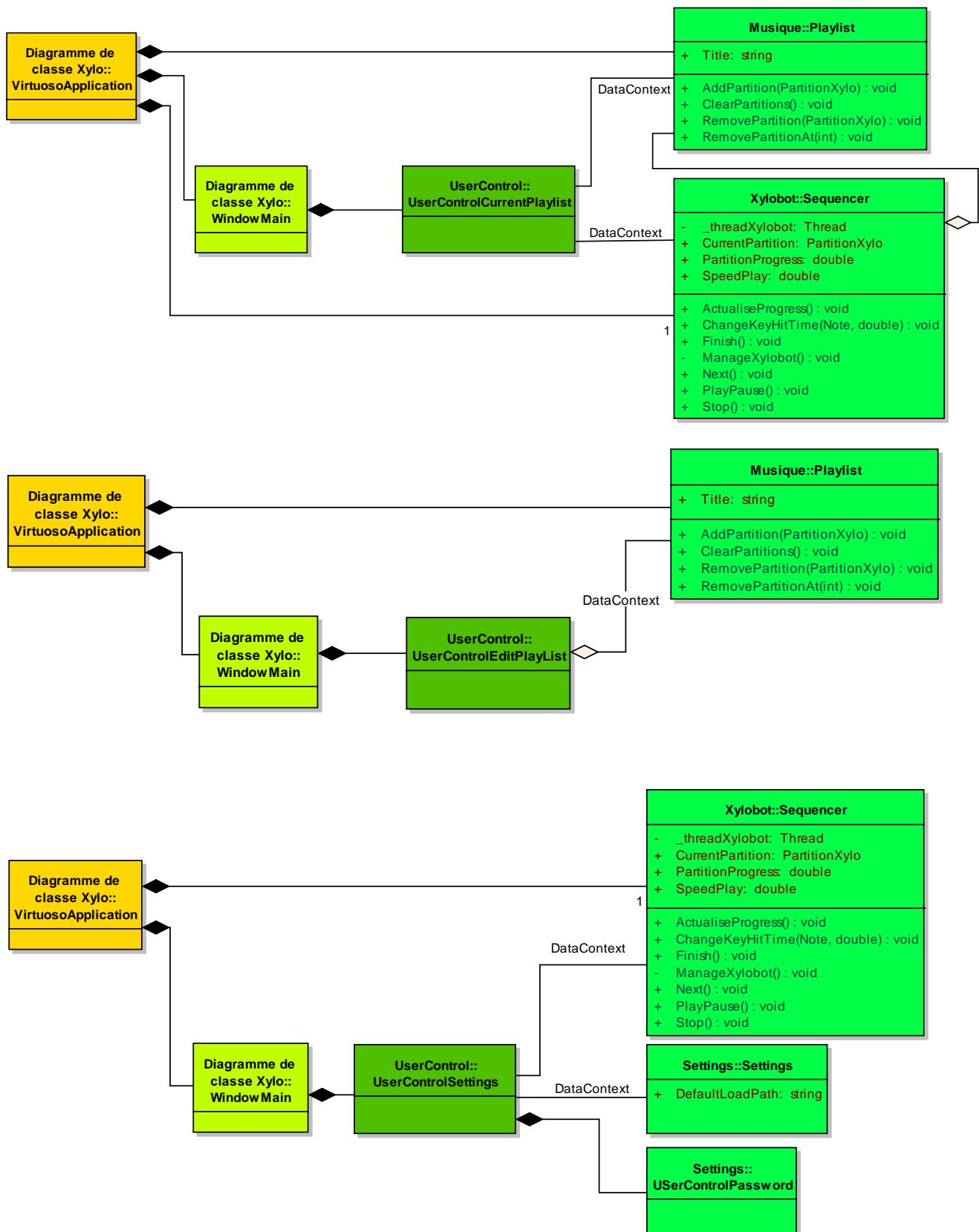
À la fin du développement de l'application, le diagramme UML de la pré-étude n'est plus correct, car entre temps plusieurs modifications et façons de penser ont été changées. Nous avons donc conçu un nouvel UML basé sur le code. Ainsi, nous n'aurons pas besoin de mettre tout le code dans le rapport, mais seulement l'UML avec des explications.

7.8.4 Démarche

Pour créer l'UML, nous avons pris chacune des classes importantes et nous les avons mises dans l'UML avec certaines de leurs propriétés et fonctions. Ainsi nous avons un UML simple à lire avec les informations essentielles.

Les premiers diagrammes concernent les UserControl décrits dans le squelette de l'application.

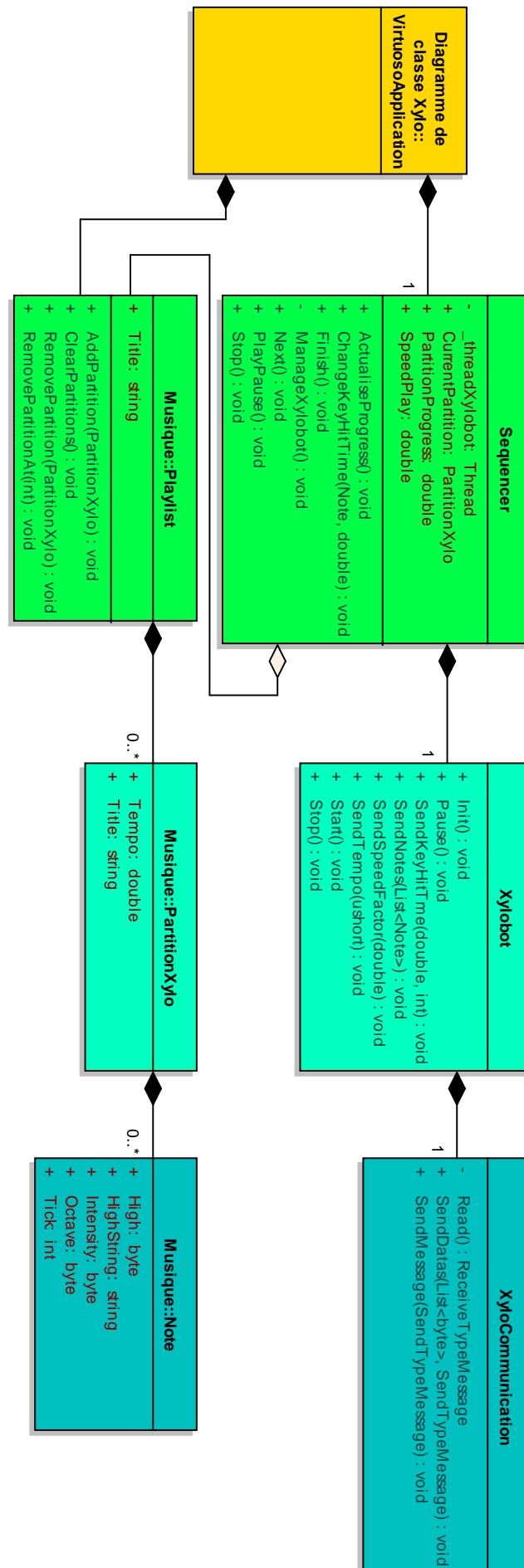




À l'aide des UML ci-dessus, nous pouvons facilement voir les dépendances que les vues ont avec les classes du code.

Une particularité pour le UserControlSettings est l'apparition d'un UserControlPassword, car il nous a été demandé de mettre un code pour accéder à la partie settings.

Il vient maintenant le corps du code caché derrière tout ce visuel. C'est la partie qui montre bien les dépendances entre les classes et comment tout se goupille.



Concrètement, ce diagramme n'a pas beaucoup changé par rapport à la pré-étude. La grosse différence est que le « Xylobot » n'est plus une composition de l'application, mais du « Sequencer ». Ce changement a été fait, car il simplifiait le code notamment pour les DataContexts des UserControl.

Mis à part cela, il n'y a pas de grand changement par rapport au premier UML. Les fonctions ont changé, car au départ nous ne savions pas exactement quelles fonctions allaient être implémentées. Et n'ayant jamais fait d'UML conséquent auparavant, il nous était difficile de tout prévoir juste.

7.8.5 Solution

Les nouveaux UML ont naturellement changé, mais malgré cela, nous voyons que les UML faits lors de la pré-études sont proches de la réalité. En conclusion, les nouveaux diagrammes sont plutôt simples et montrent que l'application a été implémentée correctement.

7.8.6 Splash screen et pop-up

7.8.6.1 Problème

Pour que toute l'application soit dans le même style, nous avons dû changer le splash screen généré par Concept HMI. Et il a été nécessaire d'avoir des fenêtres visuelles donnant un retour à l'utilisateur pour certaines des actions qu'il effectue.

7.8.6.2 Démarche

Comme l'application est sous le style « flat », et que les fenêtres doivent être des sortes de pop-up, nous avons créé des fenêtres qui sont des simples rectangles colorés.

Pour ce faire, nous avons ajouté au projet un nouveau fichier « window.xaml ». Dans celui-ci nous avons modifié les paramètres de la fenêtre comme suit :

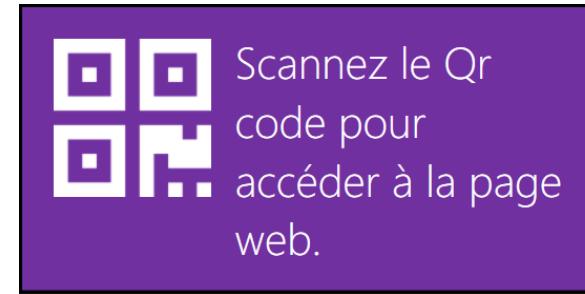
```
<Window x:Class="Framework.MessageBoxAutoClosed"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:local="clr-namespace:Framework"
    mc:Ignorable="d"
    Title="WindowMessageBoxAutoClosed" Loaded="Window_Loaded"
    ResizeMode="NoResize" WindowStyle="None" WindowStartupLocation="CenterScreen"
    SizeToContent="WidthAndHeight">
```

Permettant ainsi de supprimer les styles des « window » et d'avoir des fenêtres adaptées à leur contenu.

Ensuite, par code, où nous avons besoin d'appeler les fenêtres, nous choisissons leurs couleurs et/ou leur contenu (cela varie selon l'implémentation qu'il y a dans la fenêtre).

```
WindowMessageBoxInformation w = new WindowMessageBoxInformation();
w.Text = "Connectez-vous au wifi \"Virtuoso\" et accédez à la page web.";
w.Background = UserControlHomeWifi.Background;
w.ImageSource = new BitmapImage(new Uri(@"/Framework;component/Images/Wifi128x128.png",
    UriKind.RelativeOrAbsolute));
w.ShowDialog();
```

Ainsi nous obtenons des fenêtres dans le style suivant :



Pour la partie splash screen, nous avons pris rendez-vous chez Objectis et nous avons pu facilement le changer. Pour cela, nous avons créé une nouvelle « window » qui dans notre cas est une simple image.

```
<Window x:Class="Framework.SpalshScreen"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:local="clr-namespace:Framework"
    mc:Ignorable="d"
    Title="SpalshScreen" SizeToContent="WidthAndHeight"
    ResizeMode="NoResize" WindowStyle="None" WindowStartupLocation="CenterScreen">
    <Grid>
        <Image Source="/Framework;component/Images/Xylophone2.jpg" Width="600"
            SnapsToDevicePixels="True"/>
    </Grid>
</Window>
```

Nous avons ensuite créé un constructeur à notre classe bootstrapper qui donne la fenêtre à la classe bootstrapper de Concept HMI.

```
public class Bootstrapper : ConceptBootstrapper
{
    public Bootstrapper(Window splashScreen) : base(splashScreen)
    {

    }
}
```

Et finalement, lorsque nous créons notre instance du bootstrapper, nous lui donnons notre splash screen en paramètre.

```
public partial class App : Application
{
    protected override void OnStartup(StartupEventArgs e)
    {
        base.OnStartup(e);
        Bootstrapper boot = new Bootstrapper(new SpalshScreen());
        boot.Run();
    }
}
```

Ce qui nous affiche notre image définie dans la fenêtre du splash screen au démarrage de l'application :



7.8.7 Défilement des notes de la partition courante

7.8.7.1 Problème

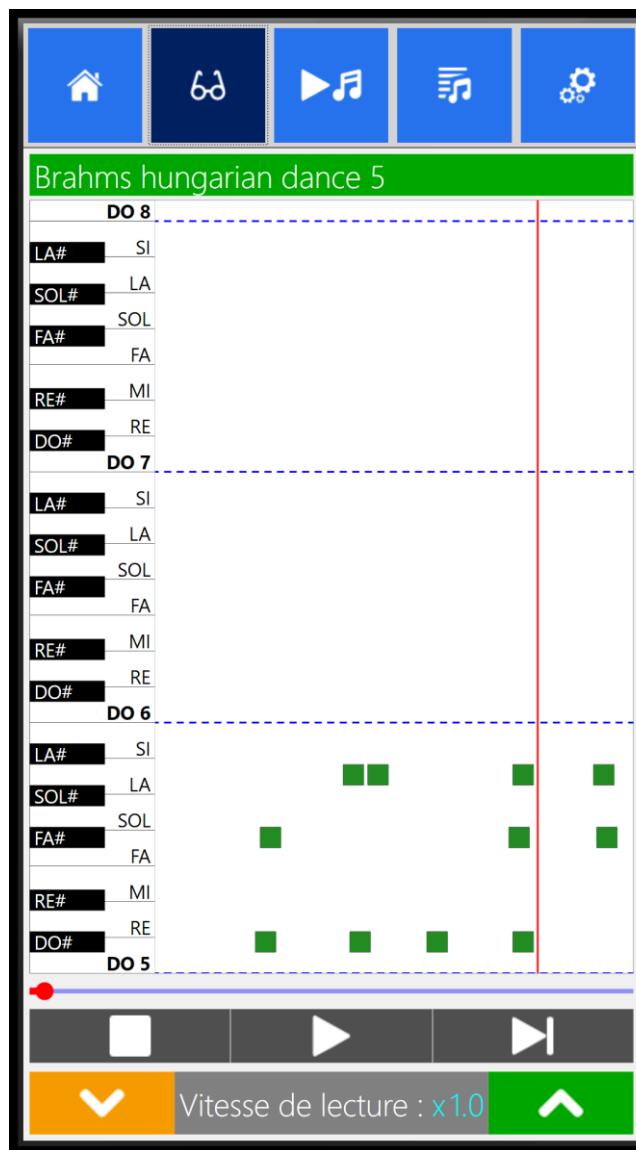
Dans le but de voir ce qu'il se passe lors de la lecture d'une partition, un composant doit afficher les notes en direct sur le Virtuoso. Cette partie permettra de voir les accords musicaux qui viennent de passer.

7.8.7.2 Démarche

La réalisation de cet affichage s'est faite dans un UserControl qui prend le séquenceur comme DataContext.

```
<local:UserControlShowPartition KeyWidth="140" DataContext="{Binding Path=Sequencer}"  
OctaveNumber="3"/>
```

Donnant l'interface :



De base le composant n'affiche qu'une zone blanche qui sont des « ScrollViewer » dans lesquels il y a un « Canvas ».

```
<ScrollViewer x:Name="ScrollView(Keys)" HorizontalScrollBarVisibility="Disabled" ScrollChanged="ScrollChanged"
    VerticalScrollBarVisibility="Hidden" Grid.Row="1">
    <Canvas x:Name="Canvas(Keys)" Width="100"/>
</ScrollViewer>
<ScrollViewer x:Name="ScrollView(Notes)" HorizontalScrollBarVisibility="Hidden"
    VerticalScrollBarVisibility="Disabled" Grid.Row="1" Grid.Column="1">
    <Canvas x:Name="Canvas(Notes)"></Canvas>
</ScrollViewer>
```

Les « ScrollViewer » nous offrent la possibilité de scroller les « Canvas » qui les composent. Grâce à ce mécanisme, il est possible de placer toutes les notes dans le « Canvas » prévu à cet effet et de le faire défiler avec les « ScrollViewer ».

Afin d'afficher les notes, lors d'un changement de la partition courante dans le séquenceur, nous générerons un rectangle par note placé en fonction du tick et de la hauteur de celle-ci moyennant des facteurs pour un meilleur affichage. Pour prendre en compte un changement de partition courante, nous assignons le délégué suivant au DataContext :

```
((Sequencer)DataContext).PropertyChanged += DataContextPropertyChangedEventHandler;
```

Dans laquelle nous regardons si la propriété du séquenceur qui a changé est bien notre partition courante pour afficher les notes. Un autre cas est introduit dedans, celui de la progression. S'il la progression a changé, nous scrollons la partition afin de la suivre lorsqu'elle est jouée.

```
private void DataContextPropertyChanged(object sender, PropertyChangedEventArgs e)
{
    if (e.PropertyName == Sequencer.CurrentPartitionPropertyName)
        ShowPartition();
    else if (e.PropertyName == Sequencer.PartitionProgressPropertyName)
        ScrollPartition();
}
```

La fonction ShowPartition efface l'ancienne partition, puis génère les notes (à l'aide de rectangle) pour la nouvelle.

```
public void ShowPartition()
{
    ReleaseDrawPartition();

    if ((DataContext as Sequencer).CurrentPartition != null)
    {
        int maxTick = 0;
        foreach (Note note in (DataContext as Sequencer).CurrentPartition.Notes)
        {
            Rectangle rect = new Rectangle();
            rect.Width = rectangleNoteSize;
            rect.Height = rectangleNoteSize;
            rect.Fill = Brushes.ForestGreen;
            rect.Stroke = Brushes.Black;
            rect.StrokeThickness = 0.3;
            rect.DataContext = note;
            rect.Visibility = Visibility.Visible;
            CanvasNotes.Children.Add(rect);
            Canvas.SetLeft(rect, note.Tick / factorSpaceNote + LineRed.X1);
            Canvas.SetBottom(rect, ((note.Octave - Xylobot.startOctaveXylophone)
                * Xylobot.octaveSize + note.High) * rectangleNoteSize);

            maxTick = maxTick < note.Tick ? note.Tick : maxTick;
        }
        CanvasNotes.Width = maxTick / factorSpaceNote + offsetScroll;
    }
}
```

La position en X des notes (Canvas.SetLeft) est corrigée avec LineRed.X1 afin que les notes jouées sur le xylophone soient affichées sous la ligne rouge.

Pour la fonction ScrollPartition, elle intervient lorsque la progression change. Elle permet de suivre l'exécution de la partition.

```
public void ScrollPartition()
{
    ScrollViewerNotes.ScrollToHorizontalOffset(
        ((Sequencer)DataContext).PartitionProgress * CanvasNotes.ActualWidth);
}
```

Comme « PartitionProgress » varie de 0 à 1, on scroll la vue de 0 à la largeur de notre Canvas.

7.9 Visuels des interfaces

7.9.1 Problème

Comme la principale fonction du Virtuoso sera son exposition aux portes ouvertes de la Heig-vd, il faut que l'interface graphique soit belle et attractive. Elle doit aussi être simple pour les utilisateurs afin qu'il ne passe pas 30 minutes à essayer de jouer une musique.

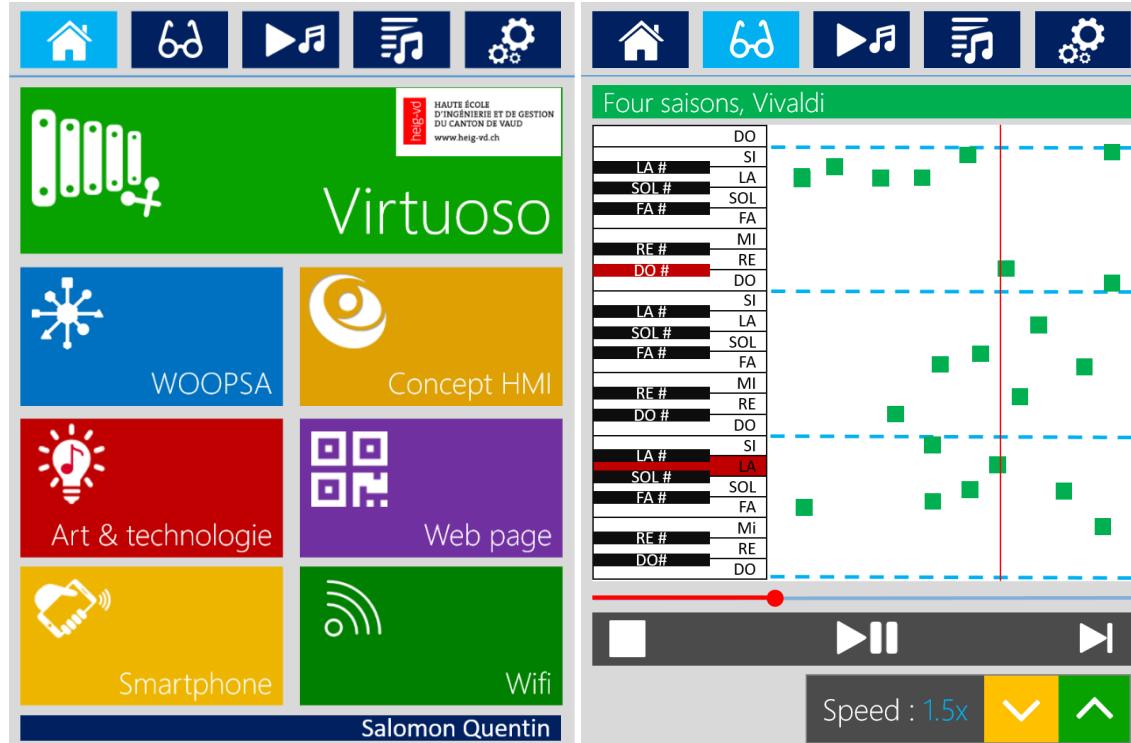
7.9.2 Démarche

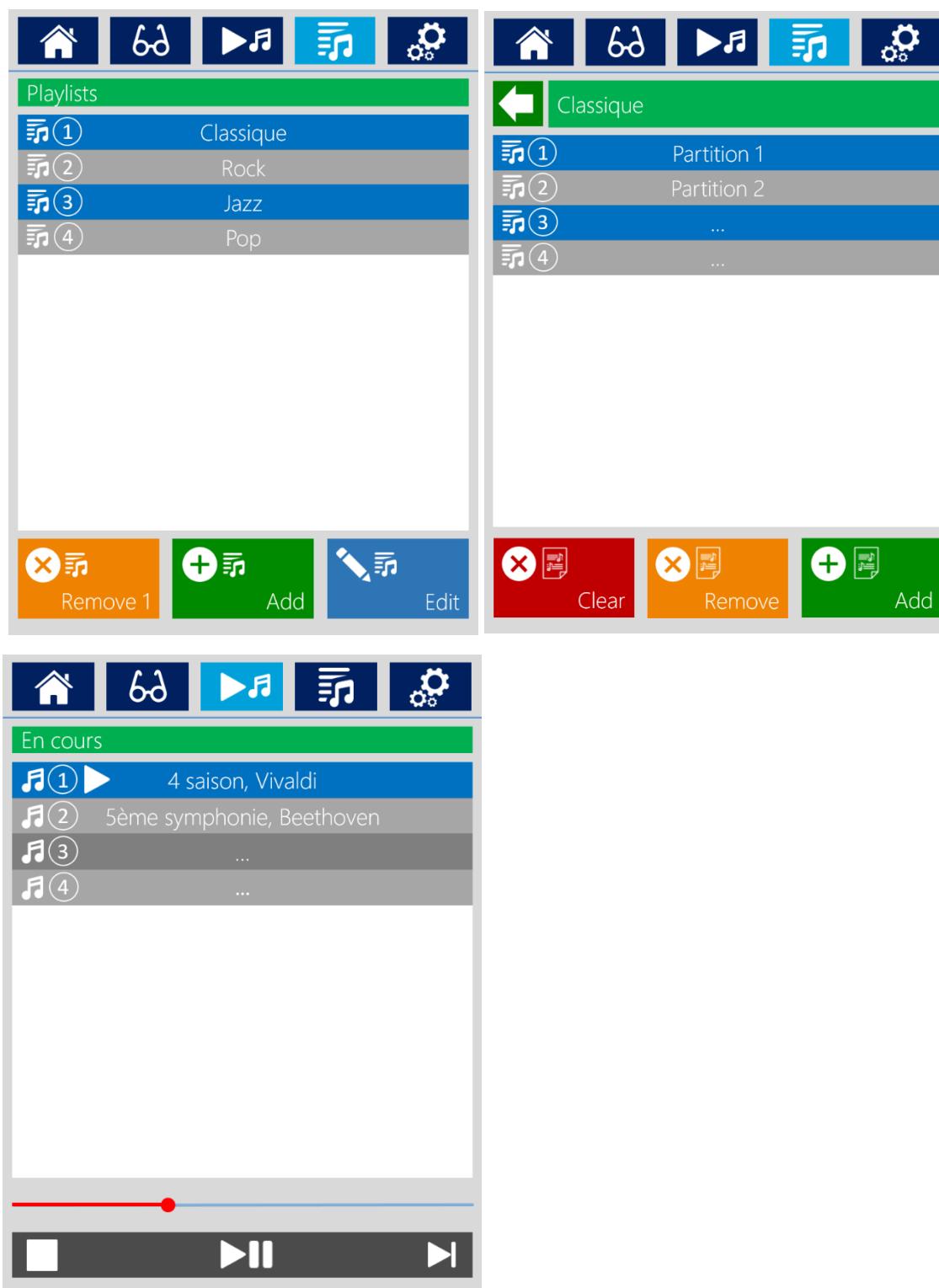
En premier lieu, des prototypes ont été faits pour avoir une idée de la direction dans laquelle partir. Puis on est venu se calquer sur ces prototypes pour créer l'interface en amenant quelques modifications.

7.9.3 Prototypes

Le fait de voir les prototypes permet aussi de faire de l'ordre dans les idées d'implémentations et de voir si l'interface sera « UserFriendly » ou non.

L'idée de l'interface est de partir sur un design plutôt flat dans le style Windows 8.





7.9.4 Possibilités offertes par concept

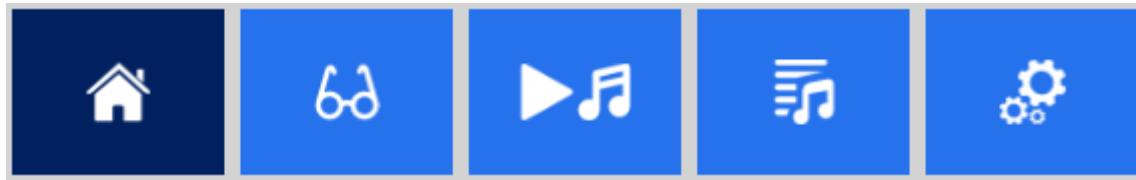
Après un rendez-vous chez Objectis pour voir ce qu'il était possible de faire, nous avons vu que nos prototypes étaient réalisables. Malgré cela, il n'a pas été aisément de faire et d'appliquer des styles.

Pour commencer, le composant principal de concept est le `NavigationControl`. C'est le seul composant WPF de concept qui a été utilisé. Le fichier de style nous a été fourni par concept et

modifié par nos soins pour correspondre à nos attentes. La seule partie du fichier qui a été modifiée est le « DataTemplate ».

```
<DataTemplate x:Key="ItemTemplate">
    <DockPanel x:Name="DockPanelButton"
        HorizontalAlignment="Center"
        Background="{DynamicResource ColorsVirtuosoBlue}"
        Width="130" Height="100" Margin="5,5">
        <Image Source="{Binding Path=(concept:NavigationView.MenuImageSource)}"
            Visibility="{Binding Source, RelativeSource={RelativeSource self},
            Converter={concept:NullableToVisibilityConverter}}"
            Width="70" Height="40" VerticalAlignment="Center" HorizontalAlignment="Center"/>
    </DockPanel>
    <DataTemplate.Triggers>
        <DataTrigger Binding="{Binding IsSelected, RelativeSource={RelativeSource Mode=FindAncestor,
            AncestorType={x:Type TabItem}}}" Value="True">
            <Setter TargetName="DockPanelButton" Property="Background"
                Value="{DynamicResource ColorsVirtuosoDarkBlue}"/>
        </DataTrigger>
    </DataTemplate.Triggers>
</DataTemplate>
```

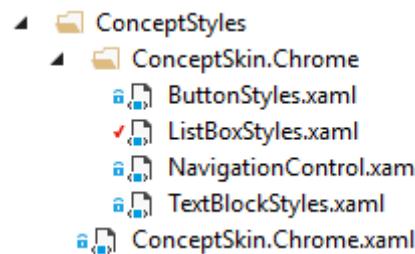
Les couleurs ont été ajustées à l'aide des propriétés « Background » et le « TextBlock » permettant d'afficher le nom sous les images a été retiré. Ce qui nous donne :



Le résultat est quasiment identique au prototype.

7.9.5 Spécificité pour appliquer les styles

Avec concept, il a fallu faire quelque modification pour pouvoir appliquer les styles. Notamment mettre tous les styles créés dans un dossier nommé ConceptStyles/ConceptSkin.Chrome



Et les appeler dans le fichier ConceptSkin.Chrome.xaml comme décrit ici :

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="clr-namespace:Framework">
    <ResourceDictionary.MergedDictionaries>
        <ResourceDictionary Source="ConceptSkin.Chrome/NavigationView.xaml"/>
        <ResourceDictionary Source="ConceptSkin.Chrome/TextBlockStyles.xaml"/>
        <ResourceDictionary Source="ConceptSkin.Chrome/ButtonStyles.xaml"/>
        <ResourceDictionary Source="ConceptSkin.Chrome/ListBoxStyles.xaml"/>
    </ResourceDictionary.MergedDictionaries>
</ResourceDictionary>
```

Ceci a été fait pour que concept prenne en compte les styles créés sans devoir tout remodifier.

7.9.6 Styles créés pour le Virtuoso

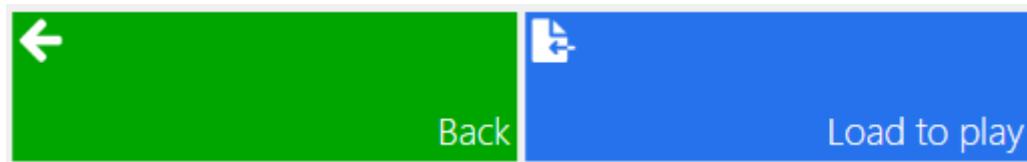
Pour « contrer » les styles de concept, nous avons créé plusieurs styles pour les composants voulus.

7.9.6.1 Style des « Buttons »

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="clr-namespace:Framework">

    <Style x:Key="ButtonStyle" TargetType="Button">
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="Button">
                    <Border CornerRadius="0" Background="{TemplateBinding Background}">
                        <ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center"/>
                    </Border>
                </ControlTemplate>
            </Setter.Value>
        </Setter>
    </Style>
</ResourceDictionary>
```

Le style est simple, il permet juste de mettre un « Border » dans les boutons. Permettant ainsi d'avoir des boutons sans angle et avec un style « flat ».



7.9.6.2 Style des « TextBlocks »

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="clr-namespace:Framework">

    <Style x:Key="TextBlockTitleStyle" TargetType="TextBlock">
        <Setter Property="FontFamily" Value="Segoe UI Light"/>
        <Setter Property="FontSize" Value="35"/>
        <Setter Property="Foreground" Value="White"/>
    </Style>

    <Style x:Key="TextBlockNormalStyle" TargetType="TextBlock">
        <Setter Property="FontFamily" Value="Segoe UI Light"/>
        <Setter Property="FontSize" Value="28"/>
        <Setter Property="Foreground" Value="White"/>
    </Style>
</ResourceDictionary>
```

Comme pour chaque application, il faut plusieurs styles de texte différents, c'est pourquoi nous en retrouvons deux ici. Le premier est pour les titres et le deuxième pour le texte « normal » (autrement le reste des textes). Les deux implémentent la même police et la même couleur pour suivre le style des prototypes.

7.9.6.3 Style des « ListBox »

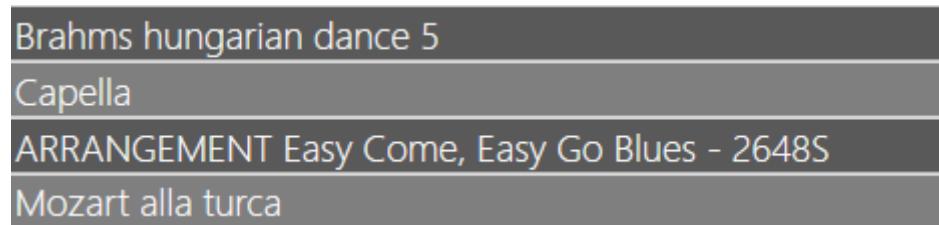
7.9.6.3.1 Alternation des couleurs des lignes

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="clr-namespace:Framework">

    <Style TargetType="{x:Type ListBoxItem}">
        <Style.Triggers>
            <Trigger Property="ItemsControl.AlternationIndex" Value="0">
                <Setter Property="Background" Value="#FF595959"></Setter>
            </Trigger>
            <Trigger Property="ItemsControl.AlternationIndex" Value="1">
                <Setter Property="Background" Value="#FF7F7F7F"></Setter>
            </Trigger>
        </Style.Triggers>
    </Style>

</ResourceDictionary>
```

Ce style donne aux ListBoxes un style moins macabre en alternant la couleur des lignes. Ici les lignes sont grise-foncé et grise-claire.



Remarque : Comme le style est défini pour une cible (`TargetType="{x:Type ListBoxItem}"`), lors de la création d'un composant ListBox, il s'applique tout seul. Nous avons fait ce choix, car toutes les ListBoxes de l'application ont cette alternation de couleurs.

7.9.6.3.2 Numéros de ligne et images

Il a fallu ensuite ajouter les images et les numéros de lignes. Ces modifications ont été implémentées directement dans le WPF ou se situe les ListBoxes, car il fallait un « convertisseur » pour le numéro de la ligne et que l'image devait pouvoir se changer pour la ListBoxes des Playlists (ces deux points se font en C# et les fichiers XAML pour les styles n'ont pas de C#).

Nous avons donc :

```
<UserControl.Resources>
    <local:IndexConverter x:Key="IndexConverter" />
    <DataTemplate x:Key="DataTemplateListBox">
        <Grid x:Name="GridListBoxItem" HorizontalAlignment="Stretch">
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="auto" />
                <ColumnDefinition Width="auto" />
                <ColumnDefinition Width="*" />
            </Grid.ColumnDefinitions>

            <Image Width="32" Height="32" Source="{Binding RelativeSource={RelativeSource AncestorType={x:Type UserControl}}, Path=ListBoxImageSource}" Margin="5"/>
            <Ellipse Stroke="White" StrokeThickness="2" Fill="Transparent" Width="32" Height="32" Grid.Column="1" HorizontalAlignment="Center" VerticalAlignment="Center" Margin="5"/>
            <TextBlock Text="{Binding RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type ListBoxItem}}, Converter={StaticResource IndexConverter}}"
                Foreground="White" FontSize="25" Grid.Column="1" Margin="3,3,3,5"
                HorizontalAlignment="Center" VerticalAlignment="Center"/>
            <TextBlock Text="{Binding Title}" Style="{DynamicResource TextBlockNormalStyle}" Grid.Column="2"
                Margin="30,5,5,5"/>
        </Grid>
    </DataTemplate>

```

Ainsi, en appliquant `DataTemplateListBox` à la propriété `ItemTemplate` des ListBoxes :

```
ItemTemplate="{DynamicResource DataTemplateListBox}"
```

Nous avons des lignes composées d'une image (qui est la même pour toutes les lignes), un numéro de ligne entouré d'un cercle et le titre des items.

L'image est binder sur une propriété :

```
public ImageSource ListBoxImageSource
{
    get { return _listBoxImageSource; }
    set
    {
        _listBoxImageSource = value;
        OnPropertyChanged("ListBoxImageSourceProperty");
    }
}
private ImageSource _listBoxImageSource;
```

Permettant pour la partie de gestion des playlists de modifier l'image des lignes.

Nous avons ensuite le numéro de ligne qui est une conversion de l'index de la ligne en string :

```
public class IndexConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
    {
        ListBoxItem item = (ListBoxItem)value;
        ListBox ListBox = ItemsControl.ItemsControlFromItemContainer(item) as ListBox;
        int index = ListBox.ItemContainerGenerator.IndexFromContainer(item);
        return index.ToString();
    }

    public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}
```

7.9.6.3.3 Selection et « Mouse Over »

Afin de changer le style de l'item sélectionné et la couleur de l'objet sur lequel on passe avec la souris, dans le `DataTemplateListBox`, nous avons ajouté des triggers.

```

<DataTemplate.Triggers>
    <DataTrigger Binding="{Binding RelativeSource=
        {RelativeSource Mode=FindAncestor, AncestorType=
        {x:Type ListBoxItem}}},Path=IsSelected}" Value="True">
        <Setter TargetName="GridListBoxItem" Property="Background"
            Value="{DynamicResource ColorsVirtuosoDarkBlue}"/>
    </DataTrigger>
    <DataTrigger Binding="{Binding RelativeSource=
        {RelativeSource Mode=FindAncestor, AncestorType=
        {x:Type ListBoxItem}}},Path=IsMouseOver}" Value="True">
        <Setter TargetName="GridListBoxItem" Property="Background"
            Value="{DynamicResource ColorsVirtuosoMiddleBlue}"/>
    </DataTrigger>
</DataTemplate.Triggers>
</DataTemplate>
</UserControl.Resources>

```

Le premier **DataTrigger** change la couleur de l'item sélectionné en bleu foncé. Et le deuxième change la couleur de l'item sur lequel la souris passe.

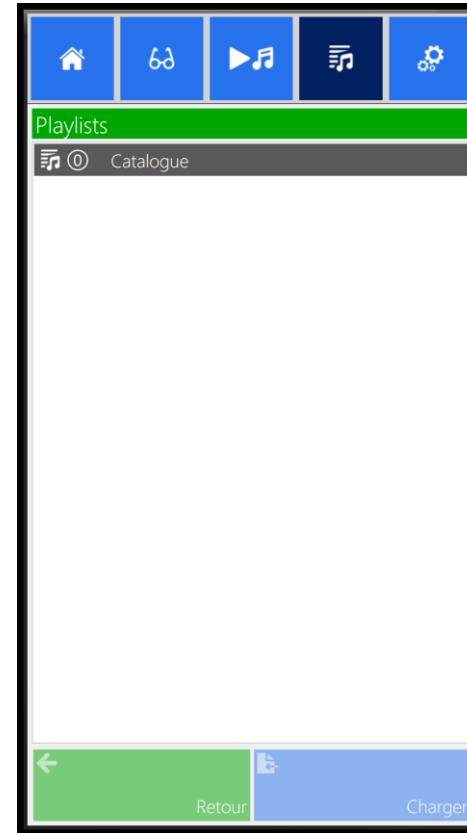
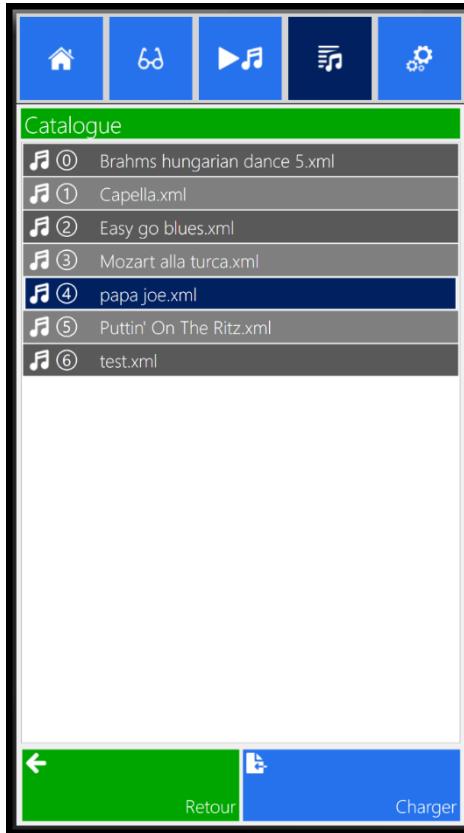
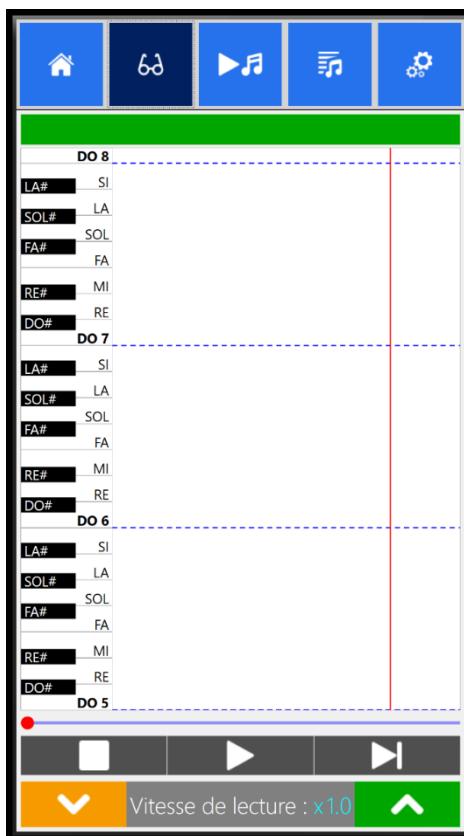
Remarque : comme l'application finale est sur un écran tactile, le trigger pour le passage de la souris n'a pas vraiment lieu d'être mis à part dans le cas où l'on glisserait le doigt sur l'écran.

7.9.7 Visuel final

Au final, le visuel de l'application correspond bien au prototype hors mis quelques modifications apportées pour le principe d'utilisation.

Par conséquent, nous sommes satisfaits du résultat obtenu.





7.10 Serveur WEB

7.10.1 ConceptWeb

7.10.1.1 Qu'est-ce que c'est ?

ConceptWeb est le travail de bachelor d'un collègue de la Heig-vd. C'est un framework qui a pour but la simplification du développement de site Web. Il est développé grâce à Woopsa.

7.10.1.2 Utilité

Pour la partie Web du Virtuoso, il a une grande utilité, car il nous évite pas mal d'heures d'apprentissage pour le développement Web.

Il a aussi l'avantage d'avoir un style plus orienté pour les smartphones, ce qui nous est bien pratique, car le serveur Web du Virtuoso cible les smartphones.

7.10.2 Problème

Dans le cahier des charges, il a été défini la création d'un serveur Web. Permettant à une tierce personne d'interagir avec le Virtuoso en voyant la partition en cours de lecture ainsi que sa progression et de faire une demande pour l'ajout d'une musique.

7.10.3 Démarche

Pour la création du serveur Web, nous avons décidé d'utiliser ConceptWeb afin de gagner du temps. Comme il n'implémentait pas encore la gestion pour les listes, nous avons commencé par l'affichage de la partition courante et sa progression. Puis nous nous sommes attaqués à la partie des listes en affichant d'abord la liste de lecture du Virtuoso puis en affichant une liste pour la requête d'ajout.

7.10.3.1 Implémentation du serveur en C#

Grâce à ConceptWeb, l'implémentation au niveau taille du code est assez simple.

Nous avons, tout d'abord, une classe VirtuosoWebServer :

```
public class VirtuosoWebServer
{
    ConceptWebServer Server { get; set; }
    private VirutosoWebController _virutosoWebController;

    public VirtuosoWebServer(Sequencer sequencer, Playlist principalPlaylist)
    {
        _virutosoWebController = new VirutosoWebController(sequencer, principalPlaylist);

        Server = new ConceptWebServer();
        Server.RegisterWebApp("Virtuoso",
            new WebMaterialApp(new CustomVirutosoWebView(_virutosoWebController)));

        Server.WoopsaServer.WebServer.Routes.Add("/", HttpMethod.GET,
            new RouteHandlerRedirect("Web/Virtuoso", WoopsaRedirection.Temporary));
    }

    public void Close()
    {
        Server.Dispose();
    }
}
```

Cette classe prend en paramètres, pour son constructeur, les objets utiles au serveur. Dans notre cas la playlist principale permettant de savoir la liste des partitions qui vont être jouées ainsi que le séquenceur qui est là pour connaître la partition courante avec sa progression.

Ces deux instances sont passées au contrôleur ([VirutosoWebController](#)) qui sert à gérer les propriétés qui seront affichées ainsi que la communication avec le serveur.

Le constructeur crée ensuite le [ConceptWebServer](#) et lui enregistre une vue à l'aide du contrôleur.

Et pour finir, on crée une redirection depuis le chemin de base du serveur vers la page avec notre contrôleur. Exemple en local : <http://localhost/> → <http://localhost/Web/Virtuoso>.

7.10.3.2 Fonctionnement du contrôleur

Lors de la création d'un [VirtuosoWebServer](#), nous avons instancié une vue avec notre contrôleur. Par conséquent, grâce au principe de ConceptWeb, toutes les propriétés et fonctions **publiques** de notre contrôleur seront affichées sur la vue.

C'est aussi l'une des raisons pour laquelle le contrôleur existe, permettant ainsi de faire le tri des propriétés et fonctions de objets qui lui sont données.

Les propriétés du contrôleur seront définies plus tard, mise à part celle de la progression qui a une petite particularité.

7.10.3.3 Fonctionnement de la vue

Comme la vue par défaut de ConeptWeb utilise ses rendus et que nous voulons les modifier par d'autres rendus, il faut créer une « custom » vue. Cette vue permet par exemple de voir la progression qui est un double sous la forme d'une ProgressBar.

Pour faire cela, c'est très simple, nous avons créé une classe qui hérite de [WebMaterialView](#) et qui redéfinit les rendus utilisés.

```
[WebRenderCustom(nameof(VirutosoWebController.PartitionProgress), typeof(WebRoundProgressBarRender))]
[WebRenderCustom(nameof(VirutosoWebController.PartitionTitle), typeof(WebMaterialStringRefreshRender))]
[WebRenderCustom(nameof(VirutosoWebController.Partitions), typeof(WebMaterialShowListRender))]
[WebRenderCustom(nameof(VirutosoWebController.Catalogue), typeof(WebMaterialSelectListRender))]
public class CustomVirutosoWebView : WebMaterialView
{
    public CustomVirutosoWebView(VirutosoWebController model)
    {
        InitDefaultView(model);
        WebRoundProgressBarRender render;

        render = Find(nameof(VirutosoWebController.PartitionProgress)) as WebRoundProgressBarRender;
        render.Color = "33cc33";
        render.ValMin = 0;
        render.ValMax = 1;
    }
}
```

Les rendus pour nos propriétés sont modifiés grâce aux balises [WebRenderCustom](#).

Comme certains rendus sont paramétrables, la vue permet aussi de les paramétriser. Par exemple, dans le code ci-dessus, la couleur et les bornes de la ProgressBar sont redéfinies.

Pour finir, notre vue personnalisée est appliquée dans le constructeur de notre [VirtuosoWebServer](#).

```

public VirtuosoWebServer(Sequencer sequencer, Playlist principalPlaylist)
{
    _virutosoWebController = new VirutosoWebController(sequencer, principalPlaylist);

    Server = new ConceptWebServer();
    Server.RegisterWebApp("Virtuoso",
        new WebMaterialApp(new CustomVirutosoWebView(_virutosoWebController)));
}

Server.WoopsaServer.WebServer.Routes.Add("/", HttpMethod.GET,
    new RouteHandlerRedirect("Web/Virtuoso", WoopsaRedirection.Temporary));
}

```

7.10.3.4 Partition courante et progression

La partition courante avec sa progression a été le point le plus facile à implémenter, car les rendus avaient déjà été créés. Une seule subtilité pour le rafraîchissement de la progression.

Comme la progression de la partition courante est mise à jour trop souvent dans le séquenceur et qu'en conséquence la progress bar n'avancait pas assez vite, il a fallu adapter le temps de rafraîchissement de la propriété dans le contrôleur.

Nous avons donc en premier lieu une propriété standard :

```

public double PartitionProgress
{
    get
    {
        return _partitionProgress;
    }
    private set
    {
        _partitionProgress = value;
    }
}
private double _partitionProgress;

```

Mais qui est mise à jour par l'intermédiaire d'un de la fonction Elapsed d'un timer :

```

private void OnTimedEvent(Object source, ElapsedEventArgs e)
{
    PartitionProgress = _sequencer.PartitionProgress;
}

```

Ce timer est une variable ce qui permet de ne pas la voir dans la vue et qui est initialisé dans le constructeur.

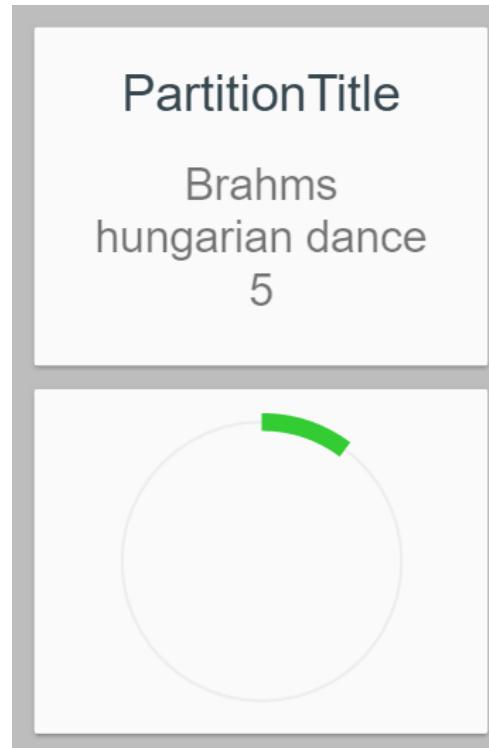
```
public VirutosoWebController(Sequencer sequencer, Playlist principalPlaylist)
{
    _sequencer = sequencer;
    _principalPlaylist = principalPlaylist;
    _timer = new System.Timers.Timer(500);
    _timer.Elapsed += OnTimedEvent;
    _timer.AutoReset = true;
    _timer.Enabled = true;
}
```

Comme il n'est pas nécessaire d'avoir un rafraîchissement très rapide pour la progression d'une partition qui habituellement de l'ordre de quelques minutes, nous avons choisi d'actualisé la propriété toutes les 500 millisecondes.

Vu que les rendus étaient déjà implémenté dans les dll, pour afficher comme il faut des deux propriétés sur la vue, il a suffit de mettre la commande montrée dans le « Fonctionnement de la vue » (chapitre précédent) :

```
[WebRenderCustom(nameof(VirutosoWebController.PartitionProgress), typeof(WebRoundProgressBarRender))]
[WebRenderCustom(nameof(VirutosoWebController.PartitionTitle), typeof(WebMaterialStringRefreshRender))]
```

Nous obtenons ainsi (sur la page Web) :



7.10.3.5 Développement du rendu pour les listes Web

7.10.3.5.1 C#

La partie C# du rendu est là pour faire la liaison de l'HTML/JavaScript au C# du reste de l'application Virtuoso.

Le rendu est une classe qui hérite de `WebPropertyRender`.

```
public class WebMaterialShowListRender : WebPropertyRender
```

Grâce à son héritage, il obtient une fonction BuildControl qui nous permet de modifier les codes HTML et JavaScript. Ces codes sont modifiés grâce à des `StreamReader`.

```
public override void BuildControl(WebPageBuilder pagebuilder)
{
    using (StreamReader file = new StreamReader(Assembly.GetExecutingAssembly().GetManifestResourceStream(
        "Framework.WebRender.WebMaterialShowListRender.Files.MaterialList.html")))
    {
        string tmp = file.ReadToEnd();
        tmp = tmp.Replace("<%title%>", Title);
        tmp = tmp.Replace("<%id%>", PropDescription.PropertyInfo.Name);
        pagebuilder.AppendHtml(tmp);
    }

    using (StreamReader file = new StreamReader(Assembly.GetExecutingAssembly().GetManifestResourceStream(
        "Framework.WebRender.WebMaterialShowListRender.Files.MaterialList.css")))
    {
        string tmp = file.ReadToEnd();

        pagebuilder.AppendCss(tmp);
    }

    using (StreamReader file = new StreamReader(Assembly.GetExecutingAssembly().GetManifestResourceStream(
        "Framework.WebRender.WebMaterialShowListRender.Files.MaterialList.js")))
    {
        string tmp = file.ReadToEnd();
        tmp = tmp.Replace("<%id%>", PropDescription.PropertyInfo.Name);
        string tmpVal = (Model as VirutosoWebController).Partitions.Replace("ff", "\",\"");
        tmp = tmp.Replace("<%values%>", "\"" + tmpVal + "\"");
        pagebuilder.AppendJs(tmp);
    }

    using (StreamReader file = new StreamReader(Assembly.GetExecutingAssembly().GetManifestResourceStream(
        "Framework.WebRender.WebMaterialShowListRender.Files.MaterialListCallBack.js")))
    {
        string tmp = file.ReadToEnd();
        tmp = tmp.Replace("<%id%>", PropDescription.PropertyInfo.Name);
        SubscriptionCallBackMethode = tmp;
    }
    pagebuilder.RegisterSubscription(PropDescription.PropertyInfo.Name, SubscriptionCallBackMethode, location);
}
```

7.10.3.5.2 HTML

Pour l'HTML, nous avons donné un titre au rendu ainsi qu'un id permettant de créer plusieurs instances du même rendu sans avoir de problème.

```
using (StreamReader file = new StreamReader(Assembly.GetExecutingAssembly().GetManifestResourceStream(
    "Framework.WebRender.WebMaterialShowListRender.Files.MaterialList.html")))
{
    string tmp = file.ReadToEnd();
    tmp = tmp.Replace("<%title%>", Title);
    tmp = tmp.Replace("<%id%>", PropDescription.PropertyInfo.Name);
    pagebuilder.AppendHtml(tmp);
}
```

Ce bout de code ouvre le fichier HMTL et remplace la balise `<%title%>` par une propriété de la classe (Title) et la balise `<%id%>` par le nom de l'instance.

```
<div class="mdl-cell mdl-cell--12-col">
    <h3 class="CardTitleText<%id%>"><%title%></h3>
</div>
<div class="mdl-cell mdl-cell--12-col">
    <form id="form<%id%>" class="ListFrom<%id%>">
        <div class="mdl-textfield mdl-js-textfield ">
            <p id="list<%id%>" style="overflow-y: scroll; height:200px; width:100%;"></p>
        </div>
    </form>
</div>
```

Ainsi avec les balises remplacées, on obtient un HTML avec un titre (Élément `<h3>`) et une liste (Élément `<p>`) qui peut se scroller. Ce qui nous donne :



7.10.3.5.3 JavaScript

La partie JavaScript est séparée en deux parties, la première est la partie qui est appelée au chargement de la page Web.

```
using (StreamReader file = new StreamReader(Assembly.GetExecutingAssembly().GetManifestResourceStream(
    "Framework.WebRender.WebMaterialShowListRender.Files.MaterialList.js")))
{
    string tmp = file.ReadToEnd();
    tmp = tmp.Replace("<%id%>", PropDescription.PropertyInfo.Name);
    string tmpVal = (Model as VirutosoWebController).Partitions.Replace("ff", "\",\"");
    tmp = tmp.Replace("<%values%>", "\"" + tmpVal + "\"");
    pagebuilder.AppendJs(tmp);
}
```

Afin de contourner des problèmes de gestion de types, la liste doit être reçue par le rendu sous la forme d'un string avec chaque item séparé par les caractères ff. Ainsi, on remplace la balise `<%values%>` par les valeurs de la liste, mises entre guillemets et séparées par une virgule. Ces modifications sont faites pour que le JavaScript reprenne la liste dans un tableau de string.

```

var parent<%id%> = document.getElementById("list<%id%>");
var listName<%id%> = [<%values%>];

function AddItemShow(i){
    <!--Ajoute la valeur à la liste-->
    var new_name = document.createElement("label");
    new_name.appendChild(document.createTextNode(listName<%id%>[i]));
    parent<%id%>.appendChild(new_name);

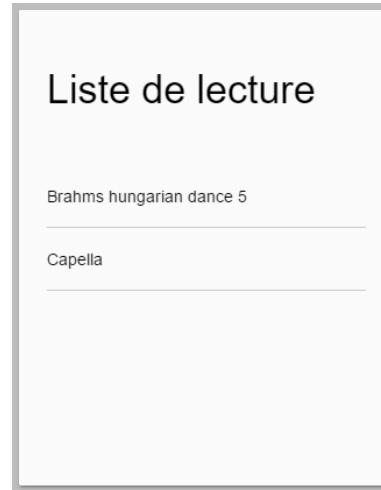
    parent<%id%>.appendChild(document.createElement("hr"));
};

function ActualiseShowList(){
    <!--$( "list<%id%>" ).empty();-->
    parent<%id%>.innerHTML = "";
    <!--Création de la liste-->
    for (var i = 0; i < listName<%id%>.length; i++) {
        AddItemShow(i);
    }
}

ActualiseShowList();

```

Le JavaScript reçoit la liste des valeurs dans son tableau « `listName<%id%>` ». Puis appelle la fonction « `ActualiseShowList()` ». Ainsi pour chaque item dans le tableau, on ajoute un item dans la liste HMTL (la balise `<p>`) et finalement, on obtient ceci :



La deuxième partie du JavaScript est là pour intervenir lorsqu'il y a un changement dans la liste. Par conséquent, notre liste reste à jour.

```

using (StreamReader file = new StreamReader(Assembly.GetExecutingAssembly().GetManifestResourceStream(
    "Framework.WebRender.WebMaterialShowListRender.Files.MaterialListCallBack.js")))
{
    string tmp = file.ReadToEnd();
    tmp = tmp.Replace("<%id%>", PropDescription.PropertyInfo.Name);
    SubsciptonCallBackMethode = tmp;
}

```

Ici, nous reprenons juste l'id qui doit être le même que pour les points précédent. Permettant, dans notre cas, d'accéder à notre tableau dans le JavaScript.

```
listName<%id%> = value.split("ff");
ActualiseShowList();
```

Le JavaScript recharge les valeurs qui ont changé dans la liste et actualise la liste HTML.

7.10.3.5.4 Rôle du contrôleur

Comme le rendu pour l'affichage d'une liste demande un string avec des items séparés par les caractères ff, cela demande une intervention du contrôleur pour convertir la liste en string.

Donc, la propriété « Partitions » transforme la liste en string dans sa propriété « get ». De ce fait, quand le rendu fera la requête de la liste des partitions, il recevra le string dont il a besoin.

```
public string Partitions
{
    get
    {
        string tmp = "";
        for (int i = 0; i < _principalPlaylist.Partitions.Count; i++)
        {
            tmp += _principalPlaylist.Partitions[i].Title;
            if (i != _principalPlaylist.Partitions.Count - 1)
                tmp += "ff";
        }
        return tmp;
    }
}
```

7.10.3.6 Développement de la demande d'ajout de partition

La demande d'ajout se fait en deux parties. La première est la sélection d'une partition et la deuxième est la requête pour ajouter la partition sélectionnée.

La partie C# étant la même que pour le rendu précédent (7.10.3.5.1), elle ne sera pas décrite ici.

Par contre, la liste sélectionnable change dans ces codes HTML et JavaScript.

7.10.3.6.1 HTML

```
<div class="mdl-cell mdl-cell--12-col">
    <h3 class="CardTitleText<%id%>"><%title%></h3>
</div>
<div class="mdl-cell mdl-cell--12-col">
    <form id="form<%id%>" class="ListFrom<%id%>">
        <div class="mdl-textfield mdl-js-textfield ">
            <p id="listSelectable<%id%>" style="overflow-y: scroll; height:200px; width:100%;"></p><br><br>
            Partition sélectionnée : <input id="inputSelected<%id%>" disabled style="height:20px; width:100%;">
        </div>
    </form>
</div>
```

Dans le but de voir quel item a été sélectionné, nous avons ajouté un input qui, on le verra dans la partie JavaScript, s'actualise lors d'une sélection. Il permettra aussi au contrôleur de savoir quel élément a été choisi.

7.10.3.6.2 JavaScript

Comme nous voulons que le code JavaScript nous renvoie des informations, il faut préciser la propriété que ça devra affecter.

```

using (StreamReader file = new StreamReader(Assembly.GetExecutingAssembly().GetManifestResourceStream(
    "Framework.WebRender.WebMaterialSelectListRender.Files.MaterialListSelect.js")))
{
    string tmp = file.ReadToEnd();
    tmp = tmp.Replace("<%id%>", PropDescription.PropertyInfo.Name);
    tmp = tmp.Replace("<%path%>", location + "/" + PropDescription.PropertyInfo.Name);
    string tmpVal = (Model as VirutosoWebController).Catalogue.Replace("ff", "\",\"");
    tmp = tmp.Replace("<%values%>", "\"" + tmpVal + "\"");
    pagebuilder.AppendJs(tmp);
}

```

Pour définir cette propriété, nous avons mis une balise <%path%>. Cette balise est assignée à la propriété sur laquelle nous appliquons le rendu.

```

var listSelectable<%id%> = document.getElementById("listSelectable<%id%>");
var inputSelected<%id%> = document.getElementById("inputSelected<%id%>");
var listName<%id%> = [<%values%>];

function AddItemSelectable(i) {
    <!--Ajoute la valeur à la liste-->
    var new_name = document.createElement("label");
    <!--new_name.appendChild(document.createTextNode(listName<%id%>[i]));-->
    new_name.innerHTML = listName<%id%>[i];
    new_name.class = "itemListLabelClickable";
    new_name.onclick = function() {
        inputSelected<%id%>.value = this.innerHTML;
        server.write("<%path%>", inputSelected<%id%>.value, function(value) {});
    }
    listSelectable<%id%>.appendChild(new_name);

    listSelectable<%id%>.appendChild(document.createElement("hr"));
};

function ActualiseSelectList() {
    <!--$( "listSelectable<%id%>" ).empty();-->
    listSelectable<%id%>.innerHTML = "";
    <!--Création de la liste-->
    for (var i = 0; i < listName<%id%>.length; i++) {
        AddItemSelectable(i);
    }
}

ActualiseSelectList();

```

Le JavaScript ressemble beaucoup à l'autre excepté que les items de la liste ont une fonction « onclick ». Et c'est cette fonction qui permet d'afficher l'item sélectionné dans l'« input » et d'envoyer la valeur au C# à l'aide de « server.write » à la propriété mise pour « path » dans le C#. En envoyant la propriété, on déclenche le setter de la propriété qui est dans le contrôleur.

Ainsi en le déclenchant, nous assignons la valeur à une variable privée du contrôleur.

```
public string Catalogue
{
    get
    {
        string path = FrameworkController.Instance.Settings.DefaultPathLoadFile + "\\Catalogue\\";
        string tmp = "";
        int i = 0;
        string[] fileNames = Directory.GetFiles(path, "*.xml");

        foreach (string fileName in fileNames)
        {
            tmp += fileName.Remove(0, path.Length);
            if (i++ != fileNames.Length - 1)
                tmp += "FF";
        }
        return tmp;
    }
    set { _selectFileName = value; }
}
```

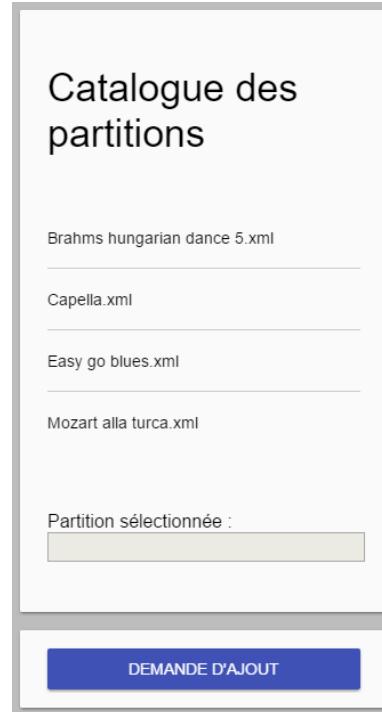
Maintenant qu'il est possible de choisir une partition, il reste la partie demande d'ajout. Cette partie est gérée avec une fonction. (Le rendu des fonctions étant déjà implémenté dans les dll de ConceptWeb, il nous a simplement suffi de créer la fonction de demande.

```
public void DoRequest()
{
    if (_selectFileName != null && _selectFileName != "")
    {
        PartitionXylo p = new PartitionXylo();
        var messages = new MessageCollection();

        p.LoadFromFile(FrameworkController.Instance.Settings.DefaultPathLoadFile + "\\Catalogue\\"
            + _selectFileName, PluginClassManager.AllFactories, messages);
        p.Name = _selectFileName.Split('.')[0];
        if (messages.Count == 0)
            Application.Current.Dispatcher.Invoke(new Action(() => _principalPlaylist.AddPartition(p)));
    }
}
```

Cette fonction reprend la variable privée avec le nom du fichier et l'ajoute dans la liste de lecture.

Ainsi on obtient, sur la page Web, le rendu suivant :



Où le bouton bleu est le rendu de notre fonction.

7.10.4 Rendu final

Grâce à ConceptWeb, nous sommes arrivés avec un effort réduit à une page Web plutôt jolie sans avoir touché au CSS. De plus, elle implémente les fonctionnalités les plus importantes du cahier des charges.

8 Gestion de projet

8.1 Planning

8.1.1 Problème

Pour suivre les modifications et savoir pourquoi un projet est arrivé ou non à bout, il faut un planning de travail. Ce planning doit décrire et permettre de suivre le déroulement du projet. Un premier planning pour la pré-étude doit être fait afin de savoir où partira le projet. Puis un deuxième planning est réalisé à la fin de la pré-étude (planning du projet).

8.2 Démarche

Pour la pré-étude, nous avons fait le planning au début du travail de bachelor.

À la fin de la pré-étude, il s'est avéré que le temps estimé était plus moins correcte, hors mis pour le diagramme UML. Même si l'UML aurait pu être plus poussé, notamment du côté des fonctionnalités des classes, il reste très proche de diagramme final.

Finalement, cette phase du travail de bachelor s'est terminée avec un peu d'avance grâce aux imprévus qui n'ont pas été nombreux.

N°	Description	Resp	Effort estimé						Consommé	Reste à faire	Delta
			s. 8	s. 9	s. 10	s. 11	s. 12	s. 14	s. 15	s. 16	
T10	Windows IoT/Raspberry Pi										
T10.10	Installer windows iot		4	2					2	0	2.0
T10.20	Faire clignoter une led		6	4					4	0	2.0
T20	Evaluation des protocoles inconnus								0		0.0
T20.10	Allumer une led avec I2C								0		0.0
T20.20	Créer une page html								4	0	0.0
T20.40	Utiliser WOOPSA pour communiquer entre la page html								4	0	0.0
T20.50	Récupérer les notes d'un instrument dans un fichier midi								10	0	-2.0
T30	Performance et fonctionnalité window iot								0		0.0
T30.10	Vérifier performance 1ms avec un thread haute priorité(oscilloscope)								0		0.0
T30.20	Vérifier portabilité WOOPSA et estimer l'effort (christophe)								2.5	0	-1.5
T30.30	Vérifier lecture/écriture de fichier								0.5	0	0.5
T30.40	Comparatif								0.5	0	0.5
T30.50	touchscreen								1.5	1	0
T40	Etudes des fonctionnalité								0.5	0	0.5
T40.10	Identifier les cas d'utilisation								1	0.5	0.5
T40.20	Diagramme de classe								2	1	1
T40.30	Architecture logicielle (UML)								1.5	1.5	8.5
T40.40	Cahier des charges détaillé								0.5	0	3.5
T40.50	Planning du projet								3.5	0	0.5
T40.50	Rapport								4	0	0.0
T50	Concept HMI								0		0.0
T50.10	Rendez-vous chez Objectis								3	0	0.0
T50.20	Tutoriels Concept								8	4	-2.0
T80	Imprevus (10 %)								10	1	10.0
T90	Gestion de projet (10%)								10	1.5	3.5
	TOTAL								102	6	22.0
									12.5	13	12.5
									9.5	2	0
									80	0	0

Pour la partie projet, les temps estimés dans le planning étaient un peu trop optimistes. Ce qui nous amène à un léger surplus de temps d'environ 7%.

Le planning a subi un léger changement en cours de route, car la priorité s'est réorientée sur le serveur Web au détriment de la partie édition des fichiers MIDI. Cette cause vient de plusieurs raisons. En premier lieu, nous avons estimé que l'interaction avec les utilisateurs était très importante. En plus, la conversion des fichiers marchant déjà au moment du changement, nous pouvions éditer les fichiers à l'aide du programme « MidiEditor ».

N°	Description	Resp	Effort estimé	s. 15	s. 16	s. 17	s. 18	s. 19	s. 20	s. 21	s. 22	s. 23	Consommé	Reste à faire	Delta
T00	Départ du projet			0.3	0.3								0	0.0	0.0
T00.10	Initialisation d'un répertoire git			2	1	1							0.3	0.0	0.0
T00.20	Création des programmes avec concept (Xylobot et Midi)			2									2	0.0	0.0
T10	Prototype essentiel												0	0.0	0.0
T10.10	Etablir le protocole de communication entre le NUC et l'Arduino			2	1	0.5							1.5	0.5	0.5
	Envoyer/recevoir des données via usb (C# et sur l'Arduino)												0	0.0	0.0
	Client C#												0	0.0	0.0
T10.20	Dev classe Xylobot			0.5	0.5								0.5	0.0	0.0
T10.30	Créer et paramétrier communication sérielle			3	1	2	1						4	-1.0	-1.0
T10.40	Gestion envoi/réponse d'un télégramme, timeout, rémission			2		1.5	1	1					3.5	-1.5	-1.5
T10.50	Test et validation protocole bas niveau			1		1		1					1	0.0	0.0
T10.60	Implémentation message sendNotes			1		0.5		0.5					1.5	-0.5	-0.5
T10.70	Implémentation message Start			1		0.5		0.5					0.5	0.5	0.0
T10.80	Implémentation message Stop			0.5		0.5		0.5					0.5	0.0	0.0
	Server Arduino												0	0.0	0.0
T10.90	Application de base Arduino			2		15							15	0.5	0.5
T10.100	Initialisation communication sérielle			1	0.5	1		1					1.5	-0.5	-0.5
T10.110	Reconnaître début de message			0.5	0.5								0.5	0.0	0.0
T10.120	Déclarer et gérer types de message			2		15	2						3.5	-1.5	-1.5
T10.130	Eliminer les messages invalides (taille de données incorrects, mauvais numéros)			1		1	1	1					2	-1.0	-1.0
T10.140	Envoi réponse acquittement			1		0.5	1						1.5	-0.5	-0.5
T10.150	Implémentation message SendNotes			1		1		1					2	-1.0	-1.0
T10.160	Implémentation message Start			1		1		1					1.5	-0.5	-0.5
T10.170	Implémentation message Stop			1		1		1					1.5	-0.5	-0.5
T10.180	Implémenter tampon circulaire notes			1		1	0.5						1.5	-0.5	-0.5
T10.190	Test (unitaire automatisé visual studio ?) du tampon circulaire			1.5		1.5	0.5						2	-0.5	-0.5
T10.200	Message réponse espace note disponible			1		0.5	0.5						1	0.0	0.0
T10.210	Implémentation machine d'état principale ou des interruptions			1.5		2	0.5						2.5	-1.0	-1.0
T10.220	Diagramme de séquence			1		0.5							0.5	0.5	0.0
	Implémentation de l'I2C												0	0.0	0.0
T10.220	Implémenter l'I2C			3				1	2.5	2	3		8.5	-5.5	-5.5
T10.230	Jouer les notes reçues avec un tempo fixe			2				1	1	3			5	-3.0	-3.0
	UserControlFileManagement												0	0.0	0.0
T10.240	Implémenter l'interface			3									1.5	1.5	1.5
	midi => XML												0	0.0	0.0
T10.250	Implémenter la librairie midi dans le projet					0.5							0.5	0.0	0.0
T10.260	Coder la classe PartitionMidi					1.5			2				3	-1.5	-1.5
T10.270	Coder la classe PartitionXml					1			1				1	0.0	0.0
T10.280	Charger et récupérer les données d'un fichier midi					2			2				2	0.0	0.0
T10.290	Convertir une PartitionMidi en PartitionXml					2			1.5				3	-1.0	-1.0
T10.300	Enregistrer une PartitionXml dans un fichier XML					2			1.5				3.5	-1.5	-1.5

N°	Description		Resp	Effort estimé	s. 21	s. 22	s. 23	s. 24	s. 25	s. 26	Consommé	Reste à faire	Delta	
T20	Prototype 2										0	0.0	0.0	
	UserControlEditPlaylist										0	0.0	0.0	
T20.10	Implémenter l'interface				3	1	1	1	2		5	-2.0	-2.0	
T20.20	Afficher le catalogue					2	0.5				0.5	1.5	1.5	
T20.30	Charger partition XML dans une instance de PartitionXylo				4	1	0.5				1.5	2.5	2.5	
T20.40	coder la classe Playlist				3	0.5	0.5				1	2.0	2.0	
	UserControlSupervision										0	0.0	0.0	
T20.50	Implémenter l'interface					3	2		2		2	4	-1.0	-1.0
T20.60	Implémenter le start/stop					4	2	0.5		0.5	3	1.0	1.0	
T20.70	nom de la partition en cours de lecture					2	2	0.5			2.5	-0.5	-0.5	
	Client C#										0	0.0	0.0	
T20.80	Implémenter l'intensité des notes					4			2.5		2.5	1.5	1.5	
	Serveur Arduino										0	0.0	0.0	
T20.90	Implémenter l'intensité des notes					3		3		3	3	0.0	0.0	
	midi => XML										0	0.0	0.0	
T20.100	Implémenter l'intensité des notes					4		3.5			3.5	0.5	0.5	
	Virtuoso										0	0.0	0.0	
T20.110	Prototype visuel					4	2	3	1	6	-2.0	-2.0	-2.0	
T20.120	Réparation du boîtier					0	3		3	3	-3.0	-3.0	-3.0	

N°	Description			Resp	Effort estimé	s. 21	s. 22	s. 23	s. 24	s. 25	s. 26	s. 27	s. 28	s. 29	s. 30	Consommé	Reste à faire	Delta
T30	Prototype 3															0	0.0	0.0
	NUC et Touchscreen															0	0.0	0.0
T30.10	Remote debugging					4		2	4							6	-2.0	-2.0
T30.20	Utilisation du Touchscreen					3		1	2							3	0.0	0.0
T30.30	Esthétique de l'application Xylobot					20				14	5	6				25	-5.0	-5.0
	Client C#															0	0.0	0.0
T30.40	Implémenter message Tempo					1		1								1	0.0	0.0
	Server Arduino															0	0.0	0.0
T30.50	Implémenter message Tempo					1		1								1	0.0	0.0
	Conversion des fichiers midi>XML															0	0.0	0.0
T30.60	Intégrer le tempo					4										3	1.0	1.0
	UserControlSupervision															0	0.0	0.0
T30.70	Changer la vitesse de lecture					4		2	4							6	-2.0	-2.0
T30.80	Afficher le déroulement de la partition en cours					8		6	4	2						12	-4.0	-4.0
	Serveur Web															0	0.0	0.0
T30.90	Implementation du serveur en Woopsa					4										5	5	-1.0
	Interface															0	0.0	0.0
T30.100	Implémenter la demande d'ajout pour une musique dans la playlist					12										10	2.0	2.0
T30.110	Création de la page hmi de supervision					4										3	1.0	1.0
T30.120	Implémenter la playlist sur la page					10										8	-6.0	-6.0
T30.130	Implémenter la partition en lecture					8										6	0.0	0.0
T30.140	Créer le wifi Virtuoso					4										2.5	2.5	1.5
	UserControlEditPlaylist															0	0.0	0.0
T30.140	UserControl liste de lecture (playlist principale)					6										2	2.0	2.0
T30.150	Gestion de la liste de lecture (playlist principale)					6										2	4.0	4.0
T30.160	UserControl Gestion des Playlists					6										4	2.0	2.0
T30.170	UserControl Gestion d'une Playlist					6										5	1.0	1.0
T30.180	Changement de logique pour la gestion des playlists (Aucun "add" dans l'application)					0										4	-4.0	-4.0
T30.190	Lancement automatique de l'application Virtuoso					1										1	0.0	0.0
T30.200	Arrêt du NUC depuis l'application Virtuoso					15										1	0.5	0.5
T31	Debugger la gestion USB															0	0.0	0.0
T31.10	Corriger le bug de "tick" dans l'arduino					0										8	2	2
																12	-12.0	-12.0

N°	Description		Resp	Effort estimé	s. 21	s. 22	s. 23	s. 24	s. 25	s. 26	s. 27	s. 28	s. 29	s. 30	Consommé	Reste à faire	Delta
T40	Prototype 4														0	0.0	0.0
T40.10	UserControlEditPartition			Implémenter l'interface											0	0.0	-1.0
T40.20		Afficher les notes dynamiquement													6.5		1.5
T40.30		Implémenter les fonctions d'édition rapide (transposition et suppression)													0	6	0.0
T40.40		Lecture de la partition													0	8	0.0
T40.50		Implémenter le clt+Z													0	4	0.0
T40.60	UserControlSettings (Xyo)			Implémenter l'interface											0	0.0	-1.0
T40.70		Chemin de recherche des fichiers													3	4	0.5
T40.80		Régler le temps de frappe des notes													1.5	1.5	-2.0
T40.90	UserControlSettings (Midi)			Implémenter l'interface											0	0.0	0.0
T40.100		Chemin d'enregistrement des fichiers													1	3	0.0
T41.10		Enlever splashscreen concept et mettre la licence													0	1	-1.0
	Prototype 5														0	0.0	0.0
T50.10	Assistant midi->xml			Implémenter les vues pour la démarche (exemple: prend les notes les plus aigues)										6	6	0.0	
T50.20		Codez les algorithmes de la démarche (exemple: prend les notes les plus aigues)													10	10	0.0
T60.0	Amélioration finale														0	10	-10.0
T70	Administratif														0	0.0	0.0
T70.10	Rapport														40	1	-8.0
T70.20		Présentation fin de semestre													8	1	-2.0
T80	Imprevus (10 %)														32	0	32.0
T90	Gestion de projet (-10 %)														32	1	7.5
	TOTAL														366.3	14	115

9 Informations pratiques

9.1 Chargement de partition sur le NUC

Pour charger des partitions sur le NUC, il suffit de se connecter au wifi « Virtuoso ». Puis dans l'explorateur de fichiers Windows et saisir le chemin : <\\VIRUTOSO\\>

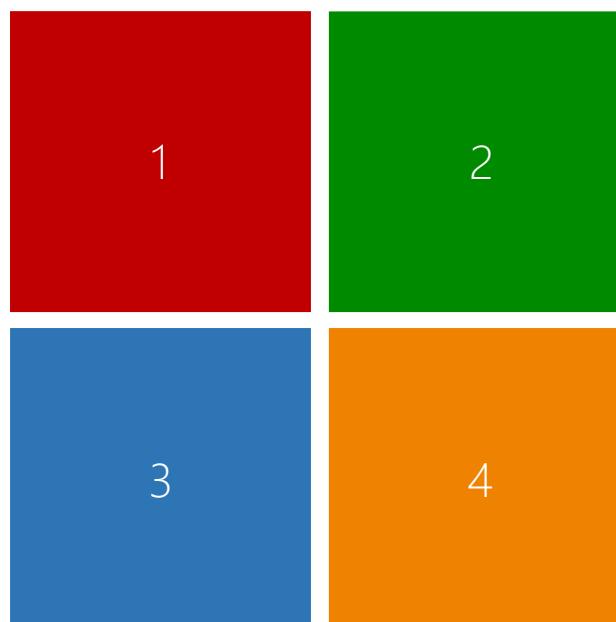
Si vous n'êtes pas encore logger, le nom d'utilisateur est « Heig » et le mot de passe est « qwertz ».

Une fois connecter il faut accéder au bureau du NUC et ouvrir le seul dossier partager (le seul dossier qui est visible) nommé « Virtuoso » dedans il y a un dossier pour les partitions.

9.2 Code pour l'accès au paramètre de l'application

Dans l'application Virtuoso, un code a été placé.

L'interface se présente de la manière suivante (sans les chiffres) :



Pour accéder, il faut taper le code : 1 2 3 4

10 Améliorations possibles

10.1 Métallophone

La première amélioration possible, serait de changer le xylophone par un métallophone, car ce dernier offre une meilleure sonorité et cela améliorerait la qualité musicale. De plus les touches s'abimeraient moins au contact des actuateurs.

10.2 Édition des fichiers MIDI

Comme, arrivés au temps imparti du projet, nous n'avons pas eu le temps de finir le programme de conversion des fichiers MIDI. Malgré sa conversion parfaite des fichiers, il n'a pas sa partie d'édition implémentée.

L'idéal serait qu'il implémente des fonctions prévues pour ajuster la partition afin de correspondre aux attentes du xylophone.

10.3 Gestion des erreurs

Un point, qui nous pensons aurait été important pour la maintenance, est la gestion des erreurs. Celle-ci à quelque base dans le programme qui ont servi au débogage de quelques petits points, mais qui n'est aucunement utilisable.

Pour qu'elle soit exploitable, il faudrait ajouter une partie graphique affichant les erreurs et implémenter toutes les erreurs importantes qui pourraient apparaître.

11 Conclusion

Après de longues heures de recherche et développement, le projet arrive à son terme. Les multiples technologies de ce projet ont été acquises et implémentées. Les interfaces utilisateurs sont simples, belles et efficaces. La page Web est aboutie et aisée d'utilisation et offre une bonne interaction avec les utilisateurs, ce qui est son but premier.

Tous ces points mènent à un projet respectant le cahier des charges et, qui plus est, abouti excepté l'édition des fichiers MIDI. Concrètement le déroulement du projet s'est bien déroulé selon les prévisions du planning, excepté quelques petits dépassements.

La mise en œuvre d'un tel projet, qui implique autant de technologies et de temps, apporte beaucoup pour l'apprentissage de la planification. Que ce soit pour la partie planning qui est présente pour tous les projets, où que ce soit la partie UML qui est là pour les programmes orientés objets.

On remarque aussi que la gestion de projet prend une grande importance et que notre planification de départ peut s'avérer partiellement fausse.

12 Liste des références

OPENCLASSROOM, « *Apprenez à créer votre site web avec HTML5 et CSS3* », in *OpenClassroom*, <https://openclassrooms.com/courses/apprenez-a-creer-votre-site-web-avec-html5-et-css3>, consulté en juillet 2016.

OPENCLASSROOM, « *Dynamisez vos sites web avec JavaScript !* », in *OpenClassroom*, <https://openclassrooms.com/courses/dynamisez-vos-sites-web-avec-javascript>, consulté en juillet 2016.

13 Journal de travail

Le planning du point 8.1 fait office de planning. Il décrit le déroulement du travail de bachelor et implémente un suivi par semaine des heures passées sur le projet.

14 Annexes

Les codes sources sont en annexe, mais pas dans le rapport dû alors taille conséquente.