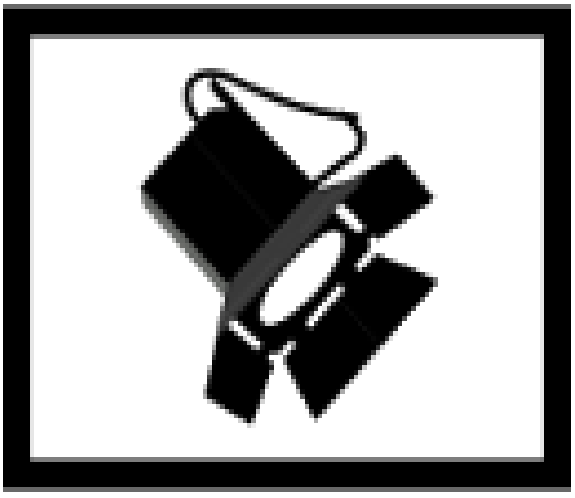
Reynier Tony

BTS SN SESSION 2018

Projet SLDMX (Revue Finale)

Éclairage de scène automatisé



campus La Salle

www.campus.lasallefrance.fr

Sommaire

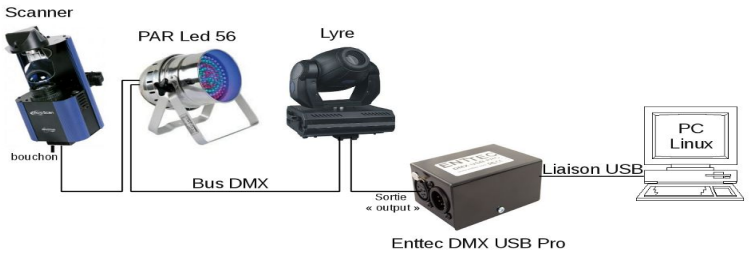
Cahier des charges

Présentation générale du projet

De nos jours,les DJ et animateurs de soirée utilisent couramment un ordinateur portable à la fois pour diffuser la musique et pour gérer les différents jeux de lumières pour éclairer et animer la piste de « spectacle » (danse, podium, scène,...)

Nous disposons ici d'une chaîne DMX, pouvant être connectée à un PC via une interface USB/DMX, actuellement composée de 6 projecteurs dont :

1 x Scanner
2 x PAR
2 x Lyres
1 x Laser



Durant ce projet, un septième projecteur, unique en son genre, sera conçu par l'étudiant EC dans le but de pouvoir diriger un éclairage pouvant tourner à 360°.

Il faudra également permettre le pilotage de la chaîne à travers 2 interfaces :

- Un pupitre PlayBack Wing de la marque ENTTEC
- Une application développée via l'outil Qt sur Linux

L'application Qt devra d'ailleurs permettre, au delà du simple pilotage de projecteurs, la création de scènes, spectacles et séquences qui seront ensuite joués depuis cette dernière.

EQUIPE DU PROJET

Élève(s) EC :

- Enzo Guibanni

Élève(s) IR :

- Thomas Demont
- Tony Reynier

Répartition des tâches

Partie EC (Enzo Guibanni)

Conception d'un projecteur spécial pouvant tourner à 360°
Paramétrage des canaux DMX du projecteur
Implémentation de ce même projecteur à la chaîne DMX

Partie IR (Tony Reynier)

Développement de la partie du logiciel concernant la création de spectacles, de scènes et de séquences
Développement des options d'ajout de projecteur sur la chaîne DMX depuis le logiciel
Création et renseignement des fichiers XML servant à la sauvegarde de paramètres des projecteurs et autres

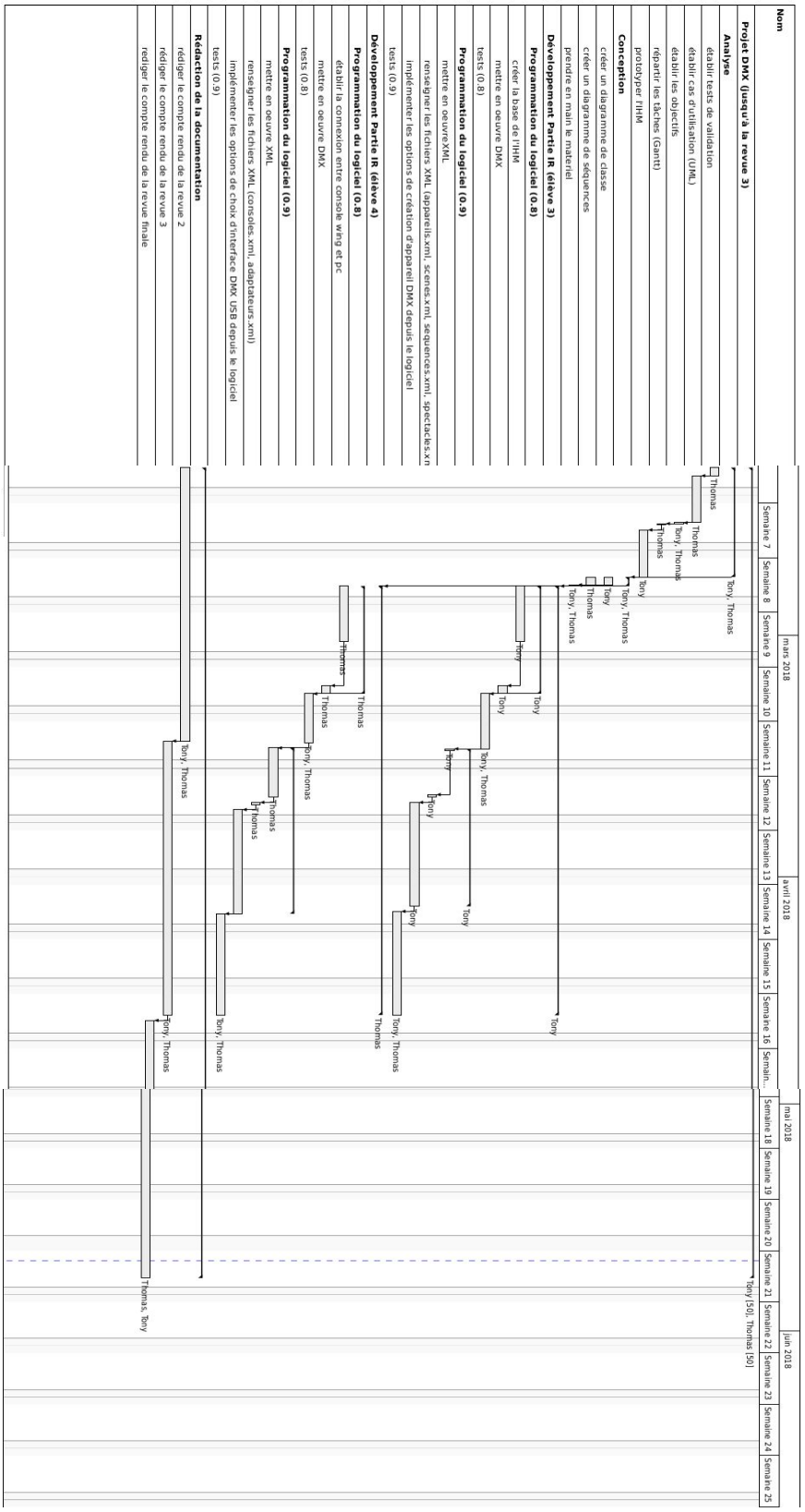
Partie IR (Thomas Demont)

Établissement de la connexion entre la console Playback Wing et la chaîne DMX

Développement de la partie du logiciel concernant le pilotage des projecteurs et canaux depuis ce dernier

Création et renseignement de fichiers XML pour les configurations des adaptateurs DMX et des consoles/pupitres

Planification



Les ressources

DMX512

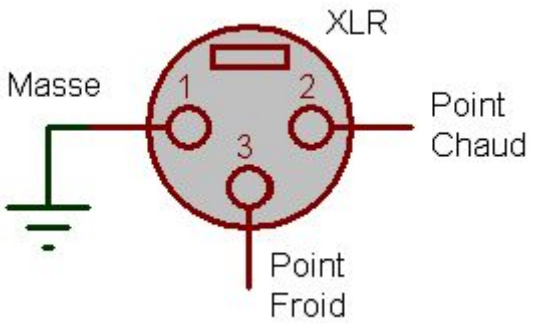
DMX512 est une norme et un protocole dédié à l'usage de gradateurs et d'interfaces servant à manipuler les projecteurs et autres spots lumineux.

Les configurations et pilotages de projecteurs sont pensés à travers un système de canaux : chaque projecteur possède un nombre délimité de canaux dans sa fabrication. Chaque canal sur un projecteur et/ou une chaîne DMX est identifiable par un numéro pouvant être compris entre 1 et 512. Chaque canal a un rôle spécifique et génère un mouvement, une direction ou un paramètre spécifique dudit projecteur. Chaque canal possède une valeur entre 0 et 255 qui se modifie dès que l'on opère une manœuvre depuis les gradateurs.

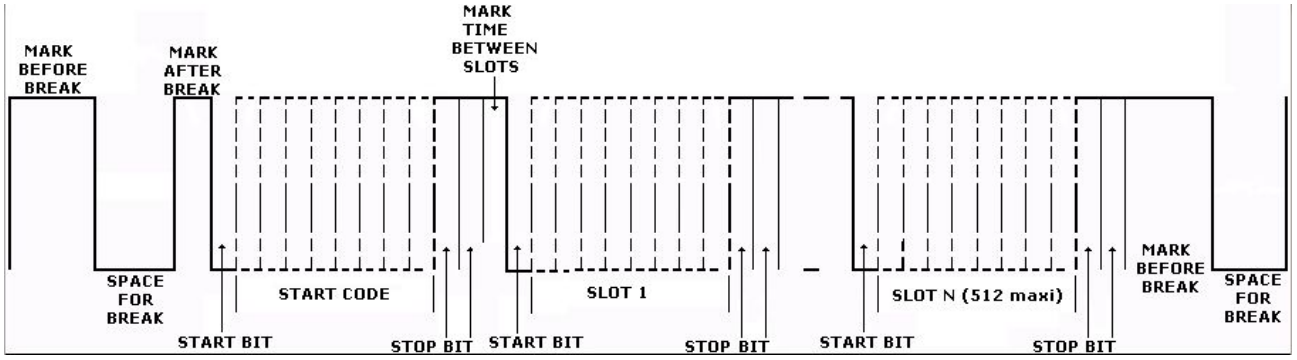
La norme DMX512 implique l'usage de connecteurs XLR à 3 broches ou 5 broches (si on utilise ceux à 5 broches, alors les broches 4 et 5 seront inutilisées).

Le débit de transmission des données est de 250 Kbits/s et la norme est RS485.

La broche n°1 fait office de masse/blindage pour limiter les pertes tandis que la broche n°2 représente le point Chaud qui envoie les trames de données et la broche n°3, quand à elle, est le point Froid qui transmet la trame avec une polarité inversée .

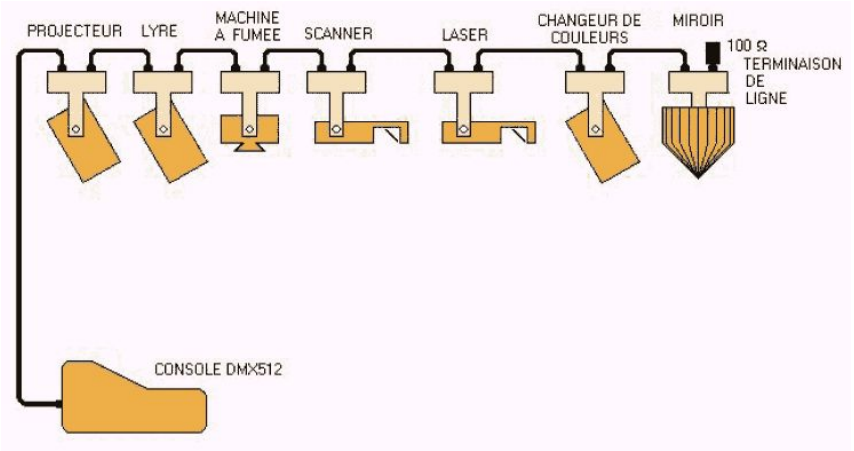


La trame DMX est composée d'une série d'octets (512 au maximum), le premier octet représente systématiquement le code de départ tandis que chaque octet suivant représente une instruction à un canal DMX.



Les appareils DMX sont reliés entre eux par le biais d'une topologie de réseau en Bus, les appareils sont reliés physiquement entre eux par un chaînage.

Un appareil DMX a systématiquement deux ports XLR dont l'un est l'entrée des données et l'autre en est la sortie, le câble de sortie d'un projecteur est relié directement à l'entrée de l'appareil voisin de la chaîne, et dès lors que l'on arrive à la fin de la chaîne, le port de sortie du dernier appareil se voit doté d'un bouchon afin d'éviter toute reflection du signal.



Interfaces USB DMX

Dans le cadre de ce projet, nous avons deux de ces interfaces : Open DMX USB et DMX USB Pro

Le PC est relié à l'interface par un câble USB A/B d'un coté tandis que de l'autre, nous avons un connecteur XLR5 qui relie l'interface à la chaîne DMX.

Ces interfaces ont chacune pour but de servir d'intermédiaire entre le PC qui envoie les informations aux canaux en récupérant ces dernières depuis l'interface USB pour constituer la trame DMX afin de l'envoyer sur la chaîne depuis l'interface XLR5.

ENTTEC Playback Wing

La console Wing est un pupitre disposant de 40 boutons génériques, 10 gradateurs, 2 grands boutons « GO/BACK » et 2 boutons « Page UP/Page DOWN ».

Elle est connectée directement au réseau via une liaison Ethernet et possède donc une adresse IP (ici étant actuellement 192.168.52.212) .

Lorsque une pression ou un relâchement d'un bouton ou un déplacement d'un gradateur se fait, Le pupitre envoie une trame dont la constitution reprend des protocoles propres au fabricateur du pupitre et le protocole UDP sur sa couche transport.

Cette trame est donc envoyée en broadcast UDP (ici via l'adresse 192.168.52.255) et peut donc être récupérée et interprétée par n'importe quel programme adapté qui est installé sur l'un des PC connectés au même réseau local.

Projecteurs

Les projecteurs utilisent chacun un canal DMX dit « canal de départ », ce canal est défini soit par le paramétrage sur l'interface LCD du projecteur, soit en modifiant les positions des interrupteurs *dip switchs* situés au dos du projecteur.

En ayant défini ce canal de départ, il nous est donc possible de connaître les autres canaux alloués aux projecteurs de la chaîne selon le type et les spécificités de ces derniers.

Scanner (Nombre de canaux DMX: 6)

- canal de départ : diaphragme
- canal de départ+1 : gobo / motif de la lumière
- canal de départ+2 : rotation du gobo /motif
- canal de départ+3 : couleur de la lumière
- canal de départ+4 : rotation horizontale du projecteur (pan)
- canal de départ+5 : inclinaison du miroir (tilt)

PAR (Nombre de canaux DMX: 4)

- canal de départ : intensité couleur rouge
- canal de départ+1 : intensité couleur verte
- canal de départ+2 : intensité couleur bleue
- canal de départ+3 : mode dimmer ou clignotement

Lyre (Nombre de canaux DMX: 6)

- canal de départ : ouverture de la lumière (shutter)
- canal de départ+1 : gobo / motif de la lumière
- canal de départ+2 : rotation du gobo/motif
- canal de départ+3 : couleur de la lumière
- canal de départ+4 : rotation horizontale du projecteur (pan)
- canal de départ+5 : rotation verticale du projecteur (tilt)

Laser(Nombre de canaux DMX : 10)

- canal de départ : mode de contrôle
- canal de départ+1 : blanking/extinction/blackout
- canal de départ+2 : galerie d'images
- canal de départ+3 : déplacement
- canal de départ+4 : rotation
- canal de départ+5 : mouvement du laser (horizontal et/ou vertical)
- canal de départ+6 : étirement du laser (horizontal et/ou vertical)
- canal de départ+7 : vitesse
- canal de départ+8 : vitesse du tracé de figures laser
- canal de départ+9 : taille de l'image

QT (API de développement orienté objet)

Qt est une bibliothèque logicielle orientée objet développée en C++ et essentiellement dédiée au développement d'interfaces graphiques. L'interface graphique est basée sur un système dit de « widgets » qui consistent eux même en une série de boutons, gradateurs, fenêtres, onglets, etc..

Parmi les différents modules comprises dans la bibliothèque Qt, il en existe deux essentiels qui seront systématiquement inclus dans tout projet Qt de base :

- QtCore : fournit les principales classes et fonctionnalités de Qt servant dans n'importe quel projet, doté d'une interface graphique ou non.
  (exemples de classes : QObject, QApplication, QDebug, QFile, QTimer...)

- QtGui : fournit des classes et fonctionnalités principales de tout projet comprenant une interface graphique.
  (exemples de classes : QWidget, QDialog, ...)

Ce type d'interface est donc ici souhaitable voire demandé dans le cadre de ce projet car il permet de créer des fenêtres pouvant émuler les différentes fonctionnalités d'un pupitre ou d'une console servant à manipuler les projecteurs ; mais étant donné qu'il est également question de programmation, on doit pouvoir être également capable de développer une interface qui puisse également être capable de gérer des paramètres bien plus complexes et générer des scènes, séquences et spectacles.

QT Creator (Environnement de développement intégré)

Qt Creator est l'environnement de développement intégré dédié à Qt et facilite la gestion d'un projet Qt.

Il possède une coloration syntaxique adaptées aux termes du *framework* Qt et également un débogueur intégré; si besoin est, il est également possible d'utiliser l'outil Qt Designer pour pouvoir concevoir le programme et son interface à partir d'une interface graphique.

XML (Langage de balisage Extensible)

XML (*eXtensible Markup Language*) est un métalangage utilisant un système de balisage, lesdites balises sont ainsi nommées et peuvent contenir une valeur spécifique mais aussi des attributs ayant leur valeur et permettant ainsi de mieux détailler ces balises selon le contexte. Il est également possible qu'une balise contienne des éléments, qui sont d'autres balises ayant les caractéristiques citées ci dessus.

Un document XML est composé d'éléments désignés par des balises et structuré sous la forme d'un arbre avec un et un seul élément racine (*root*). Les éléments sont aussi appelés noeuds ou *nodes*. En programmation, le DOM (*Document Object Model*) permet de représenter la structure d'un document et de ses éléments sous forme d'un arbre.

Nous avons donc besoin de disposer de fichiers renseignés avec ce langage dans le but de répertorier et sauvegarder nos scènes, séquences, spectacles et projecteurs afin de pouvoir les réutiliser et les gérer comme bon nous semble depuis l'application Qt.

Liste des fichiers XML de l'application :

- adaptateurs.xml
- appareils.xml
- consoles.xml
- scenes.xml
- sequences.xml
- spectacles.xml

Exemple :

```
<?xml version='1.0' encoding='UTF-8'?>
<appareils nbAppareils="6">
  <appareil nom="Lyre1" nbCanaux="6" type="LYRE" uuid="{6bd378fd-51ba-4657-8526-1965d346261d}">
        <canal id="1">Shutter</canal>
        <canal id="2">Gobo</canal>
        <canal id="3">Gobo rotation</canal>
        <canal id="4">Color</canal>
        <canal id="5">Pan</canal>
        <canal id="6">Tilt</canal>
  </appareil>
  ...
</appareils>
```

Qt fournit le module **QtXml** qui contient un ensemble de classes pour pouvoir ouvrir, fermer, lire et écrire des documents XML depuis notre application. Les classes principales sont :

- `QDomDocument` : classe qui représente un document XML
- `QDomElement` : classe qui représente un élément dans l'arborescence DOM
- `QDomNode` : classe de base pour tous les nœuds d'un arbre DOM

Pour accéder à un fichier XML, on utilisera un objet `QFile` puis on associera son contenu à un objet `QDomDocument`.

Exemple :

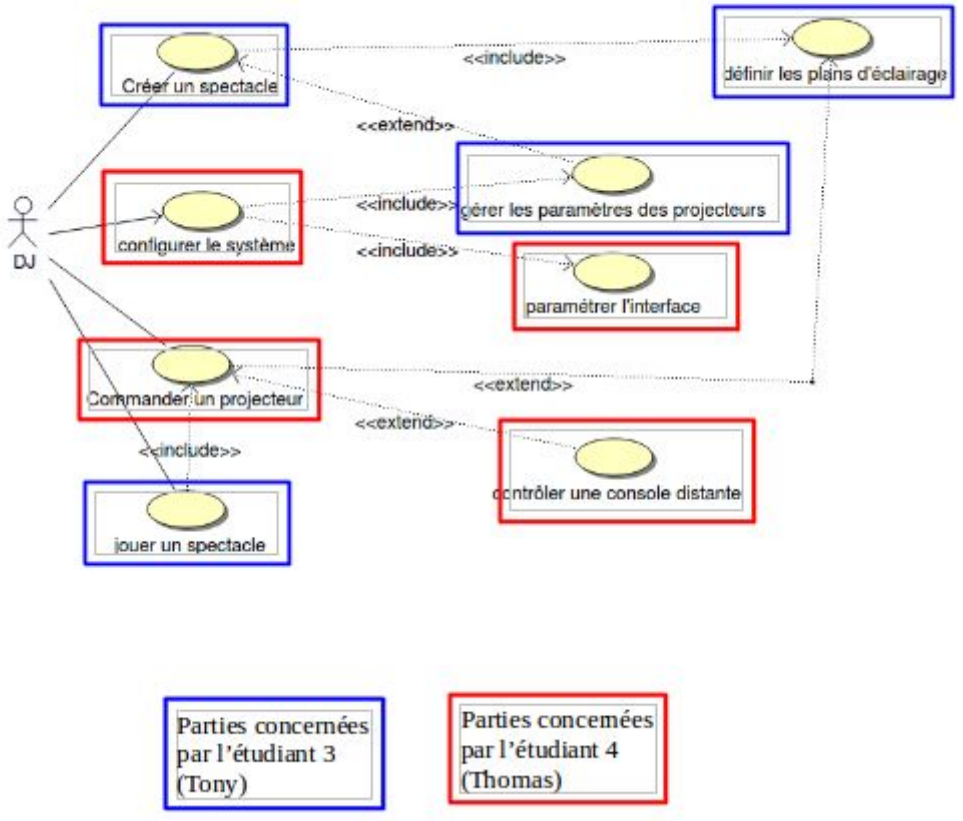
```
QFile fichierAppareils; // On instancie un objet QFile

fichierAppareils.setFileName("appareils.xml");  // On fait le lien entre l'objet
// fichierAppareils et le fichier appareils.xml

fichierAppareils.open(QIODevice::ReadOnly); // On ouvre le fichier en lecture
// seule

QDomDocument docAppareils; // On instancie un objet QDomDocument

docAppareils.setContent(&fichierAppareils, false); // On attribue le contenu du
// fichier appareils.xml à ce document afin de pouvoir manipuler ce fichier
depuis // le code

QDomElement root = docAppareils.documentElement(); // <appareils
nbAppareils="6">
…
fichierAppareils.close();
```

Conception

Cas d'utilisations (vue générale)

Le logiciel doit proposer les fonctionnalités suivantes :

- La possibilité de créer des spectacles en emboîtant des séquences composées de différentes scènes pour ensuite les exécuter.
    - La création de spectacle impliquera donc l'usage de projecteurs répertoriés dans la mémoire du logiciel.
- L'interface de paramétrage des projecteurs, de la chaîne DMX et des interfaces USB DMX.
    - Il sera donc possible de répertorier, modifier et supprimer des projecteurs dans le logiciel mais aussi de choisir l'interface USB servant à l'envoi des trames vers la chaîne DMX.
- Le menu de commande des projecteurs de la chaîne depuis le logiciel ou un pupitre distant (ici étant une console Enttec Playback Wing)

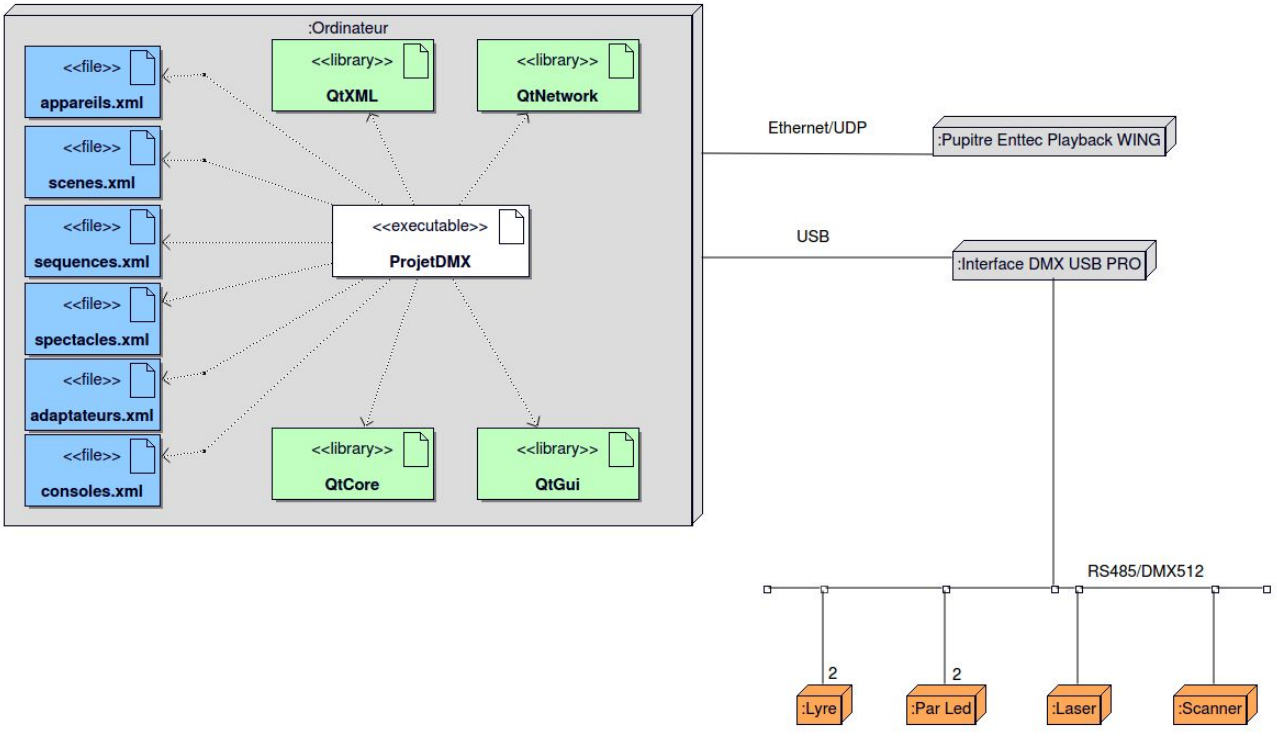


Vue de déploiement de la chaîne DMX

L'exécutable **ProjetDMX** dépend donc des six fichiers XML et de quatre bibliothèques Qt pour assurer son fonctionnement.

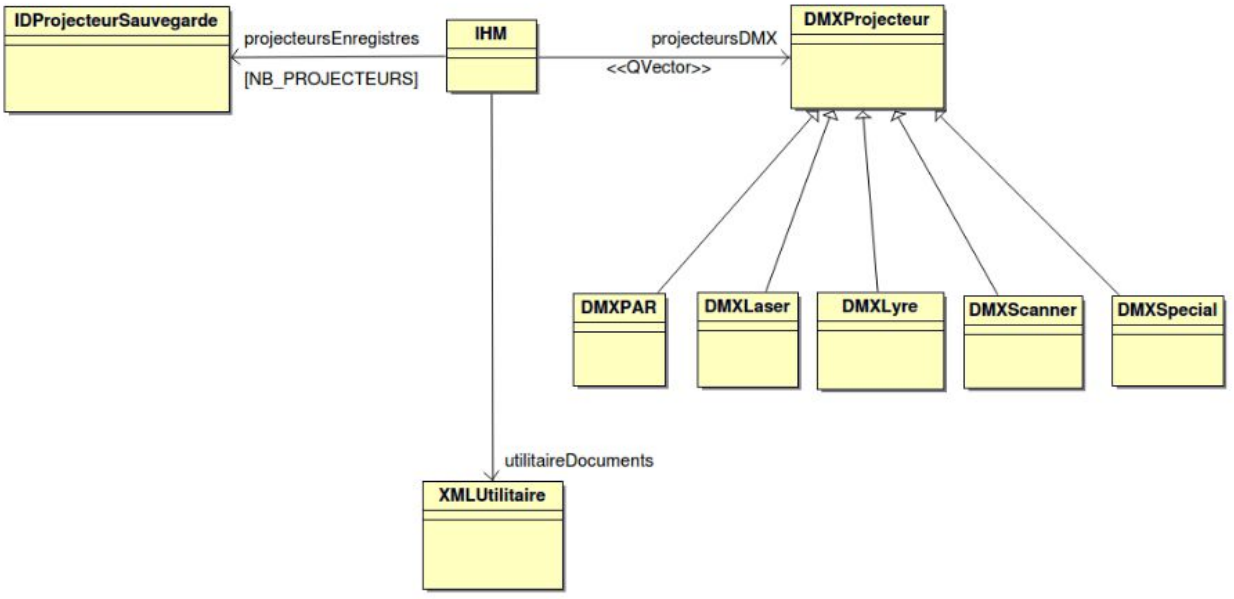
La console Playback Wing est reliée au réseau local par Ethernet en suivant le protocole UDP pour transmettre ses trames.

La topologie de câblage des projecteurs DMX est en Bus, elle est reliée à l'ordinateur par le biais de l'**interface DMX USB PRO** qui retranscrit les messages USB en DMX512 et inversement.



Architecture logicielle

ProjetDMX comprend un ensemble de classes étant chacune liées par différentes relations.



- **IHM**: Cette classe gère l'interface Homme-Machine, elle inclut donc les différentes fonctionnalités Qt d'interface graphique (Widgets, boutons, fenêtres, etc..)

- **DMXProjecteur**: Classe générale représentant tout appareil DMX de la chaîne sous forme d'objet lors des interactions avec cette dernière depuis le logiciel.

  Cette classe est pourvue de plusieurs spécialisations telles que **DMXPAR**, **DMXLaser**, **DMXLyre** et **DMXScanner**; elles représentent toutes un type de projecteur DMX existant.

  Une classe fille du nom de **DMXSpecial** a été également prévue pour tout appareil DMX de type inconnu ou non conventionnel tel que le nouveau projecteur en cours de conception durant ce projet (Partie EC).

  la classe **DMXProjecteur** est dans une relation d'agrégation/association avec **IHM** et transmet à cette dernière un conteneur QVector afin d'avoir accès à un moyen d'organiser les projecteurs DMX dans une chaîne.

- **XMLUtilitaire**: Toute méthode ayant recours à des fonctionnalités de QtXml appartient à cette classe.

  Cette classe est en relation d'association avec **IHM** et transmet un seul objet faisant office d'utilitaire pour qu'il soit possible de manipuler les fichiers xml et y faire des sauvegardes d'écriture.

- **IDProjecteurSauvegarde**: Cette classe représente un ensemble de widgets permettant d'accéder à divers paramètres d'un projecteur ayant été répertorié dans la chaîne dans le logiciel.

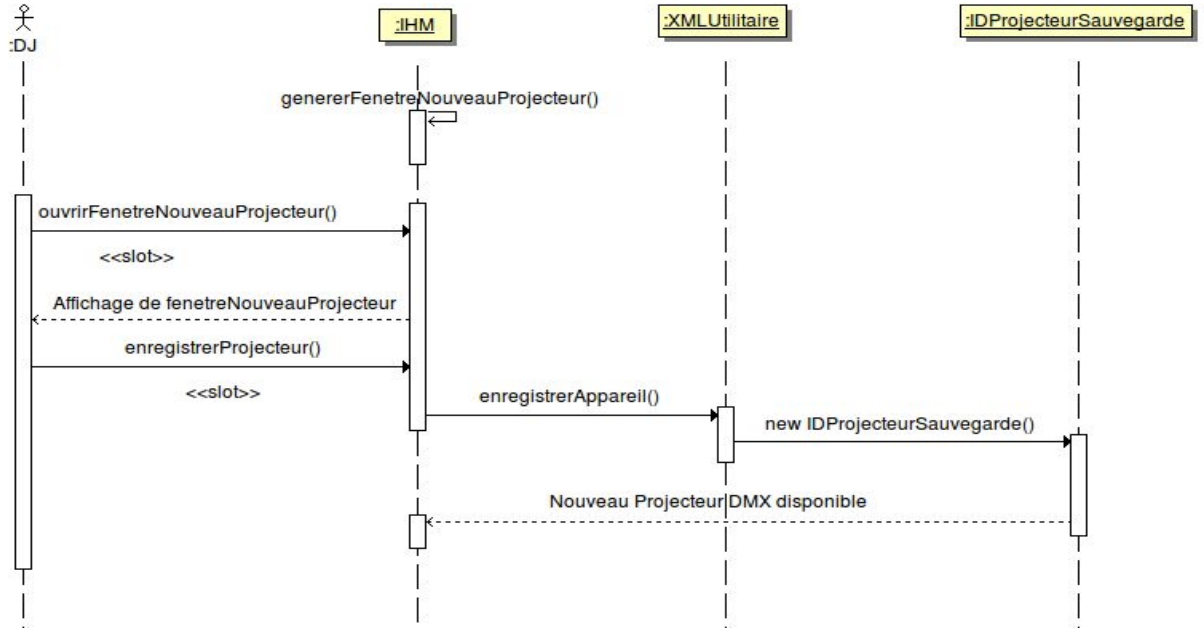
  un conteneur d'objets de cette classe apparaît donc dans **IHM** étant donné qu'une relation d'agrégation/association existe entre cette dernière et **IDProjecteurSauvegarde.**

Scénarii

L'ajout d'un projecteur à la chaîne DMX :

Depuis cette interface, nous pouvons interagir avec les paramètres des projecteurs DMX enregistrés, il est donc possible de modifier le nombre de canaux et les noms des canaux du projecteur sélectionné, de le supprimer ou de voir ses informations.

Nous pouvons également ajouter un projecteur à la chaîne depuis ce menu.



La méthode `genererFenetreNouveauProjecteur()` crée un objet `fenetreNouveauProjecteur` à partir de la classe `QDialog` :


```
void IHM::genererFenetreNouveauProjecteur() {
    // crée un objet de type QDialog
    fenetreNouveauProjecteur = new QDialog;
    fenetreNouveauProjecteur->setWindowTitle("Nouvel Appareil DMX");

    // crée les widgets
    labelNom = new QLabel(QString::fromUtf8("Nom :"), this);
    choixNom = new QLineEdit();
    labelType = new QLabel(QString::fromUtf8("Type :"), this);
    choixTypes = new QComboBox();
    choixTypes->addItem("PAR LED");
    choixTypes->addItem("LYRE");
    choixTypes->addItem("SCANNER");
    choixTypes->addItem("LASER");
    choixTypes->addItem("AUTRES");
    labelNBCanaux = new QLabel(QString::fromUtf8("Nombre de Canaux :"), this);
    choixNBCanaux = new QSpinBox;
    choixNBCanaux->setMinimum(1);
    choixNBCanaux->setMaximum(10);
    labelCanalDepart = new QLabel(QString::fromUtf8("Canal de Depart :"), this);
    choixCanalDepart = new QSpinBox;
    choixCanalDepart->setMinimum(1);
    choixCanalDepart->setMaximum(512);
    labelNomCanaux = new QLabel(QString::fromUtf8("Noms des Canaux :"), this);
    for(int i = 0; i < 10; i++)
        nomsCanaux[i] = new QLineEdit;
    boutonEnregistrer = new QPushButton(QString::fromUtf8("Enregistrer"), this);

    // positionne les widgets dans les layouts
    QVBoxLayout *mainLayoutFenetre = new QVBoxLayout;
    QHBoxLayout *layoutEnregistrementNom = new QHBoxLayout;
    QHBoxLayout *layoutEnregistrementType = new QHBoxLayout;
    QHBoxLayout *layoutEnregistrementParametresCanaux = new QHBoxLayout;
    QVBoxLayout *layoutEnregistrementNomsCanaux = new QVBoxLayout;
    layoutEnregistrementNom->addWidget(labelNom);
    layoutEnregistrementNom->addWidget(choixNom);
    layoutEnregistrementType->addWidget(labelType);
    layoutEnregistrementType->addWidget(choixTypes);
    layoutEnregistrementParametresCanaux->addWidget(labelNBCanaux);
    layoutEnregistrementParametresCanaux->addWidget(choixNBCanaux);
    layoutEnregistrementParametresCanaux->addWidget(labelCanalDepart);
    layoutEnregistrementParametresCanaux->addWidget(choixCanalDepart);
    layoutEnregistrementNomsCanaux->addWidget(labelNomCanaux);
    for(int i = 0; i < 10; i++)
    {
        layoutEnregistrementNomsCanaux->addWidget(nomsCanaux[i]);
        connect(choixNBCanaux,SIGNAL(valueChanged(int)),this,SLOT(bloquerChoixNomCanaux(int)));
    }
    mainLayoutFenetre->addLayout(layoutEnregistrementNom);
    mainLayoutFenetre->addLayout(layoutEnregistrementType);
    mainLayoutFenetre->addLayout(layoutEnregistrementParametresCanaux);
    mainLayoutFenetre->addLayout(layoutEnregistrementNomsCanaux);
    mainLayoutFenetre->addWidget(boutonEnregistrer);
    fenetreNouveauProjecteur->setLayout(mainLayoutFenetre);
}
```

Lorsque le DJ clique sur le bouton “Nouveau Projecteur”, cela déclenche l’exécution du *slot* `ouvrirFenetreNouveauProjecteur()`. Cette méthode affichera la fenêtre ci-dessus :

```
void IHM::ouvrirFenetreNouveauProjecteur()
{
        // réinitialise la fenêtre
        fenetreNouveauProjecteur->setWindowTitle("Nouvel Appareil DMX");
        choixNom->setText("");
        choixTypes->setCurrentIndex(0);
        choixNBCanaux->setValue(0);
        choixCanalDepart->setValue(0);
        for(int i = 0; i < 10; i++)
        {
           nomsCanaux[i]->setText("");
        }
        bloquerChoixNomCanaux(choixNBCanaux->value());
        // affiche la fenêtre
        fenetreNouveauProjecteur->show();
}
```

Le DJ saisit les différents paramètres du projecteur et clique sur “Enregistrer”. Cela déclenche le slot `enregistrerProjecteur()`. Cette méthode a la responsabilité d’appeler la méthode `enregistrerAppareil()` de la classe `XMLUtilitaire`. `enregistrerAppareil()` va procéder à l'enregistrement dans un fichier intitulé « appareils.xml » .

Ce fichier répertorie chaque projecteur enregistré sous la forme d'un élément ou nœud des contenant lui même d'autres éléments représentant les canaux de ce projecteur avec leurs valeurs de nom respectives ainsi que des attributs nous procurant des caractéristiques globales dudit projecteur qui auront été définis directement à partir des paramètres choisis durant la création du projecteur.

Parmi ces attributs, nous avons le champ « uuid » qui permet d’identifier avec un numéro unique le projecteur. C’est important lorsqu’il faudra appeler un projecteur bien précis depuis le fichier XML.

La méthode `enregistrerAppareil()` retourne vrai si l'enregistrement a réussi (sinon faux) et reçoit en paramètres :

- nom (QString) : le nom de l'appareil
- type (QString) : le type d'appareil
- nbCanaux (int) : le nombre de canaux pour l'appareil
- canalDepart (int) : le numéro de canal de départ de l'appareil
- nomsCanaux[ ] (QString) : un tableau contenant lesz libellés des canaux de l'appareil

```
bool XMLUtilitaire::enregistrerAppareil(QString nom, QString type, int nbCanaux, int canalDepart,
QString nomsCanaux[])
{
        // ouvre le fichier XML
        if(!fichierAppareils.open(QIODevice::ReadWrite))
        {
             qDebug() << Q_FUNC_INFO << "Erreur ouverture" << fichierAppareils.fileName();
             return false;
        }
        else
        {
               QUuid monUuid = QUuid::createUuid();
               QDomDocument documentAppareils;
               documentAppareils.setContent(&fichierAppareils, false);
               QDomElement root = documentAppareils.documentElement();
             // crée un élément appareil
               QDomElement nouvelAppareil = documentAppareils.createElement("appareil");
             // on fixe ses attributs
               nouvelAppareil.setAttribute("nom", nom);
               nouvelAppareil.setAttribute("nbCanaux", nbCanaux);
               nouvelAppareil.setAttribute("type", type);
               nouvelAppareil.setAttribute("uuid", monUuid.toString());
             // crée les éléments canal pour l'élément appareil
               QDomElement nouveauxCanaux[10];
               QDomText texteNomCanal;
               for(int i = 0; i < nbCanaux; i++ )
               {
                       nouveauxCanaux[i] = documentAppareils.createElement("canal");
                       nouveauxCanaux[i].setAttribute("id", canalDepart+i);
                       texteNomCanal = documentAppareils.createTextNode(nomsCanaux[i]);
                       nouveauxCanaux[i].appendChild(texteNomCanal);
                       nouvelAppareil.appendChild(nouveauxCanaux[i]);
               }
             // ajoute l'élément appareil à l'arbre
               root.appendChild(nouvelAppareil);
             // modifie le nombre d'appareils
               int nouveauNbAppareils = root.attribute("nbAppareils").toInt() + 1 ;
               root.setAttribute("nbAppareils", nouveauNbAppareils);
             // écriture dans le fichier XML
               QTextStream out(&fichierAppareils);
               fichierAppareils.resize(0); // supprime le contenu précédent
               documentAppareils.save(out, 2); // enregistre le document dans le fichier
               fichierAppareils.close(); // ferme le fichier XML
               return true;
        }
}
```

Les commandes de pilotage des projecteurs enregistrés :

Cette partie de l'IHM permet de manipuler des gradateurs pour altérer les valeurs des canaux ainsi que des spinbox qui, eux, permettent de choisir les canaux à modifier.

Il est notable que cette partie de l'application a été conçue pour pouvoir alterner entre la prise de contrôle des canaux par cette dernière et par la console physique Wing soit via le combobox “Pilotage :” qui propose les choix “Pilotage IHM” et “Pilotage Console” , soit par les boutons GO (prendre la main) et BACK (redonner le contrôle à l'application pc) de la console Wing .

Si l'on sélectionne depuis le combobox "Projecteur :" un des appareils répertoriés dans la chaîne DMX, les sliders servant à modifier les valeurs des canaux prendront alors les numéros et les noms des canaux correspondant à ceux du projecteur choisi.

```
- <appareil nom="Lyre2" nbCanaux="5" type="LYRE" uuid="{2f00720e-1c30-44d2-867f-00a3d2825d3e}">
    <canal id="65">Vibration</canal>
    <canal id="66">Pattern</canal>
    <canal id="67">Couleur</canal>
    <canal id="68">PAN</canal>
    <canal id="69">TILT</canal>
  </appareil>
```

La récupération de ces informations se fait indirectement depuis le fichier “appareils.xml” par le biais du vecteur **projecteursDMX** qui met à jour son contenu systématiquement dès lors qu’un nouveau projecteur était ajouté

```
void IHM::enregistrerProjecteur(/*QString uuid*/)
{
        [...]
        utilitaireDocuments->lireAppareils(projecteursDMX); // On utilise l'utilitaire XML pour // récuperer les valeurs et noms des canaux de chaque appareil qui sera ensuite // placé en tant qu'objet DMXProjecteur dans le vecteur.

        [...]
}


bool XMLUtilitaire::lireAppareils(QVector<DMXProjecteur*> &projecteursDMX)
{
        if (!fichierAppareils.open(QIODevice::ReadOnly))
        {
                [...]
        }
        else
        {
                // Vider le conteneur avant de le remplir
                for(int i = 0; i < projecteursDMX.count(); i++)
                {
                        delete projecteursDMX.at(i);
                }
                projecteursDMX.clear();

                QDomDocument documentXML;

                documentXML.setContent(&fichierAppareils, false);
                QDomElement racine = documentXML.documentElement(); // <appareils>
                if(racine.isNull())
                {
                        qDebug() << Q_FUNC_INFO << "Erreur racine !";
                        fichierAppareils.close();
                        return false;
                }

                QDomNode noeudAppareil = racine.firstChild();
                if(noeudAppareil.isNull())
                {
                        qDebug() << Q_FUNC_INFO << "Erreur racine vide !";
                        fichierAppareils.close();
```

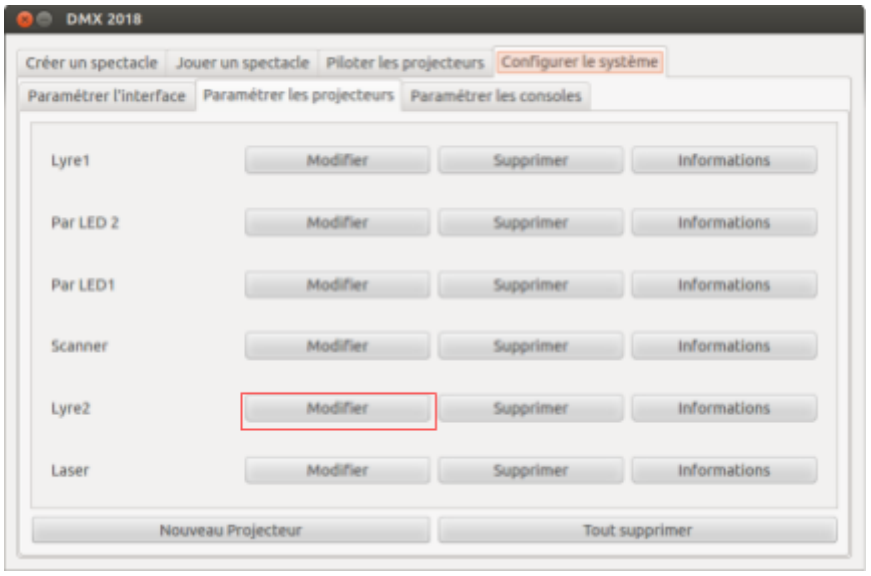
```
                        return false;
                }

                while(!noeudAppareil.isNull())
                {
                        QDomElement elementAppareil = noeudAppareil.toElement(); // <appareil nom="" nbCanal=""
type="" uuid="">
                        if(elementAppareil.isNull())
                        {
                                qDebug() << Q_FUNC_INFO << "Erreur element !";
                                break;
                        }

                        DMXProjecteur* projecteurDMX = creerAppareil(elementAppareil);

                        QDomNode noeudCanal = elementAppareil.firstChild();
                        if(!noeudCanal.isNull())
                        {
                                int canalDebut = 1;
                                QStringList nomCanaux;
                                int i = 0;
                                while(!noeudCanal.isNull())
                                {
                                        QDomElement elementCanal = noeudCanal.toElement(); // <canal
id="">XXX</canal>

                                        // Lit les attributs du canal
                                        //qDebug() << Q_FUNC_INFO << elementCanal.attribute("id").toInt();
                                        if(i == 0)
                                                canalDebut = elementCanal.attribute("id").toInt();
                                        // Lit l'élément du canal
                                        //qDebug() << Q_FUNC_INFO << elementCanal.text();
                                        nomCanaux << elementCanal.text();

                                        noeudCanal = noeudCanal.nextSibling();
                                        ++i;
                                }
                                projecteurDMX->setCanalDebut(canalDebut);
                                projecteurDMX->setNomCanaux(nomCanaux);
                        }

                        projecteursDMX.push_back(projecteurDMX);
                        noeudAppareil = noeudAppareil.nextSibling();
                }
        }
        fichierAppareils.close();
        return true;
}
```

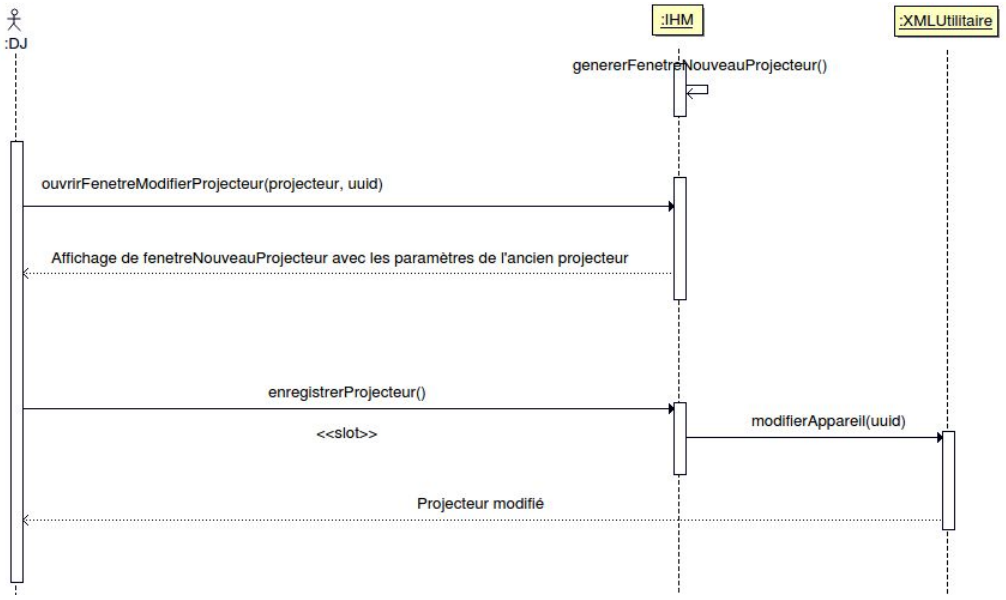
La modification d'un projecteur de la chaîne DMX :

Il est possible de modifier un appareil choisi depuis l'IHM.

Pour pouvoir retrouver ledit appareil dans appareils.xml, nous allons utiliser son uuid.

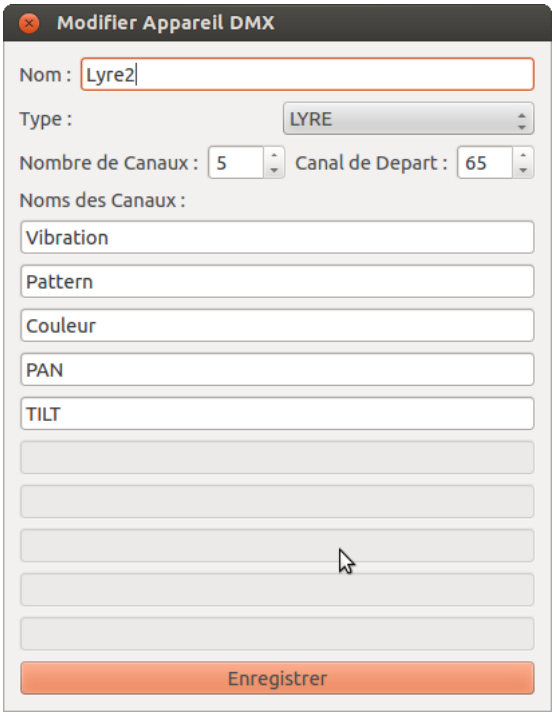
(NB : l'uuid est crée procéduralement à l'enregistrement de chaque nouvel appareil, ce dernier est inchangeable et unique à chaque appareil, il permet donc d'identifier ce dernier et de tolérer d'éventuels doublons dans les noms affichés des appareils)





Lorsque l'on clique sur le bouton "modifier" d'un appareil DMX de son choix, le slot ouvrirFenetreModifierProjecteur est enclenché.
De ce fait, "fenetreNouveauProjecteur" s'ouvre en étant modifiée, son titre change et devient "Modifier Appareil DMX" et les paramètres du projecteur choisi remplissent les cases vides servant à les définir.

```
void IHM::ouvrirFenetreModifierProjecteur(QString projecteur, QString uuid)
{
	fenetreNouveauProjecteur->setWindowTitle("Modifier Appareil DMX"); // on change le titre
	choixNom->setText(projecteur);
	for(int i = 0; i < projecteursDMX.size(); i++)
	{
	// on cherche un objet du conteneur de DMXProjecteur dont l'attribut uuid a la même valeur que
	// l'argument uuid de la méthode
		if(projecteursDMX[i]->getUuid() == uuid)
		{
			// on règle le combobox choixType avec le type du projecteur choisi par défaut
			if(projecteursDMX[i]->getType() == "PAR LED")
			choixTypes->setCurrentIndex(0);
			else if(projecteursDMX[i]->getType() == "LYRE")
			choixTypes->setCurrentIndex(1);
			else if(projecteursDMX[i]->getType() == "SCANNER")
				choixTypes->setCurrentIndex(2);
			else if(projecteursDMX[i]->getType() == "LASER")
				choixTypes->setCurrentIndex(3);
			else
				choixTypes->setCurrentIndex(4);

			// on ajoute les valeurs par défaut pour choixNBCanaux et choixCanalDepart
			choixNBCanaux->setValue(projecteursDMX[i]->getNomCanaux().size());
			choixCanalDepart->setValue(projecteursDMX[i]->getCanalDebut());


			DMXProjecteur *projecteurDmx = projecteursDMX[i];

			// on ajoute les noms des canaux correspondant et on grise les cases en trop
			for(int i = 0; i < projecteurDmx->getNomCanaux().size() ; i++)
			{
				nomsCanaux[i]->setText(projecteurDmx->getNomCanaux().at(i));
			}
			bloquerChoixNomCanaux(choixNBCanaux->value());

		}
	}
	fenetreNouveauProjecteur->show();

}
```



Pour appliquer la modification, il suffira de cliquer sur le bouton Enregistrer, ce qui actionnera le slot enregistrerProjecteur(); qui n'appellera non pas , par la suite, enregistrerAppareil() mais modifierAppareil(uuid) de la classe **XMLUtilitaire**.

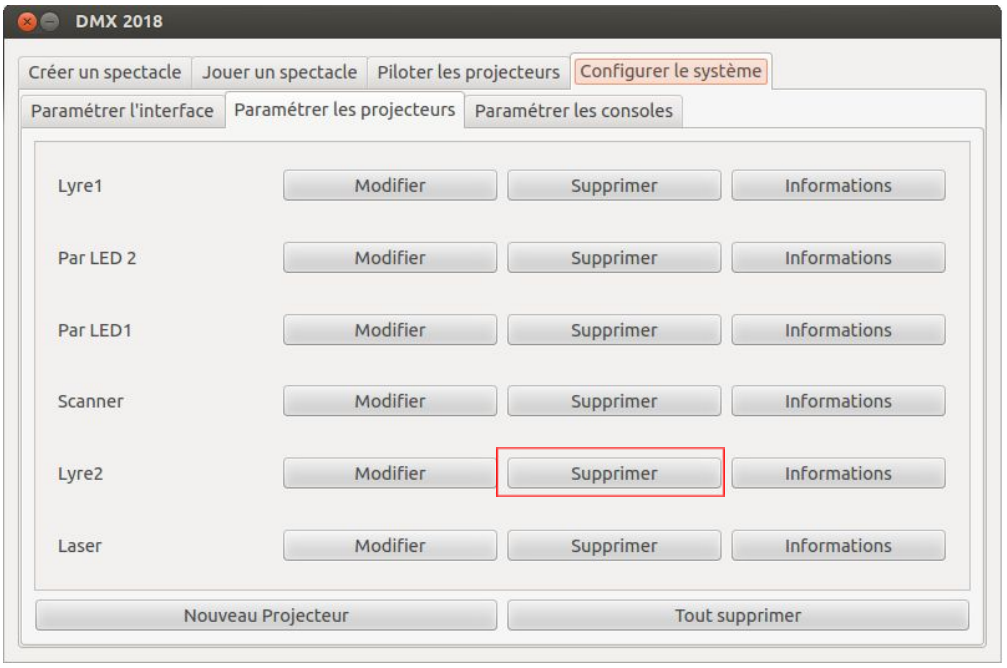
L'uuid du projecteur selectionné est donc passé en argument et aura pour utilité de retrouver le bon élément dans appareils.xml afin de modifier ce dernier.

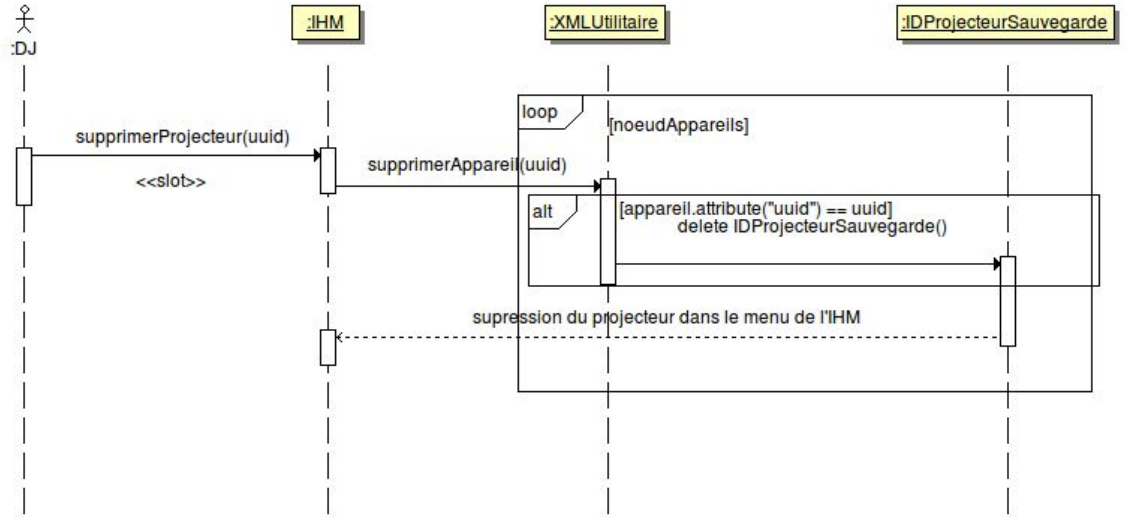
```
void IHM::enregistrerProjecteur(/*QString uuid*/)
{
        [...]

        if(fenetreNouveauProjecteur->windowTitle() == "Nouvel Appareil DMX" )
        [...]

        else if(fenetreNouveauProjecteur->windowTitle() == "Modifier Appareil DMX" )
        // si on passe par la fenetre de modification, alors on appelle la méthode modifier
        // Appareil et on passe l'uuid en plus des autres valeurs en argument
        utilitaireDocuments->modifierAppareil(cacheUuid ,choixNom->text(), choixTypes->currentText(),
choixNBCanaux->value(), choixCanalDepart->value(), nomsCanauxEnregistres);

        [...]
}
```

```
bool XMLUtilitaire::modifierAppareil(QString uuid, QString nom, QString type, int nbCanaux, int
canalDepart, QString nomsCanaux[])
{

        if(!fichierAppareils.open(QIODevice::ReadWrite))
        {
        [...]
        }
        else
        {

        QDomDocument documentAppareils;
        documentAppareils.setContent(&fichierAppareils, false);
        QDomElement root = documentAppareils.documentElement();
        QDomNode noeudAppareils = root.firstChild();
        while(!noeudAppareils.isNull())
        {

                QDomElement appareil = noeudAppareils.toElement();
                QDomText texteNomCanal;
                QDomElement nouveauxCanaux[10];
                if(appareil.attribute("uuid") == uuid)
                {
                // dès que les valeurs de l'argument uuid de la méthode et de l'attribut uuid de  //
                l'élement sont égales, on change tous les autres attributs avec les valeurs de //
                leurs équivalents en arguments de la méthode
                appareil.setAttribute("nom", nom);
                appareil.setAttribute("nbCanaux", nbCanaux);
                appareil.setAttribute("type", type);

                QDomNode noeudCanaux = appareil.firstChild();

                while(!noeudCanaux.isNull())
                {
                        // on supprime les anciens canaux avant de mettre à jour
                        if(!noeudCanaux.nextSibling().isNull())
                        {

                        noeudCanaux = noeudCanaux.nextSibling();
                        appareil.removeChild(noeudCanaux.previousSibling());
                        }
                        else
                        {

                        appareil.removeChild(noeudCanaux);
                        noeudCanaux = noeudCanaux.nextSibling();
                        }
                }

                for(int i = 0; i < nbCanaux; i++ )
                {
                        // on ajoute les nouveaux canaux de l'appareil
                        nouveauxCanaux[i] = documentAppareils.createElement("canal");
                        nouveauxCanaux[i].setAttribute("id", canalDepart+i);
                        texteNomCanal = documentAppareils.createTextNode(nomsCanaux[i]);
                        nouveauxCanaux[i].appendChild(texteNomCanal);
                        appareil.appendChild(nouveauxCanaux[i]);
                }

                QTextStream out(&fichierAppareils);
                fichierAppareils.resize(0);
                documentAppareils.save(out, 2);
                fichierAppareils.close();
                return true;
                }
                noeudAppareils = noeudAppareils.nextSibling();
        }
    [...]
        }
}
```

La suppression d'un projecteur de la chaîne DMX :

Nous pouvons, grâce à l'uuid des projecteurs répertoriés dans le fichier XML, procéder à une suppression ciblée d'un projecteur ayant été sélectionné.





Dès que le DJ clique sur le bouton “Supprimer” d’un des projecteurs , le slot `supprimerProjecteur(uuid)` de la classe **IHM** est actionné, ce dernier appelle ainsi la méthode `supprimerAppareil(uuid)` de la classe **XMLUtilitaire**, son argument “uuid” récupère la valeur argument “uuid” de la méthode `supprimerProjecteur(uuid)` ayant lui même la valeur de l’attribut uuid de l’objet **IDProjecteurSauvegarde** qui aura été transmis à partir du signal, cet argument se retrouve donc comparé à l’attribut “uuid” de de chaque élément “Appareil” dans le fichier appareils.xml .

```
[...]
connect(projecteursEnregistres[i], SIGNAL(suppression(QString)), this,
SLOT(supprimerProjecteur(QString)));
[...]
```

Dès que l’argument uuid de la méthode et l’attribut uuid d’une balise Appareil sont identiques, ladite balise est dès lors supprimée du fichier et le projecteur sélectionné disparaît de l’IHM .

```
void IHM::supprimerProjecteur(QString uuid)
{
        qDebug() << Q_FUNC_INFO;
        // on efface d’abord l’IHM avant de procéder aux modification
        effacerAffichageProjecteursEnregistres();
        effacerAffichageProjecteursPilotage();

        utilitaireDocuments->supprimerAppareil(uuid); // on apelle la méthode provenant de
        XMLUtilitaire

        // pour supprimer l’appareil choisi du fichier appareils.xml
        utilitaireDocuments->afficherProjecteursEnregistres(projecteursEnregistres); // on met à jour
        l’IHM

        utilitaireDocuments->lireAppareils(projecteursDMX);
        mettreAJourProjecteursPilotage();
        mettreAJourProjecteursEnregistres();

}

bool XMLUtilitaire::supprimerAppareil(QString uuid)
{
        qDebug() << Q_FUNC_INFO;
        // on tente d’ouvrir le fichier en lecture/ecriture puis on vérifie si l’ouverture a été
accomplie
```

```
	if(!fichierAppareils.open(QIODevice::ReadWrite))
	{



		// si le fichier refuse de s'ouvrir, la méthode s'arrête ici et retourne la
		// valeur "false"
		qDebug() << Q_FUNC_INFO << "Erreur ouverture" << fichierAppareils.fileName();
		return false;
	}
	else
	{
		// si le fichier est bel et bien ouvert, le programme suit son cours
		QDomDocument documentAppareils;
		documentAppareils.setContent(&fichierAppareils, false); //on assigne le contenu
		// du document à celui du fichier appareils.xml
		QDomElement root = documentAppareils.documentElement(); // on affecte l'element //
		Appareils à root
		QDomNode noeudAppareils = root.firstChild();
		while(!noeudAppareils.isNull()) // boucle pour lire chaque élement enfantde Appareils
		{
				QDomElement appareil = noeudAppareils.toElement();
				if(appareil.attribute("uuid") == uuid)
				{
				// si l'attribut uuid de l'élement a une valeur égale à l'argument
				// uuid de la méthode, alors cet élement sera supprimé du fichier
					root.removeChild(noeudAppareils);
					int nouveauNbAppareils = root.attribute("nbAppareils").toInt()
- 1;

					root.setAttribute("nbAppareils", nouveauNbAppareils);
					QTextStream out(&fichierAppareils);
					fichierAppareils.resize(0);
					documentAppareils.save(out, 2);
					fichierAppareils.close();
					return true;
				}
				// sinon on passe à l'élément enfant suivant
				noeudAppareils = noeudAppareils.nextSibling();
		}
		qDebug() << Q_FUNC_INFO << "Erreur supression !" ;
		fichierAppareils.close();
		return false;
	}
}
```

Tests

DESCRIPTION	Oui	En cours	Non
La création d'un nouveau spectacle est possible			X
La création d'une scène est opérationnelle		**X**	
L'exécution d'une scène est fonctionnelle		**X**	
La création d'une séquence est opérationnelle			X
L'exécution d'une séquence est fonctionnelle			X
L'ajout d'un projecteur est réalisable à partir de l'IHM	**X**		
Les fichiers XML existent et sont correctement renseignés	**X**		

Conclusion

Il est donc possible à ce jour de pouvoir ajouter, modifier et/ou supprimer des projecteurs de la chaîne DMX.

Le système d'ajout et de paramétrage de scènes est bientôt achevé. Après, il faut également créer l'interface et les fonctionnalités pour les séquences et les spectacles.

De nouvelles connaissances telles que ce qui concerne les fichiers XML et leur intégration dans un programme Qt ainsi qu'une plus grande maîtrise de l'outil Qt dans sa globalité auront été acquises.

Glossaire

Adresse IP : adresse constituée d'une série de nombres en suivant le protocole IP (couche 3 OSI) et qui permet à un poste de s'identifier et communiquer sur un réseau de postes informatiques.

Attribut (XML) : Valeur servant à identifier un élément XML (exemple: un nom donné ou son Uuid)

Broadcast (Adresse IP) : adresse IP d'un réseau dont les communications sont destinées à tous les autres postes appartenant à ce même réseau

Canal (DMX): ensemble de 9 bits compris dans une trame DMX en 512 exemplaires, chaque canal définit un paramètre spécifique d'un appareil pouvant recevoir des données DMX512

DMX512 : norme protocolaire universelle servant aux pilotages et paramétrages de spots lumineux

Elément (XML) : représentation de la totalité d'une balise, comprenant ses attributs et sa valeur

Gobo : masque ayant un motif par lequel la lumière passe pour donner une forme à cette dernière

Pan : socle motorisé du projecteur permettant la rotation horizontale de ce dernier

Tilt : moteur permettant la rotation verticale d'un projecteur

IHM : Interface Homme-Machine

Laser (Projecteur) : Projecteur pouvant générer un faisceau laser pouvant se déplacer de façon à créer des images point par point

Lyre (Projecteur) : Projecteur doté d'une rotation verticale et horizontale, il peut également changer de « masque » (gobo) pour donner une image et/ou une couleur à la lumière projetée

PAR LED (Projecteur) : Projecteur dépourvu de rotations motorisées utilisant des LED reprenant les trois couleurs primaires (rouge vert bleu) pour générer une lumière prenant n'importe quelle couleur

Scanner (Projecteur) : Projecteur similaire à la Lyre mais dont la rotation verticale n'est pas motorisée, le paramètre « Tilt » étant remplacé par le déplacement d'un miroir dont l'angle déplace le faisceau lumineux

Scène : mise en application d'un plan d'éclairage des projecteurs DMX

Séquence : enchaînement de scènes ayant chacune une durée déterminée au préalable.

Spectacle : enchaînement de séquences.

Uuid : Universal unique identifier, identifiant ici pouvant être généré procéduralement sous forme d'une chaîne de caractères de façon à ce qu'il ne soit qu'en un seul exemplaire dans un même registre.

XML : eXtensible Markup Language, métalangage principalement utilisé pour faire intermédiaire d'échange de données par systèmes d'information interposées ou pour sauvegarder des données et des valeurs sous forme de « nœuds » .

Annexe

Pour plus d'informations, la documentation technique et le code source sont disponibles dans le répertoire Echanges du serveur.
Pour y accéder, il faut passer par cette arborescence via le NAS SMB :
“ echanges/Rapports-Projets-2018/DMX/Reynier “

DMX 2018

Revue Finale

Projet effectué par : Demont Thomas (IR)

version 1.0

La Salle Avignon

Sommaire

Présentation générale

De nos jours, les DJ et animateurs de soirée utilisent couramment un ordinateur portable à la fois pour diffuser la musique et pour gérer les différents jeux de lumières pour éclairer et animer la piste de « spectacle ».

Le contrôle de la lumière et des éclairages font parties des techniques essentielles de la représentation et de la mise en scène d'espace, cela à la fois sur des scènes de spectacle et lors de soirées d'animation.

Le DMX 512 est une norme destinée à faciliter le raccordement des projecteurs sur les consoles lumières. En définissant les contraintes techniques, le type de câble et les conditions d'utilisation, il permet de rendre compatible de nombreux produits.

Le DMX 512 est à ce jour le protocole le plus répandu et le plus universel car il permet tout ce qui est nécessaire en terme d'éclairage.

Un glossaire est disponible à la fin de ce document pour les termes techniques inhabituels.

Expression du besoin

La gestion d'une régie d'éclairage DMX se fait généralement avec une console DMX. Cependant, de plus en plus de responsables d'éclairage utilisent des interfaces informatiques leurs permettant de gérer leurs projecteurs (via des adaptateur USB/DMX souvent pour pouvoir utiliser des applications depuis un PC).

De plus, il devient courant de modifier tout ou une partie d'une scène d'éclairage en fonction de différents paramètres.

Il faut donc une application de supervision et de commande pour différents type de projecteurs, compatible avec la norme DMX 512. Le but de ce projet est de développer cette application.

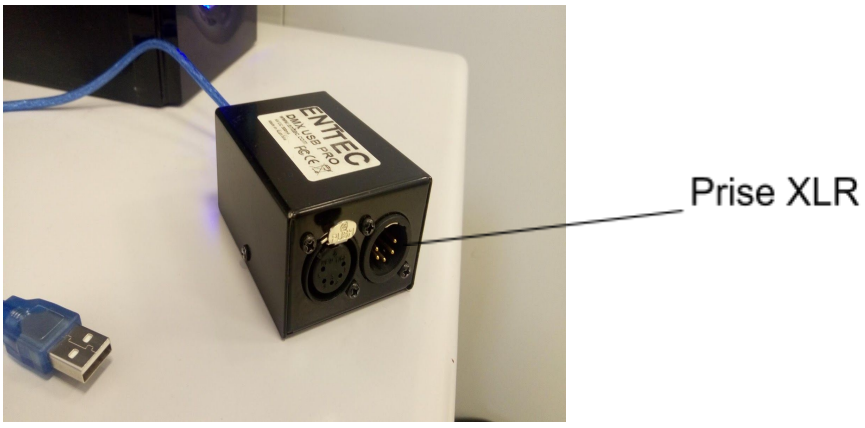
Spécification techniques

Pour réaliser ce projet, certaines spécifications techniques sont nécessaires.

DMX 512

Le protocole DMX512, basé sur la norme RS485(cadencée à 250 kb/s) permet de contrôler 512 canaux (9 bits d'adressage), chacun de ces canaux possède une valeur comprise entre 0 et 255 (8 bits de données par canal). Chaque appareil reçoit en même temps l'ensemble des 512 valeurs qui correspond donc à une trame DMX. La norme permet de contrôler un maximum de 32 appareils sur une même ligne DMX.

Les câbles respectant la norme DMX512 possèdent des prises XLR.



L'interface USB/DMX Pro

Nous utilisons une interface USB/DMX Pro de la marque Enttec ou OPEN DMX USB, aussi de la marque Enttec afin de pouvoir communiquer avec les projecteurs avec le protocole DMX 512 en passant par une connexion USB. Lors de ce projet, les interfaces sont attachées à un port, identifiées par un fichier situé à '/dev/nomDufichier'. Ce nom est 'ttyUSBn' 'n' étant le numéro de l'interface (si il y en a une seule ttyUSB1, 2 ttyUSB2...).

Les deux types d'interfaces (OPEN ou PRO) ne changent pas le fonctionnement de l'application, le seul changement est que les interfaces PRO permettent d'obtenir des informations supplémentaires sur elles mêmes (par exemple, la version de l'interface).

Dans notre cas, nous voulons avoir au maximum 32 projecteurs, cela correspond donc parfaitement à la norme DMX 512.

Console Playback Wing

La console Playback Wing dispose de 10 faders (des boutons coulissants aussi appelés sliders), de 40 boutons classiques et 4 autres touches de contrôle, Page Up, Page Down, Back et Go. Durant ce projet, seul les touches de contrôles et les faders sont utilisés. Les faders contrôlent les projecteurs directement et les touches de contrôle possèdent chacune des fonctionnalités:

-La touche ‘Page Up’ incrémente le canal DMX actif de 10.

- ‘Page Down’ décrémente le canal DMX actif de 10.

- ‘Back’ permet de passer le contrôle des projecteurs en mode console uniquement.

-‘Go’ permet de passer le contrôle des projecteurs en mode interface uniquement.

Des combinaisons de touches sont aussi disponibles :

-’Back’ + ‘Page Up’ permet de passer au projecteur suivant.

-’Back’ + ‘Page Down’ permet de passer au projecteur précédent.

La console Playback Wing permet d'envoyer des trames que nous nommerons « trame Wing » sur le réseau. Elle est reliée à ce dernier par câble Ethernet et envoie les trames Wing en broadcast UDP sur le réseau vers le port 3330.

Remarque : Il est aussi possible de se connecter avec la console en TCP, ainsi, la console passera en mode connectée et chaque message est acquitté, cependant, au bout de 3 seconde d'inactivité, la console repasse automatiquement en UDP (le port reste le 3330 pour UDP et TCP).

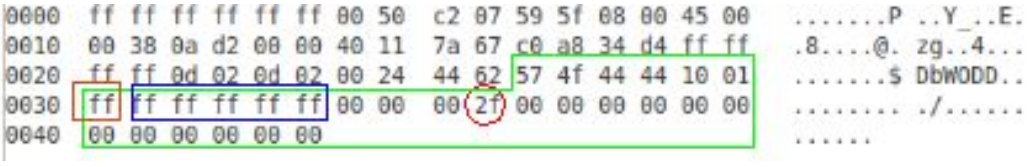
La trame Wing est formée de manière structurée, elle commence toujours par 4 octets donnant le type de message. Le type de message permet d'indiquer de quel type de trame il s'agit, il y en a 2 possibles : WODD ou WIDD. Le reste de la trame dépend du type de message.

Si le type de message est « WODD »,le message est une sortie de données depuis la console, ce type de message peut indiquer : une sortie de donnée classique, une sortie de donnée raccourcie / simplifiée ou une sortie de donnée du programme de la console. Ce projet n'utilise que la sortie classique qui est la sortie par défaut. Ici, chaque appui et relâchement d'une touche provoque l'envoi d'une trame décrite dans le tableau ci-dessous.

Taille en octets	Description
4	Type de message 'WODD'
1	Firmware de la console, 0 à 255
1	Flags Wing = 1
1	Bit 7: Page Up 0 = appuyé, 1 = relâché Bit 6 : Page Down 0 = appuyé, 1 = relâché Bit 5 : Back 0 = appuyé, 1 = relâché Bit 4 : Go 0 = appuyé, 1 = relâché
1	Touches classiques, 0 = appuyé et 1 = relâché Bit 7: Bouton 32 Bit 6: Bouton 33 Bit 5: Bouton 34 Bit 4: Bouton 35 Bit 3: Bouton 36 Bit 2: Bouton 37 Bit 1: Bouton 38 Bit 0: Bouton 39
1	Touches classiques, 0 = appuyé et 1 = relâché Bit 7: Bouton 29 Bit 6: Bouton 22 Bit 5: Bouton 23 Bit 4: Bouton 24 Bit 3: Bouton 25 Bit 2: Bouton 26 Bit 1: Bouton 30 Bit 0: Bouton 31
1	Touches classiques, 0 = appuyé et 1 = relâché Bit 7: Bouton 16 Bit 6: Bouton 17 Bit 5: Bouton 18 Bit 4: Bouton 19 Bit 3: Bouton 20 Bit 2: Bouton 21 Bit 1: Bouton 27

	Bit 0: Bouton 28
1	Touches classiques, 0 = appuyé et 1 = relâché Bit 7: Bouton 8 Bit 6: Bouton 9 Bit 5: Bouton 10 Bit 4: Bouton 11 Bit 3: Bouton 12 Bit 2: Bouton 13 Bit 1: Bouton 14 Bit 0: Bouton 15
1	Touches classiques, 0 = appuyé et 1 = relâché Bit 7: Bouton 0 Bit 6: Bouton 1 Bit 5: Bouton 2 Bit 4: Bouton 3 Bit 3: Bouton 4 Bit 2: Bouton 5 Bit 1: Bouton 6 Bit 0: Bouton 7
3	Inutilisé
10	Faders de 0 à 9, valeurs = 0 à 255
3	Inutilisé

Capture d'une trame Wing de type WODD



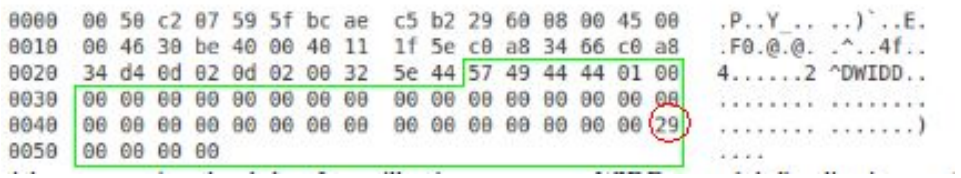
La trame Wing est entourée en vert (ce qui se situe avant est l'empaquetage des différentes couches du modèle réseau).

Sur cette capture, le fader numéro 1 à été bougé jusqu'à la valeur 47, cela correspond à l'octet entouré en rouge, les 6 octets entourés en bleu sont les octets correspondants aux boutons classiques, ici, ils sont tous en état relâchés donc les bits correspondants à 1, et donc les octets à ff. La partie orange est quant à elle l'octet gérant les boutons de contrôle (Page Up, Down, Back , Go).

Il est aussi possible d'envoyer une trame WING vers la console, le message est alors une entrée de données et le type de message WIDD. La trame peut être une entrée de données classique ou une entrée de donnée pour le programme Wing, nous utilisons uniquement l'entrée classique. On peut donc afficher un nombre (compris entre 0 et 99) sur l'écran LED de la console, pour ce faire, il faut indiquer le code en hexadécimal du nombre à afficher. Cela est ici utilisé pour afficher le nombre des dizaines du canal actif. La trame suit la structure suivante :

Taille en octets	Description
4	Type de message 'WIDD'
1	Numéro de version = 1
32	Inutilisé
1	Valeur de l'écran LED sept segment en hexadécimal
4	Inutilisé

Capture d'une trame Wing de type WIDD



Ici, le message n'est plus de la même taille, c'est un message WIDD envoyé de l'application vers la console, indiquant de changer la valeur de l'écran LED à 41 (0x29 = 41). Les autres octets de la trame de données Wing sont tous à zéro à part les 5 premiers qui permettent d'identifier la trame. Ceci est dû à un choix lors de la programmation de la fonction permettant d'envoyer les données à la console car il ne s'agit que de remplissage, cette option à été choisi car le fabricant (Enttec) utilise des 0 pour bourrer dans les trames Wing.

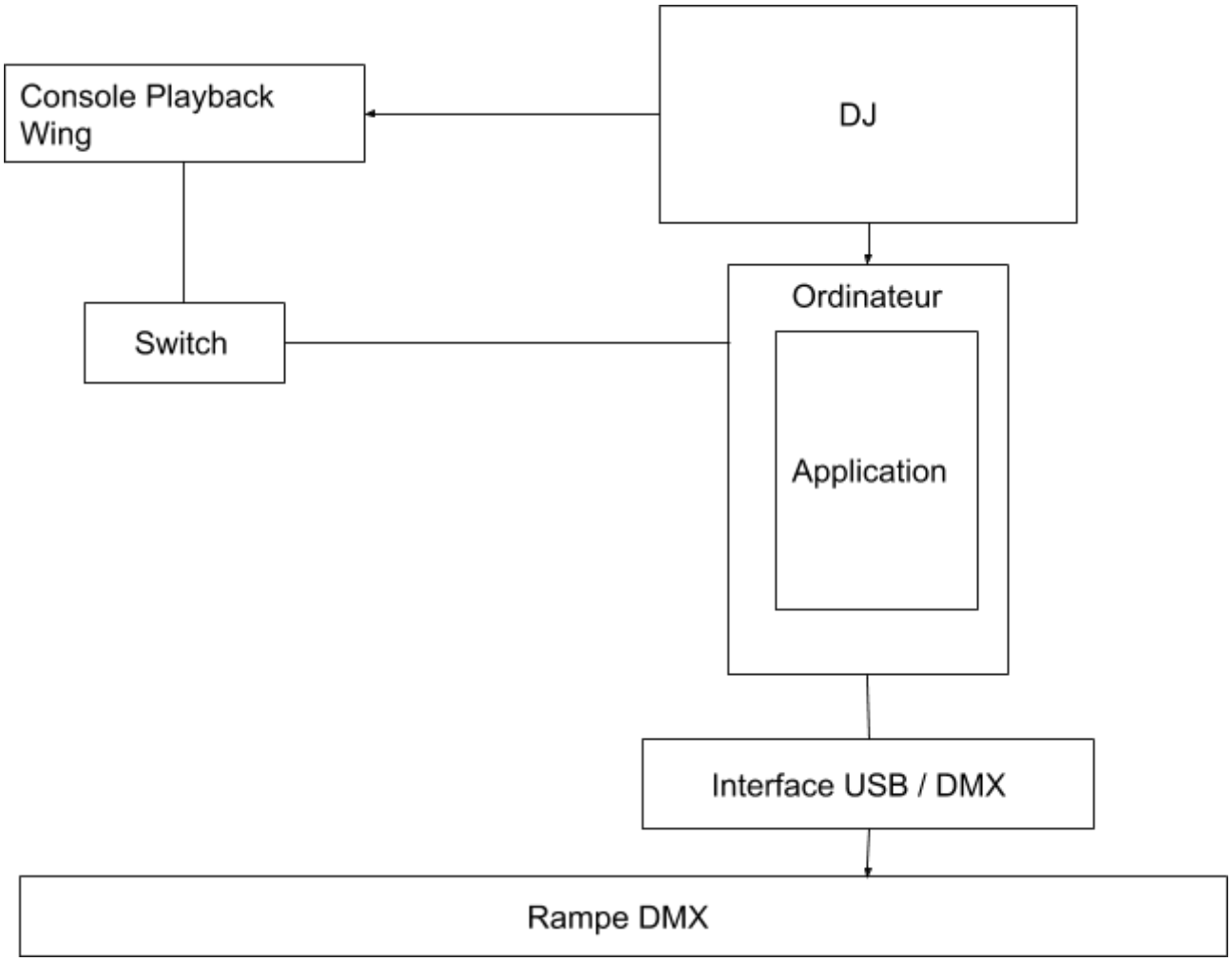
Présentation du projet

Matériels

Lors de la création de ce projet, je dispose de différents matériels :

- Une console professionnelle réseau (Ethernet UDP/IP) de type Enttec Playback Wing permettant de piloter les appareils DMX à distance.
- Différents appareils d'éclairage (scanner, lyre, projecteurs LED...)
- Une interface de communication USB/DMX (modèle “DMX USB PRO” et “OPEN DMX” de la société Enttec).
- Un réseau cablé Ethernet 100BASET.
- Un switch Ethernet.
- Un ordinateur.

Synoptique du projet



Pour ce projet, je dispose de 6 projecteurs, tous connectés à une chaîne DMX en bus :

- 2 x Lyre
- 1 x Laser
- 2 x Par LED
- 1 x Scanner

Pour communiquer avec les projecteurs, je dispose de la console réseau précédemment citée.

Le PC est la machine où l'application est exécutée, c'est elle qui contrôle l'ensemble.

Enfin, l'interface USB/DMX permet de faire la liaison entre le PC (et donc l'application) et la chaîne DMX.

Objectifs

Ce projet sera conçu par 3 étudiants, 1 option EC et 2 option IR.

Répartition des tâches

Chaque étudiant possèdes des objectifs différents qui, assemblés forment le projet DMX :

Étudiant EC 1 :

- Choisir une solution pour créer un projecteurs avec des fonctionnalités particulières.
- Compatibilité avec la norme DMX 512.

Étudiant IR 1 :

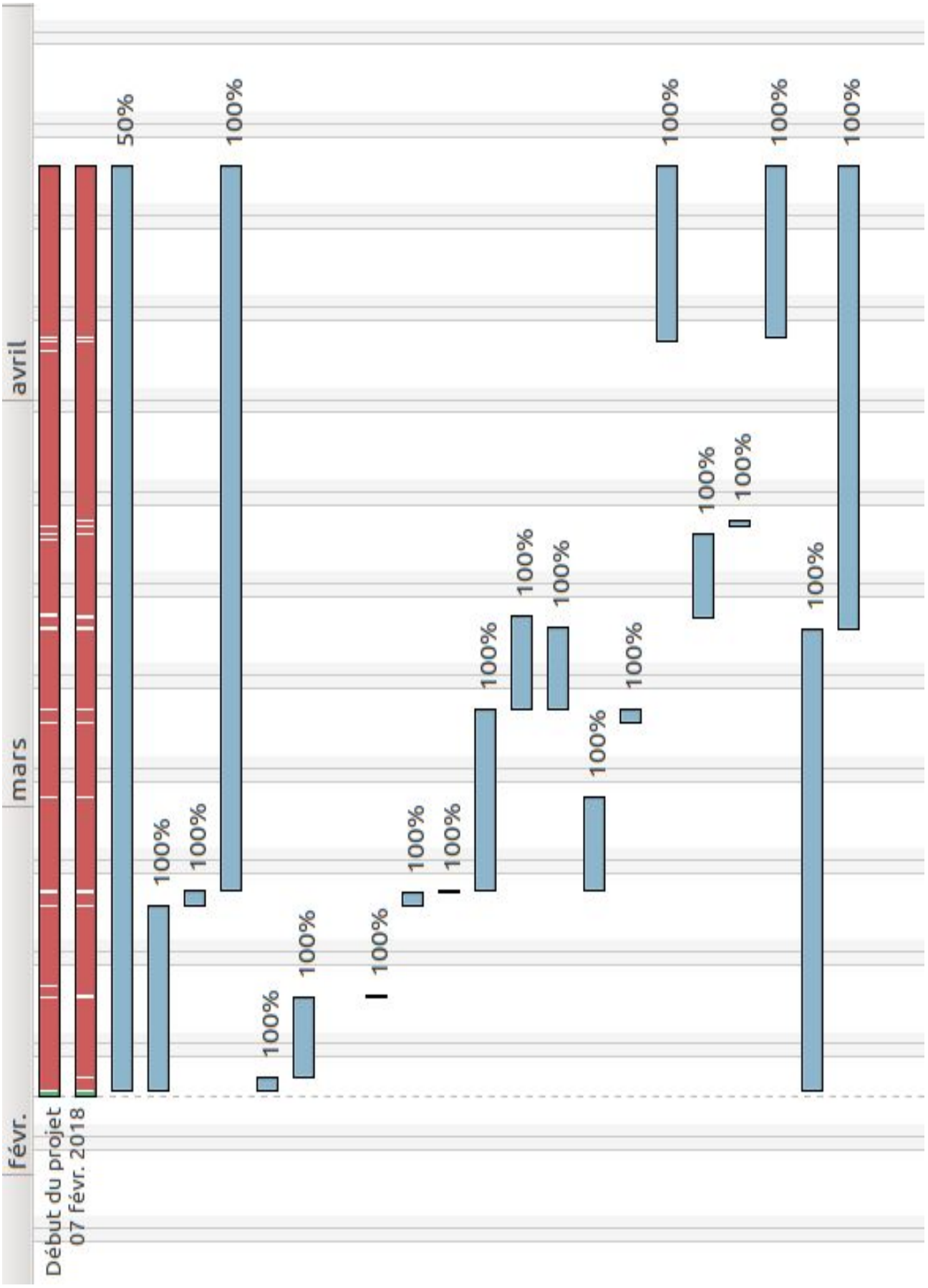
- Créer un spectacle
- Définir les plans d'éclairage
- Gérer les paramètres des projecteurs
- Jouer un spectacle

Étudiant IR 2 :

- Configurer le système. Gérer les paramètres de l'application
- Paramétrer / Gérer les paramètres de l'interface USB/DMX
- Piloter les projecteurs. Pouvoir piloter les projecteurs depuis l'application créée
- Contrôler une console distante. Pouvoir piloter les projecteurs depuis la console Wing

Planification

TPÉ	Nom	Travail
1	▾ **Projet DMX (jusqu'à la revue 3)**	**86j 2h**
1.1	▾ **Analyse**	**3j 6h**
1.1.1	établir tests de validation	1j
1.1.2	établir cas d'utilisation (UML)	1j
1.1.3	établir les objectifs	3h
1.1.4	répartir les tâches (Gantt)	3h 30min
1.1.5	prototyper l'IHM	1j
1.2	▾ **Conception**	**2j 4h**
1.2.1	créer un diagramme de classe	1j
1.2.2	créer un diagramme de séquences	1j
1.2.3	prendre en main le materiel	4h
1.3	▸ **Développement Partie IR (élève 3)**	**18j 7h**
1.4	▾ **Développement Partie IR (élève 4)**	**24j 2h**
1.4.1	▾ **Programmation du logiciel (0.8)**	**3j 1h**
1.4.1.1	établir la connexion entre console wing et pc	2j 1h
1.4.1.2	mettre en oeuvre DMX	1j
1.4.2	tests (0.8)	3j 2h
1.4.3	▾ **Programmation du logiciel (0.9)**	**12j 5h**
1.4.3.1	mettre en oeuvre XML	1j 6h
1.4.3.2	renseigner les fichiers XML (consoles.xml, adaptateurs.xml)	7h
1.4.3.3	implémenter les options de choix d'interface DMX USB depuis le logiciel	10j
1.4.4	tests (0.9)	5j 2h
1.5	▾ **Rédaction de la documentation**	**36j 6h**
1.5.1	rédiger le compte rendu de la revue 2	18j 1h
1.5.2	rédiger le compte rendu de la revue 3	18j 4h


Févr.
mars
avril
Début du projet
07 févr. 2018
50%
100%
100%
100%
100%
100%
100%
100%
100%
100%
100%
100%
100%
100%
100%
100%
100%
100%
100%
100%
100%

Le projet est séparé en plusieurs parties :

- **Analyse**, c'est à dire l'analyse des besoins du client et la recherche des solutions disponibles. Cette étape permet de comprendre les besoins du client afin de réaliser la conception qui est la prochaine étape.
- **Conception**, la prise en main du projet avec la réalisation des diagrammes et scénario / objectifs possibles pour répondre aux besoins établis à l'analyse, ainsi que la prise en main du matériel.
- **Implémentation**, le développement de l'application et des fichiers nécessaires au fonctionnement de cette dernière. Ceci correspond au code source, c'est donc le cœur du projet.
- **Test**, cette partie permet de vérifier que les besoins établis sont bien transcrits dans l'application et qu'il n'y a pas d'erreurs dans l'implémentation.

Architecture du système

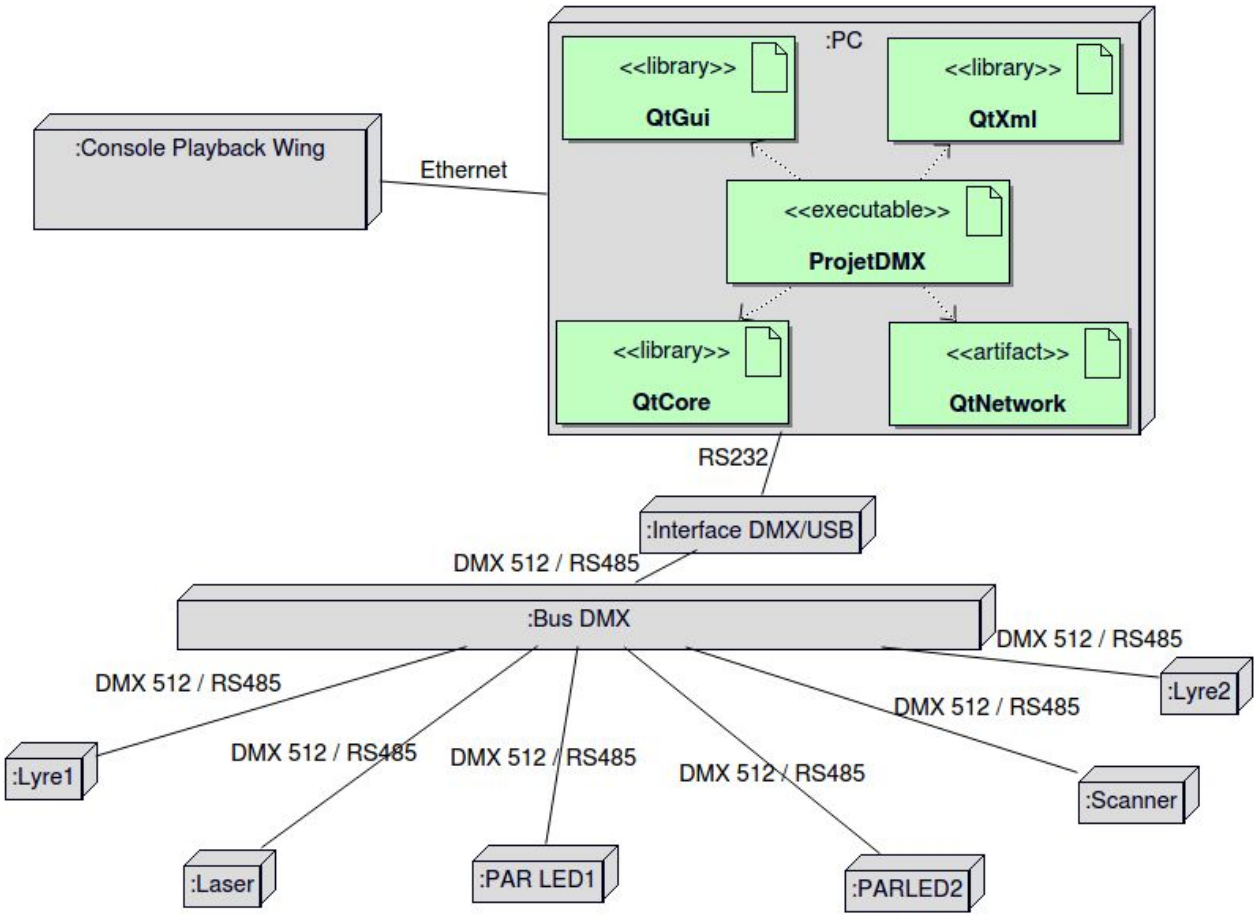


Diagramme de déploiement

On voit ici les différents équipements du projet :

- La console Playback Wing est connectée au réseau par Ethernet en UDP, elle communique en broadcast, ainsi, tous les équipements connectés au réseau reçoivent les trames Wing.
- Le PC envoi les données à l'interface par USB. L'interface permet ainsi de se connecter aux projecteurs depuis le PC.
- Les projecteurs sont tous situés sur une rampe DMX en bus, les données sont donc envoyées à tous les projecteurs du bus en même temps.
- L'interface est aussi reliée à la rampe avec la norme DMX 512 basée sur la norme RS485.

Les ressources de développement

Pour la création de l'application, nous disposons de ressources logicielles :

OS	GNU Linux (Ubuntu 12.04.5 LTS)
EDI	Qt Creator 2.4.1
Compilateur	GNU g++/gcc version 4.6.3
Débuguer	GNU gdb 7.4
Fabrication	Qmake 2.01a et GNU make 3.81
API GUI	Qt 4.8.1
UML	bouml 7.4
Version	subversion (client svn 1.6.17)
Documentation	Doxygen version 1.7.6.1
Gantt	Planner (version 0.14.5)

Analyse

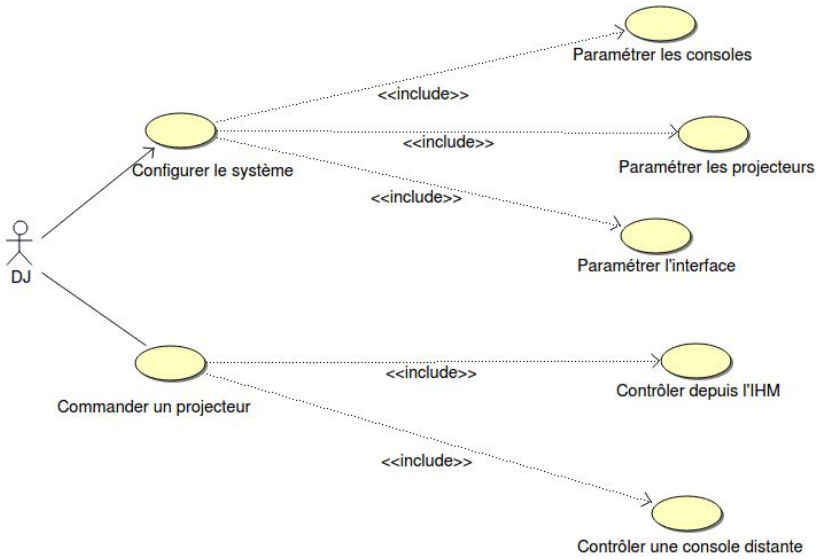


Diagramme de cas d'utilisation du projet

Ce diagramme permet de voir les différentes possibilités proposées par l'application du point de vue de l'acteur, ici, le DJ.

Ainsi, on voit que le DJ peut configurer le système en paramétrant les projecteurs il peut, depuis l'application, ajouter une console via son adresse IP et choisir parmi les consoles déjà disponibles. Il est aussi possible de paramétrer l'interface USB/DMX, comme pour les consoles, il est possible d'ajouter une interface en indiquant sont port et son type, de même qu'il est possible de choisir parmi les différentes interfaces déjà disponibles. Enfin, il peut paramétrer les projecteurs en modifiant leurs différentes informations et en ajouter / supprimer.

L'utilisateur peut commander un projecteur depuis l'Interface de l'application ou depuis la console distante Wing. Depuis l'interface, une page est disponible pour contrôler les projecteurs, avec des sliders (un bouton coulissant prenant différentes valeurs, ici, 0 à 255) correspondant à ceux de la console d'où il est aussi possible de piloter les projecteurs.

Présentation détaillée

Les fonctionnalités

Les fonctionnalités dont je suis responsable sont les suivantes :

- Cas d'utilisation
    - Configurer le système
    - Piloter les projecteurs
    - Contrôler une console distante

Les tâches dont je suis responsable sont les suivantes :

- Installation
    - L'interface de communication DMX PRO
    - La console Wing
- Mise en œuvre
    - Les fichiers XML
    - L'environnement de développement
- Configuration
    - L'interface de communication DMX PRO ou OPEN DMX
    - La console Wing et le réseau
- Réalisation
    - Les diagrammes UML
    - L'IHM et les classes du module
- Documentation
    - Le dossier technique et les documentation des équipements utilisés (Interface DMX/USB PRO et console Playback Wing).

Cahier des charges

Il faudrait donc que l'application possède les fonctions énumérées précédemment :

- **Configurer le système**, cela signifie en fait gérer les paramètres des consoles et paramétrer l'interface USB DMX. Cela nécessite une détection et connexion à l'interface afin de communiquer avec elle. Les consoles ne sont elles pas détectées automatiquement par l'application, toutes les consoles enregistrées dans l'application peuvent émettrent, cependant cela nécessite de les enregistrer auparavant ce qui est possible en indiquant leurs adresses IP et leurs ports de communication.

- **Configurer un projecteur**, le pilotage est disponible depuis l'interface de l'application ou depuis une console distante (enregistrée). Cependant, il n'est pas possible de piloter les projecteurs depuis l'interface et la console en même temps car cela pourrait perturber l'utilisateur, il ne faut que quelqu'un ait accès à l'un ou l'autre tandis que le propriétaire est occupé sur l'un.

- **Contrôler une console distante**, le contrôle de la console est disponible mais pas manuellement. En effet, la seule action que l'on peut faire sur la console depuis le PC est changer l'affichage de l'écran LED, cela est donc attribué automatiquement par l'application, il affiche le chiffre des dizaines du canal DMX actuellement actif. Comme dit précédemment, la console peut elle communiquer avec le PC pour indiquer les changements d'état de ses touches. Cependant, cette fonctionnalité n'est pas activé au démarrage de l'application, ainsi, l'utilisateur peut choisir depuis la console ou l'interface celui des deux qu'il veut utiliser pour commencer.

Architecture logicielle

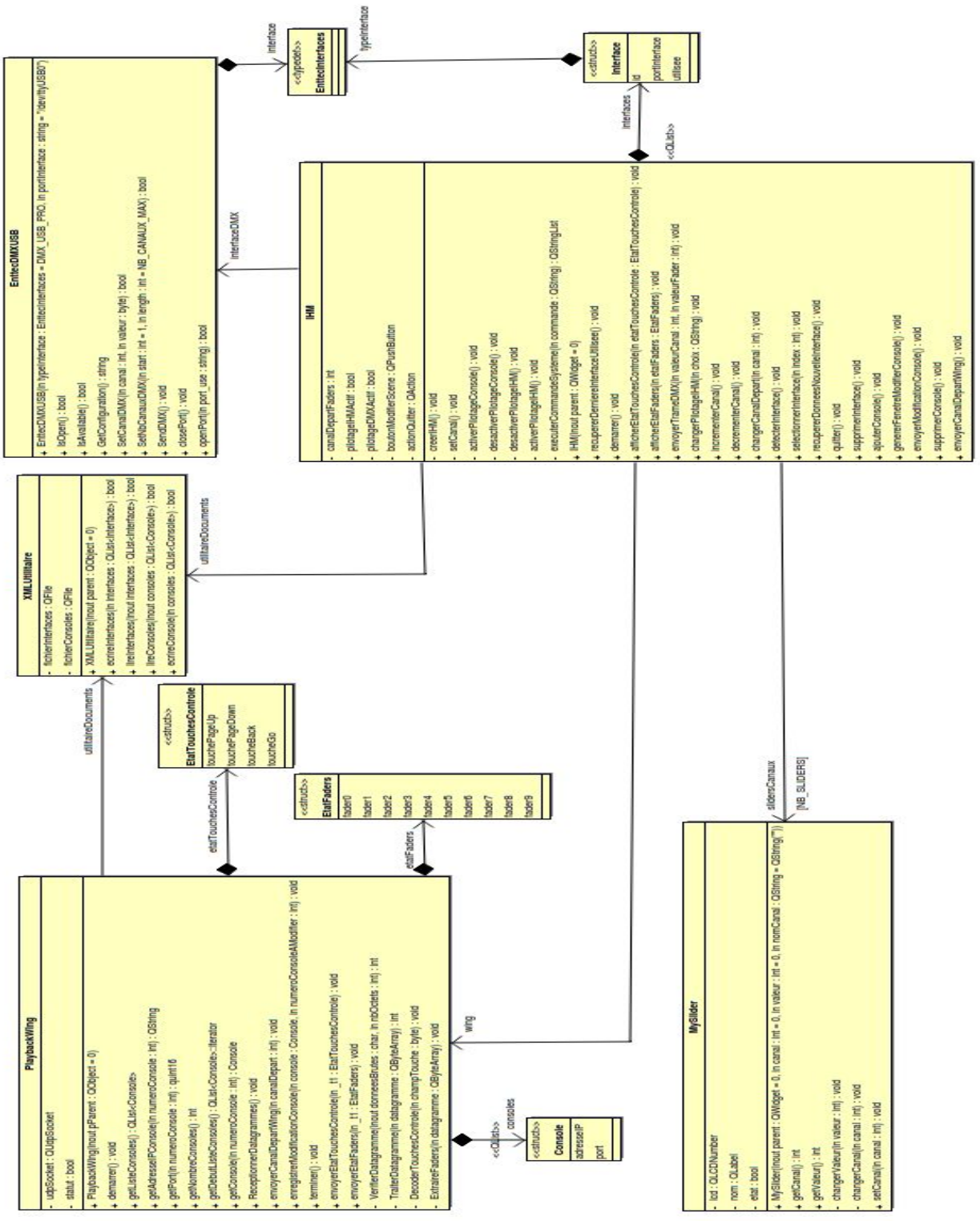
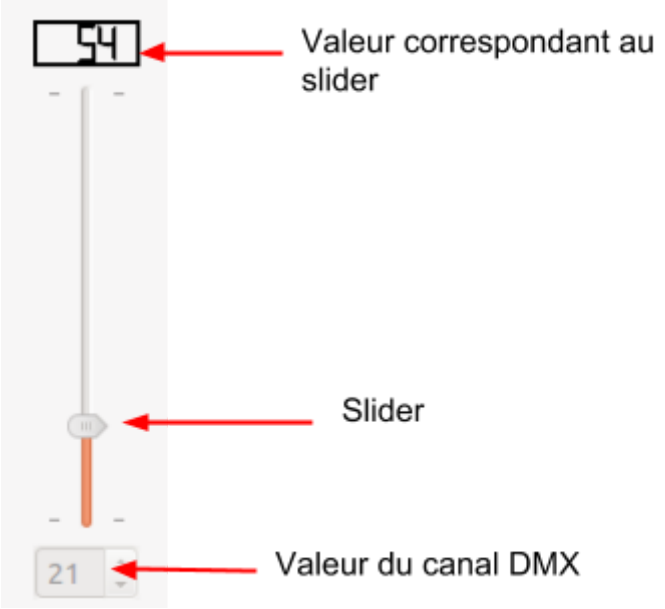


Diagramme de classes (simplifié)

Avec ce diagramme de classes, on voit les principales classes du projet. D'autres classes sont nécessaires mais ne concernent pas ma partie du projet et donc ne sont pas affichées ci dessus.

La classe **PlaybackWing** est la classe gérant la réception et le traitement de données en provenance de la console Wing, elle envoie aussi les états des touches de la commande Wing (via les structures « EtatFaders » et « EtatTouchesControles » qui informe de l'état de ces touches) vers des structures correspondants au différents types de boutons (Slider et boutons de commande). De plus, c'est elle qui gère l'envoi de données vers la console, qui sert à envoyer le chiffre des dizaines du canal DMX actif (donc le canal de départ du projecteur utilisé) vers la console afin qu'elle l'affiche sur son écran LED. Elle est aussi composée de la structure « consoles » permettant de connaître l'adresse IP et le port de la console.

La classe **IHM** est la classe gérant l'interface de l'application. La plupart de ses attributs sont donc des widgets (non affiché sur le diagramme de classe). La classe peut afficher l'état des touches sur l'interface et envoyer ces données vers l'interface USB/DMX ce qui permet de relayer ces données vers les projecteurs. Cette classe est le cœur de l'application, ce qui explique ses nombreuses relations vers les autres classes. La structure interfaces permet de gérer les paramètres des interfaces USB/DMX (son id définit par l'ordinateur, le port auquel elle est rattachée et si elle est utilisée).



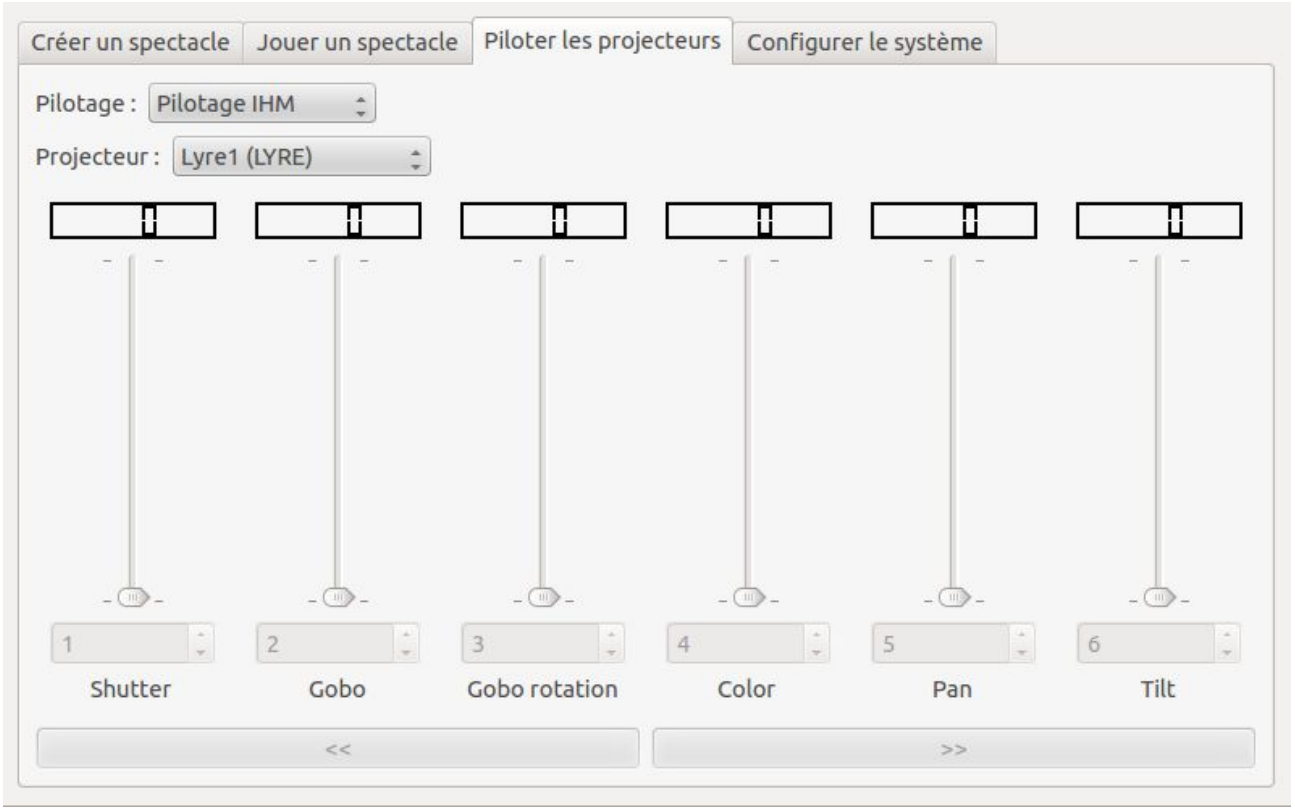
La classe MySlider permet d'instancier des widgets de type slider personnalisés, il permettent d'afficher leur valeur ainsi que le numéro du canal avec un affichage LCD (voir ci contre).

La classe **EnttecDMXUSB**, permet de faire l'intermédiaire entre les données envoyées par l'application vers l'interface et les projecteurs c'est dire les canaux et leurs valeurs (et de(s) l'interface / projecteurs vers l'application).

Enfin, la classe **XMLUtilitaire** est la classe permettant de récupérer / enregistrer des données depuis / dans des fichiers xml, c'est à dire les informations sur les interfaces (port, type) et les consoles (adresse IP et port).

IHM

Piloter des projecteurs



Capture d'écran de l'IHM

Depuis l'IHM, il est possible de contrôler les projecteurs avec, par défaut, les 10 sliders (correspondants aux *sliders* de la commande Wing), ainsi que leurs canaux avec les champs disponible sous les sliders. Il est aussi possible de verrouiller le pilotage depuis l'IHM ou la console avec la liste en haut de la fenêtre (ce qui est aussi possible depuis la console avec la touche Back verrouillant l'IHM et la touche Go verrouillant la console).

Ci dessus, on voit le pilotage d'un projecteur en particulier, ici, la lyre 1. Quand on sélectionne un projecteur, seul les canaux qu'il possède sont disponibles pour les sliders et le nom de chaque canal est indiqué.

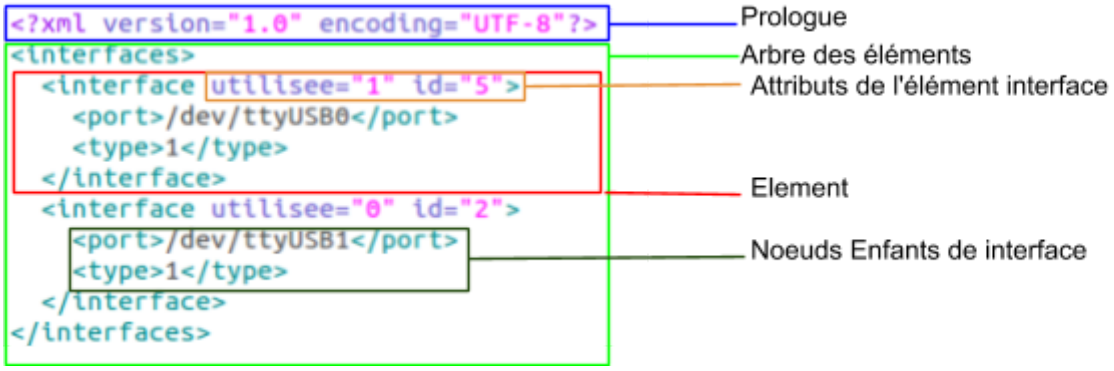
Configurer les interfaces

Depuis cette page, on peut configurer les différents paramètres des interfaces. On peut donc ajouter (en indiquant son port et son type) ou supprimer une interface (en la sélectionnant dans la liste déroulante).

Au niveau technique, cela est possible grâce à l'utilisation du langage XML. Ce langage de description de document nous permet d'enregistrer des données sous forme de texte facilement et universellement (ASCII).

Je suis en charge de 2 fichiers XML :

- le fichier **consoles.xml** (qui sera détaillé plus tard),
- le fichier **adaptateurs.xml** pour les informations concernant les types d'interfaces (1 pour l'interface PRO et 0 pour l'interface OPEN), le port de l'interface et si l'interface est utilisée (1) ou non (0).



adaptateurs.xml

Sur le fichier adaptateurs.xml ci dessus, on distingue 2 parties.

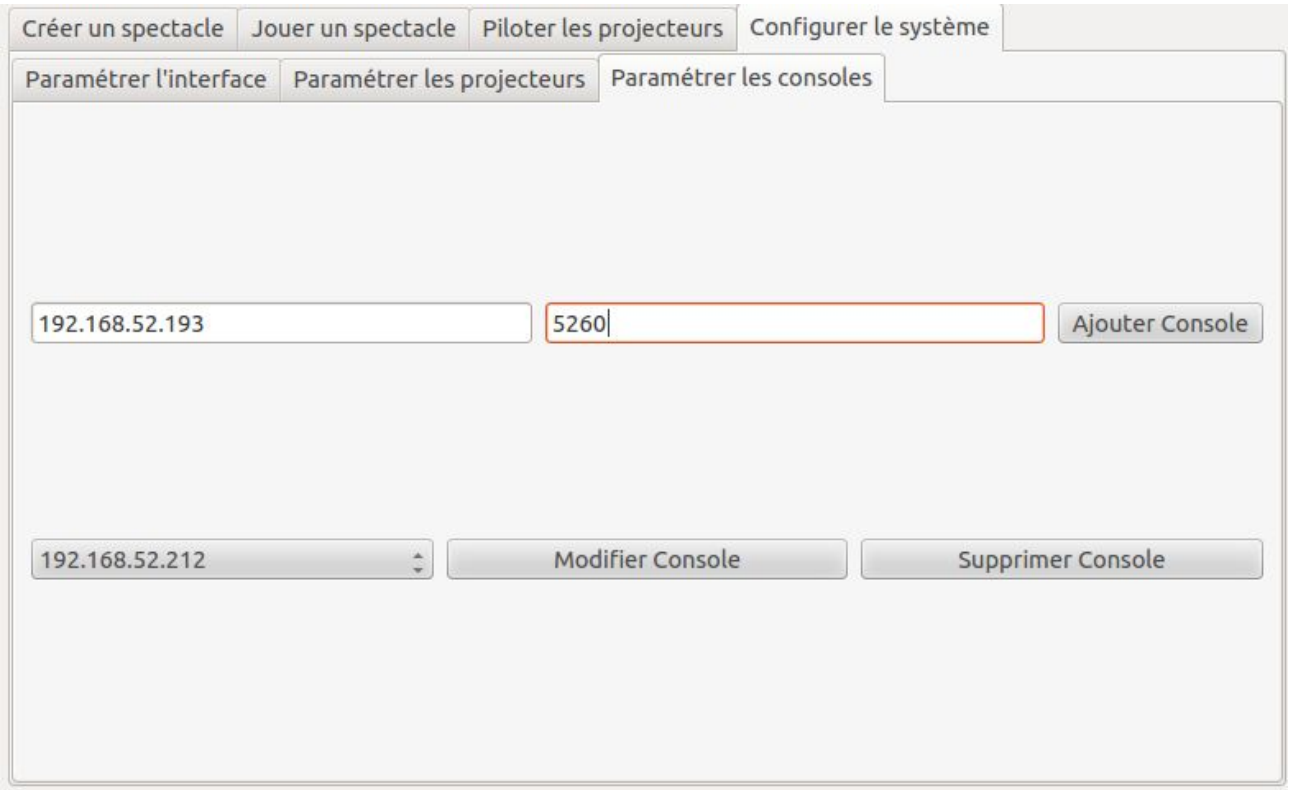
La première est le prologue, il permet de définir la version, ici 1.0 et le type d'encodage, UTF-8. Cette partie ne contient aucune donnée.

Le reste du fichier est l'**arbre des éléments (DOM)**, il pourrait il y avoir des instructions de traitement avant, mais cela n'est pas utile dans notre cas. Le fichier est construit sur un élément racine nommé 'interfaces', il est le premier élément et est obligatoire. Le premier noeud fils de la racine est 'interface', il possède 2 attributs : 'utilisee' indique si l'interface est utilisée ou non (cet attribut permet de savoir quelle est la dernière interface utilisée), le second est 'id', il rend possible l'identification de l'interface par la machine exécutant l'application. Enfin, l'élément interface possède lui même 2 noeuds enfants : 'port' contient le port de l'interface et 'type' le type d'interface (1 pour PRO et 0 pour OPEN).

En accédant à ce fichier, la classe xmlutilitaire permet de récupérer les informations citées précédemment et les transcrire dans l'application. De plus, grâce à cette même classe, il est possible d'écrire dans ce fichier pour modifier ou ajouter / supprimer une interface.

Un des avantages du langage XML est qu'il est lisible humainement, ainsi, on remarque, en comparant avec le résultat du bouton 'Détecter interfaces' qu'une interface à été ajoutée depuis la capture d'écran de l'IHM, elle n'est pas utilisée, son port est '/dev/ttyUSB1' est elle est de type PRO ('utilisee="0"', '<port>/dev/ttyUSB1</port>' et '<type>1</type>'). Cela à été possible grâce à la fonction permettant l'ajout d'interface depuis l'IHM.

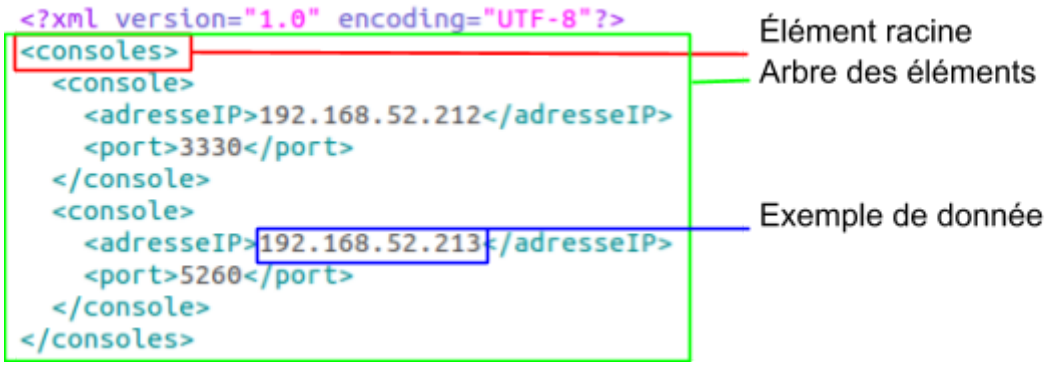
Configurer les consoles



Depuis cette page, on peut configurer les différents paramètres des consoles. On peut donc ajouter (en indiquant son adresse IP et son port), modifier ou supprimer une console (en la sélectionnant dans la liste déroulante). Dans la capture ci dessus, on ajoute donc une console à l'adresse 192.168.52.193 qui communique sur le port 5260.

Ici, c'est donc le fichier consoles.xml qui rend possible l'enregistrement des informations sur les consoles. Il contient les informations suivantes :

- adresse IP de la console pour pouvoir communiquer avec elle (notamment pour envoyer les données concernant l'affichage sur l'écran LED).
- le numéro de port sur lequel la console communique avec le PC.



consoles.xml

Sur ce fichier est présent le même prologue car même version et encodage.

Le fichier consoles.xml est construit sur l'élément racine 'consoles', les éléments racines sont nommées comme le fichier pour plus de facilité. Le premier enfant de cette racine est le noeud 'console', il contient lui même 2 noeuds enfants, 'adresseIP' qui contient l'adresse IP de la console sélectionnée et 'port' son port de communication. On notera qu'il n'y a ici pas d'attribut 'id' contrairement au fichier adaptateurs.xml car les consoles sont identifiées par leurs adresse IP.

De même que pour adaptateurs.xml et la fenêtre 'configurer interface', les actions faites sur l'IHM dans la fenêtre 'configurer consoles' modifient ou lisent les données du fichier consoles.xml.

Ici, on voit que la console indiquée dans la capture d'écran de l'IHM (à la ligne correspondant à l'ajout de console) a été ajoutée. Ces données ont donc été écrites dans le fichier et sont désormais accessibles, par exemple son adresse IP encadrée en bleu dans la capture ci dessus.

Scénarios

Paramétrer Consoles

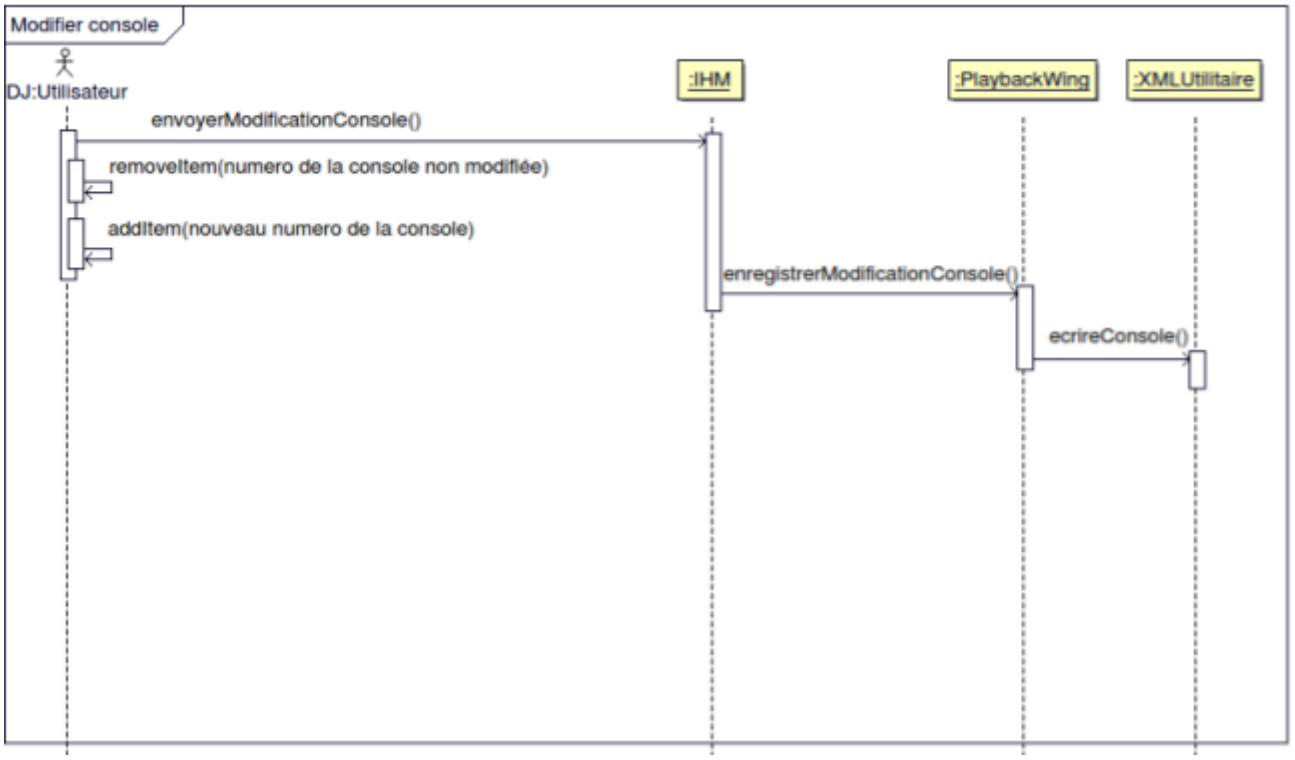


diagramme de séquence

Sur ce diagramme, c'est la fonction permettant la modification d'une console qui est illustrée, cependant, la suppression et l'ajout fonctionne exactement de la même manière, seul les noms des méthodes changent (et leurs paramètres).

Lorsque l'utilisateur modifie par exemple l'adresse IP d'une console depuis l'IHM car il a changé de réseau, la méthode envoyerModificationConsole() : void est appelée, elle récupère les données entrées par l'utilisateur qui ont été stockées dans une structure de type 'Console' (composée de 2 attributs : adresse IP et port) et envoie ces données vers un objet 'wing' de la Classe PlaybackWing par le biais de la fonction enregistrerModificationConsole(Console, int) : void qui va elle même modifier une liste (QList) de structure 'Console' pour mettre à jour les données modifiées et envoyer cette

liste à l'objet 'utilitaireDocuments' de type XMLUtilitaire avec la fonction ecrireConsole(QList) qui modifie le fichier consoles.xml et le réécrit afin qu'il corresponde aux modifications apportées par l'utilisateur.

Méthode envoyerModificationConsole()

```
void IHM::envoyerModificationConsole()
{
   Console console;
   if(champAdresseIPModifierConsole->text() == "")
   {
       console.adresseIP =
wing->getAdresseIPConsole(listeChoixConsole->currentIndex());
       //qDebug() << "adresse IP console = " <<
console.adresseIP;
   }
   else console.adresseIP =
champAdresseIPModifierConsole->text();
   if(champPortModifierConsole->text() == "")
   {
       console.port =
wing->getPort(listeChoixConsole->currentIndex());
       //qDebug() << "port console = " << console.port;
   }
   else console.port =
champPortModifierConsole->text().toInt();

   wing->enregistrerModificationConsole(console,
listeChoixConsole->currentIndex());

   fenetreModifierConsole->close();

   listeChoixConsole->clear();
   for(int i = 0; i < wing->getNombreConsoles(); i++)
   {

listeChoixConsole->addItem(wing->getAdresseIPConsole(i));
   }
}
```

Objet de type Console (structure), permet de stocker les données afin de les envoyer en une fois après coup

Affectation des valeurs entrées par l'utilisateur dans les différents champs (QLineEdit) aux attributs de 'console'

Appel de la méthode enregistrerModificationConsole() de l'objet wing (classe PlaybackWing), la console en paramètre envoie les données et le deuxième paramètre permet d'identifier quelle console à été modifiée

Suppression de l'item correspondant à l'ancienne console et ajout de la nouvelle dans la liste (QComboBox) listeChoixConsole qui permet de choisir la console à modifier

Méthode enregistrerModificationConsole()

```
void PlaybackWing::enregistrerModificationConsole(Console
console, int numeroConsoleAModifier)
{
   qDebug() << "nombre de consoles" << consoles.count();
   consoles.removeAt(numeroConsoleAModifier);
   consoles.append(console);
   utilitaireDocuments->ecrireConsole(consoles);
}
```

Supprime la console à modifier de 'consoles' (QList) en l'identifiant avec le numéro passé en paramètre et ajoute la console modifiée a consoles. Puis, passe la liste à la fonction ecrirConsole() de l'objet utilitaireDocument (XMLUtilitaire).

Méthode ecrireConsole() [partielle]

```
bool XMLUtilitaire::ecrireConsole(const QList<Console> &consoles)
{
    if (!fichierConsoles.open(QIODevice::ReadWrite))
    {
        qDebug() << Q_FUNC_INFO << "Erreur ouverture !" <<
fichierConsoles.fileName();
    }
    if(!(fichierConsoles.isOpen()))
    {
        QMessageBox::critical(0, QString::fromUtf8("Erreur"),
QString::fromUtf8("Le fichier %1 n'a pas pu être ouvert
!").arg(fichierConsoles.fileName()));
        return false;
    }
```

Vérifie si l'ouverture du fichier se passe bien

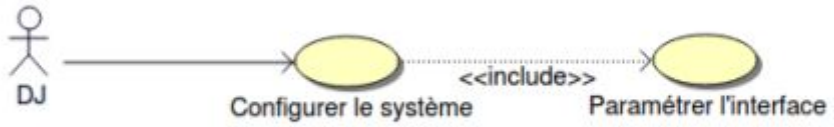
```
        QDomDocument documentXML;

        fichierConsoles.resize(0);
        //documentXML.setContent(&fichierInterfaces, false);
        QDomNode xmlNode = documentXML.createProcessingInstruction("xml","version=\"1.0\"
encoding=\"UTF-8\"");
        documentXML.insertBefore(xmlNode, documentXML.firstChild());

        if(consoles.size() > 0)
        {
            QDomElement root = documentXML.createElement("consoles");
            documentXML.appendChild(root);

            for(int i = 0; i < consoles.size(); i++)
            {
                QDomElement elementConsole = documentXML.createElement("console");
                root.appendChild(elementConsole);

                QDomElement elementAdresseIP = documentXML.createElement("adresseIP");
                elementConsole.appendChild(elementAdresseIP);
                QDomText text = documentXML.createTextNode(consoles[i].adresseIP);
                elementAdresseIP.appendChild(text);

                QDomElement elementPort = documentXML.createElement("port");
                elementConsole.appendChild(elementPort);
                text = documentXML.createTextNode(QString::number(consoles[i].port));
                elementPort.appendChild(text);
```

Prologue

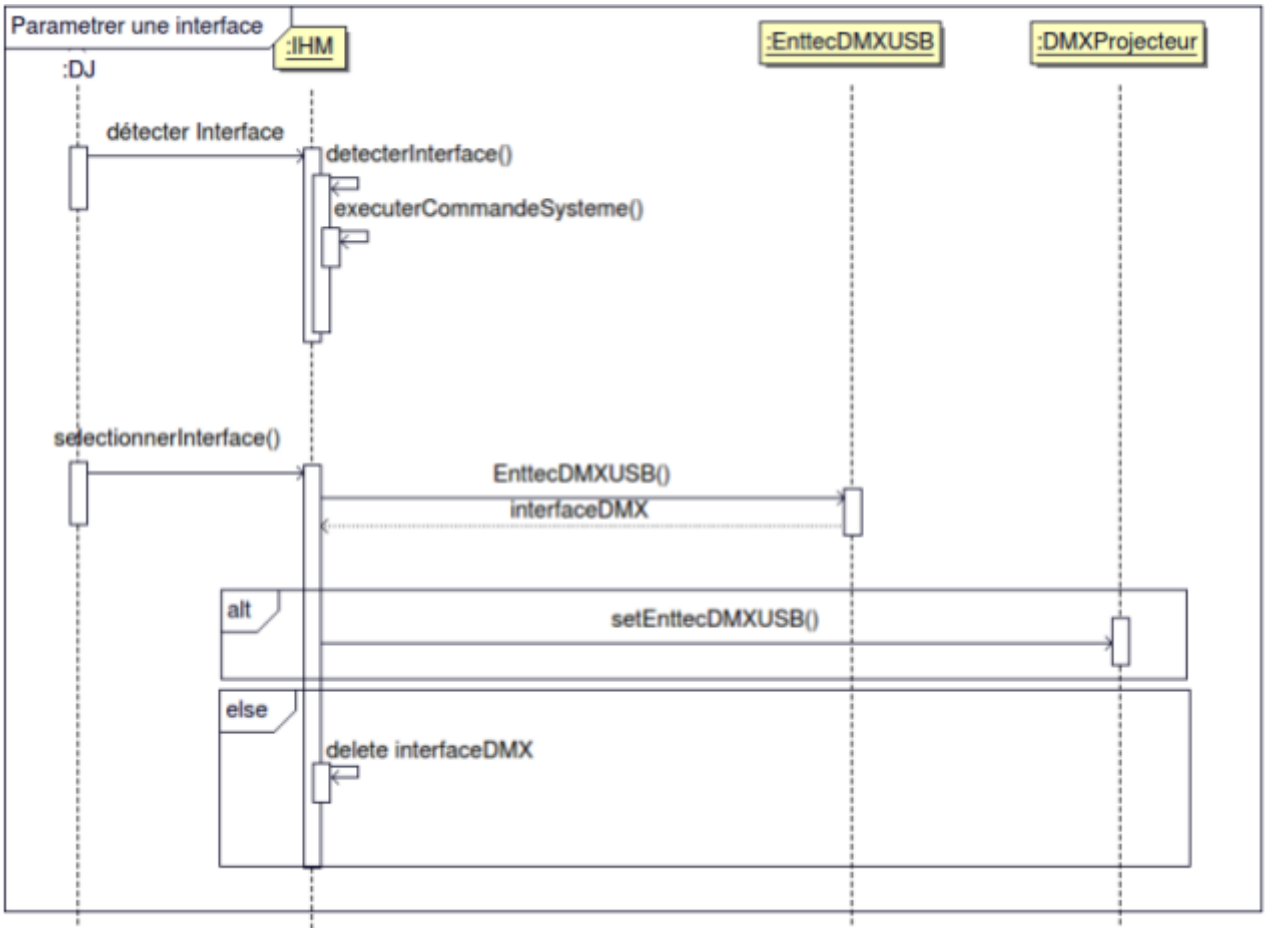
Création et placement de la racine

Création de l'element adresse IP et affectation de sa valeur

Paramétrer Interfaces



Sur la page de paramétrage des interfaces, l'utilisateur peut ajouter / supprimer des interfaces, choisir l'interface active ou détecter les interfaces disponibles, je montre ici le scénario où on détecte les interfaces disponibles, puis choisit parmis elles celle que l'on veut. L'ajout / suppression fonctionne de la même manière que pour les consoles.



Ici, l'utilisateur commence par détecter les Interfaces connectées à sa machine en cliquant sur le bouton de l'IHM. Cela enclenche la fonction 'detecterInterface()', cette dernière appelle la méthode 'executerCommandeSysteme(QString)' qui prend en paramètre une commande système puis l'exécute et 'detecterInterface()' affiche dans une zone de texte (QTextEdit) le résultat de la commande.

La seconde action de l'utilisateur est de choisir une nouvelle interface dans la liste (QComboBox). Cela enclenche la méthode 'selectionnerInterface()' de la classe IHM, celle ci va créer une nouvelle interface DMX, puis tester si cette interface peut être utilisée, si oui, elle est initialisée comme interface utilisée, sinon, un message d'erreur est envoyé à l'écran et la fonction s'arrête.

```
   resultatDetectionInterface->append(executerCommandeSysteme("lsusb |
grep -i serial").join("\n"));
   resultatDetectionInterface->append("Interfaces disponibles : ");
   resultatDetectionInterface->append(executerCommandeSysteme("ls
/dev/ttyUSB*").join("\n"));
```

Cet extrait de la méthode 'detecterInterface()' montre l'appel de la méthode 'executerCommandeSyteme()' et son écriture dans la zone de texte dédiée.

Méthode selectionnerInterface() [partielle]

```
interfaceDMX = new
EnttecDMXUSB(interfaces.at(index).typeInterface,
interfaces.at(index).portInterface.toLocal8Bit().data());
   interfaces[index].utilisee = 1;

   if(interfaceDMX->IsOpen())
   {
       for(int i = 0; i < projecteursDMX.count(); i++)
       {

            projecteursDMX.at(i)->setEnttecDMXUSB(interf
            aceDMX);
       }
   }
   else
   {
       QMessageBox::critical(0, QString::fromUtf8("DMX
2018"), QString::fromUtf8("Impossible d'ouvrir l'interface
DMX !"));
       delete interfaceDMX;
       interfaceDMX = NULL;
   }
```

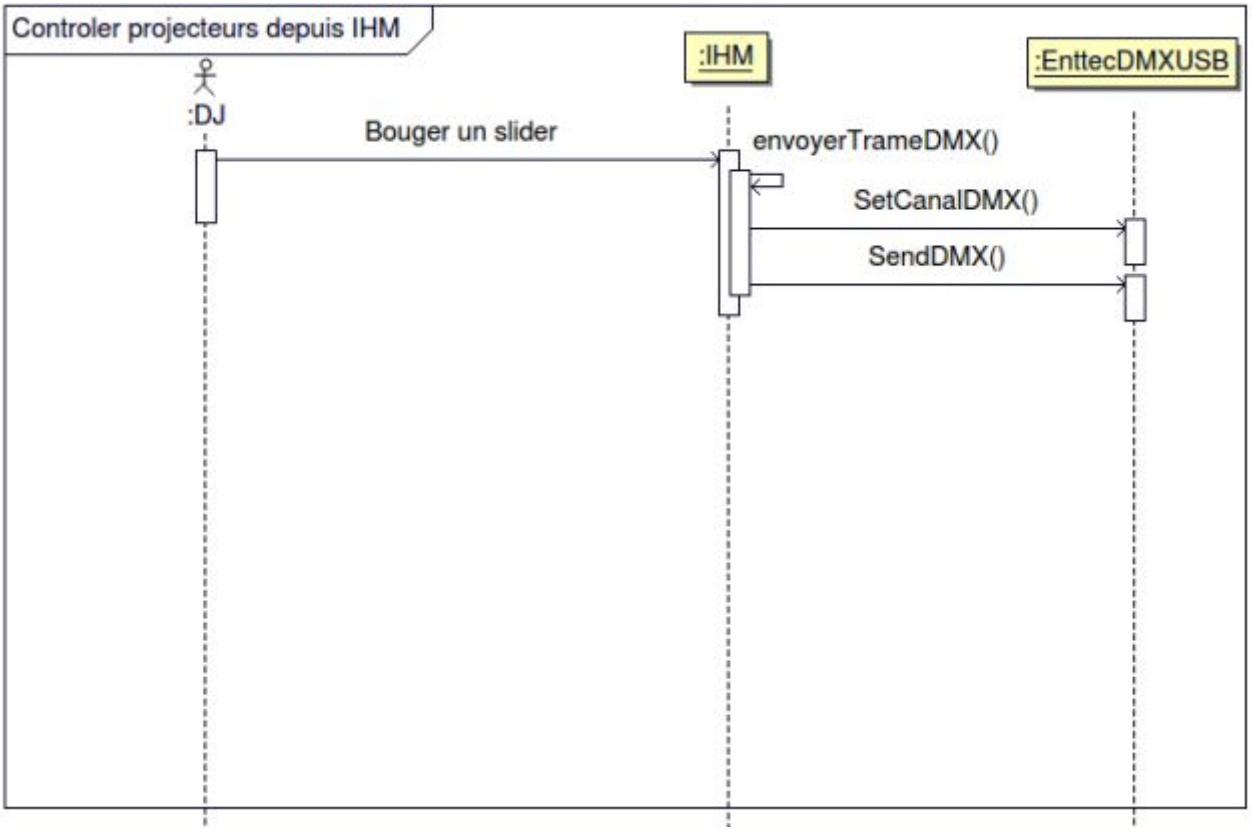
Instanciation de la nouvelle interface à partir des informations stockées dans le fichier adaptateurs.xml

Test de l'ouverture de l'interface et attribution en tant qu'interface utilisée avec la méthode setEnttecDMXUSB()

Si le test est négatif, l'interface est supprimée et le message d'erreur apparaît

Commander projecteurs depuis IHM

Le scénario suivant est celui ou l'animateur va utiliser un slider personnalisé de l'IHM depuis la page Piloter les projecteurs afin de modifier l'état d'un projecteur.



Grâce à un connect, lorsque le DJ bouge un slider sur l'IHM, le signal 'sliderChange(int, int)' est émis par le slider personnalisé qui est connecté au slot 'envoyerTrameDMX(int, int)' de l'IHM. Ils se passent le canal et sa valeur en paramètres. Le slot envoi ensuite ces 2 valeurs à l'interface par le biais des fonction SetCanalDMX(int, byte) qui les attribue à un de ses attributs. La méthode 'SendDMX()' envoie quand à elle les valeurs vers l'interface et les projecteurs (en récupérant l'attribut précédemment cité).

Méthode ‘envoyerTrameDMX()’

```
void IHM::envoyerTrameDMX(int valeurCanal,int valeurFader)
{
   if(pilotageDMXActif)
   {
       if(interfaceDMX != NULL)
       {
           interfaceDMX->SetCanalDMX(valeurCanal, valeurFader);
           interfaceDMX->SendDMX();
       }
   }
}
```

Cette méthode permet de vérifier qu’une interface est active avant d’envoyer les données.

Les méthodes ‘SetCanalDMX()’ et ‘SendDMX()’ ont été fournis au début du projet.

Commander projecteurs depuis consoles

Dans cette situation, le DJ va commander un projecteurs depuis la console distante, en passant par l’application.

La console est connectée en UDP sur le réseau et communique en broadcast, pour accéder aux trame qu’elle envoie, il faut donc se situer dans le même réseau et avoir un socket udp attaché sur le port auquel la console communique (3330).

Un socket UDP est instancié et lié dans la classe ‘PlaybackWing’ à l’adresse ‘0.0.0.0’ port 3330 afin de correspondre à toutes les console playback wing (3330 étant le port choisi par la société Enttec).

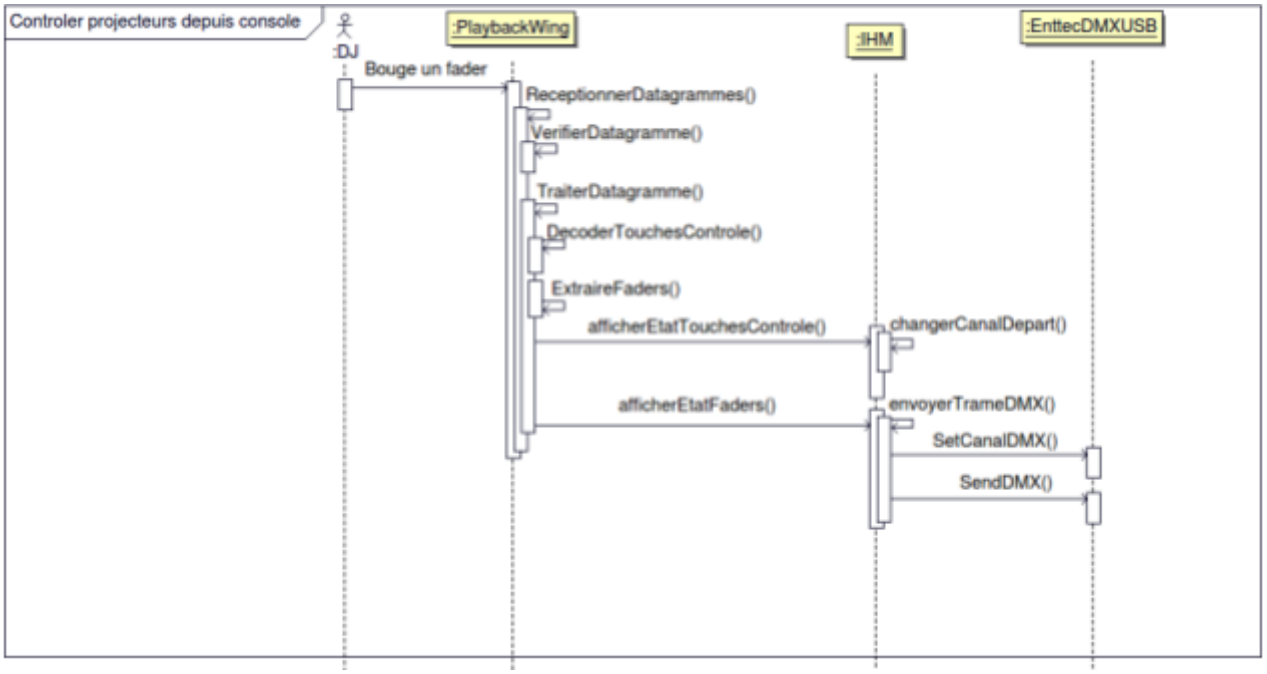
Instanciation et attachement du socket udp

```
udpSocket = new QUdpSocket(this);

utilitaireDocuments = new XMLUtilitaire;
utilitaireDocuments->lireConsoles(consoles);
// Attachement locale de la socket UDP :
statut = udpSocket->bind(QHostAddress((QString)"0.0.0.0"), PORT_ENTTEC);
if(!statut)
{
    QMessageBox::critical(NULL, QString::fromUtf8("PlaybackWing"),
QString::fromUtf8("Erreur bind sur le port %1").arg(PORT_ENTTEC));
}
```

On instancie le socket

Affichage d'un message d'erreur si l'attachement n'a pas fonctionné

On connecte ensuite le signal 'readyRead' du socket à un slot 'ReceptionnerDatagrammes()' qui, lorsque qu'appelé, écrit les données reçus par le socket dans un tableau d'octets (QByteArray). 'readyRead' est enclenché lorsqu'un datagramme arrive sur le socket.



Méthode ReceptionnerDatagrammes() [partielle]

```
while (udpSocket->hasPendingDatagrams())
  {
     QByteArray donneesDatagramme;
     QHostAddress emetteurAdresse;
     quint16 emetteurPort;


     // Fixe la taille du tableau au nombre d'octets reçus en
attente
     donneesDatagramme.resize(udpSocket->pendingDatagramSize());

     // Lit le datagramme en attente
     //nbOctets = udpSocket->readDatagram(datagram.data(),
datagram.size());
     nbOctets = udpSocket->readDatagram(donneesDatagramme.data(),
donneesDatagramme.size(), &emetteurAdresse, &emetteurPort);
```

tableau d'octets stockant les données

Stockage des données

Après les avoir réceptionnées, il faut valider les données en vérifiant qu'elles viennent bien d'une console (avec le fichier xml où sont enregistrées ces dernières) et que ce sont bien des données de type sortie classique de la console (avec la taille et les 4 premiers octets donnant le type), cela est effectué par la méthode 'VerifierDatagramme()'. Ensuite 'ReceptionnerDatagrammes()' appelle la méthode 'TraiterDatagramme()'(QByteArray).

Méthode TraiterDatagramme()

```
int PlaybackWing::TraiterDatagramme(const QByteArray &datagramme)
{
   DecoderTouchesControle((unsigned
char)datagramme[INDEX_TOUCHES_CONTROLE]);
   ExtraireFaders(datagramme);

   emit envoyerEtatTouchesControle(etatTouchesControle);
   emit envoyerEtatFaders(etatFaders);

   return 1;
}
```

Cette méthode reçoit en paramètre une référence vers les données, cela permet de modifier de façon permanente les données lors de l'exécution des méthodes 'DecoderTouchesControle()' et 'ExtraireFaders()' qui vont traduire le datagramme en données exploitables.

Méthode DecodertouchesControle()

```
void PlaybackWing::DecoderTouchesControle(unsigned
char champTouche)
{
   // Touches de contrôle (INDEX_TOUCHES_CONTROLE 6)
   /*
       Bit 7: 0=PageUp key pressed, 1=PageUp key
released.
       Bit 6: 0=PageDown key pressed, 1=PageDown key
released.
       Bit 5: 0=Back key pressed, 1=Back key
released.
       Bit 4: 0=Go key pressed, 1=Go key released
   */
   unsigned char masque = 0;

   #ifdef DEBUG_PlaybackWing
   qDebug("0x%02X", champTouche);
   #endif
   masque = 1 << PAGE_UP;
   etatTouchesControle.touchePageUp = (champTouche &
masque) >> PAGE_UP;
   masque = 1 << PAGE_DOWN;
   etatTouchesControle.touchePageDown = (champTouche
& masque) >> PAGE_DOWN;
   masque = 1 << BACK;
   etatTouchesControle.toucheBack = (champTouche &
masque) >> BACK;
   masque = 1 << GO;
   etatTouchesControle.toucheGo = (champTouche &
masque) >> GO;

}
```

Déclaration du masque

Contient la valeur 7 car 7eme bit

Enregistrement de l'état dans une structure de type EtatTouchesControle

Etant donné que l'état des touches de contrôle est envoyé sur un bit, il faut utiliser un masque, les valeurs de chaque bit sont dans des constantes comme 'PAGE_UP' ou la valeur 7 est stockée. Après avoir été extraites, les valeurs sont mises dans une structure de type 'EtatTouchesControle'.

Extrait méthode Extrairefaders()

```
etatFaders.fader0 = (unsigned char)datagramme[FADER_NUMERO_0];
```

Comme les valeurs des faders sont stockés sur un octet en hexadécimal, il n'est pas nécessaire de faire de calcul, les valeurs sont stockés en castant en char les données. Chaque octet est identifié par une constante définie en amont comme 'FADER_NUMERO_0'.

La méthode 'TraiterDatagramme()' envoie ensuite les signaux 'envoyerEtatTouchesControle' et 'envoyerEtatFaders' qui sont connectés respectivement aux slots 'afficherEtatTouchesControle()' et 'afficherEtatFaders()'.

Ces deux slots sont très basiques, ils influent uniquement sur les sliders personnalisés de l'interface qui lorsque changé, envoi automatiquement leurs états à l'interface.

Extrait de la méthode afficherEtatTouchesControle()

```
    if(!etatTouchesControle.toucheBack &&
!etatTouchesControle.touchePageUp)
    {
        int index = listeProjecteursPilotage->currentIndex();
        if(index != listeProjecteursPilotage->count() - 1)
        {
            listeProjecteursPilotage->setCurrentIndex(index + 1);
        }
        else qDebug() << "Dernier projecteur atteint";

    }
```

Test de l'état des touches Back et Up

Incrementation de la liste des projecteurs à piloter

La méthode traduit les différents états des touches de contrôles comme ci dessus avec la combinaison de touches Back + Up, en fonctionnalité, ici, l'incrémentation de la liste des projecteurs.

Il en est de même pour 'afficherEtatFaders' qui affecte directement les valeurs de la structure 'etatFaders' aux sliders comme ci dessous.

```
    slidersCanaux[0]->setValeur(etatFaders.fader0);
```

Tests de validation

Désignation	Démarche à suivre	Résultat attendu	O/N	Remarques
Ajouter et Supprimer une console	-Ajouter une console depuis l'onglet 'Paramétrer les consoles' -Supprimer une console depuis l'onglet 'Paramétrer les consoles'	Ajout et suppression de la console sont tout 2 effectués	O	
Ajouter et Supprimer une interface	-Ajouter une interface depuis l'onglet 'Paramétrer les interfaces' -Supprimer une interface depuis l'onglet 'Paramétrer les interface'	Ajout et suppression de l'interface sont tout 2 effectués	O	
Commander les éclairages depuis l'application	-Sélectionner 'Piloter les projecteurs' -Changer les valeurs des sliders / canaux	Les projecteurs bougent/ changent de couleurs	O	Il faut sélectionner une interface USB / DMX fonctionnelle
Commander les éclairages depuis la console	Changer les valeurs des faders / canaux	Les projecteurs bougent/ changent de couleurs	O	Il faut sélectionner une interface USB / DMX fonctionnelle et enregistrer la console

Conclusion

Si on regarde les fonctions demandées dans le cahier des charges et le diagramme de Gantt créé au début du projet, le projet a atteint ses objectifs.

En terme d'amélioration, il pourrait être avantageux de développer une compatibilité Windows (car l'application n'est fonctionnelle que sur Linux) afin que la plupart des utilisateurs puissent l'utiliser. De plus, il serait intéressant de modifier la page de pilotage de l'IHM de sorte à ce qu'il y ait des widgets plus intuitif pour l'utilisateur comme, par exemple, une roue des couleurs pour modifier ces dernières.

Durant ce projet, j'ai pu apprendre par exemple à créer et configurer une communication UDP, j'ai aussi pu rencontrer et comprendre la norme DMX ce qui me permet de réaliser que de très nombreuses normes existent pour chaque domaine de travail, en l'occurrence l'éclairage de spectacle.

Glossaire

- Slider : Bouton coulissant, réfère aux widgets personnalisés présentés page 22.
- Fader : Bouton coulissant, réfère aux boutons coulissant de la console Playback Wing.
- Shutter : Diaphragme
- Gobo : Image, paterne
- PAN : Déplacement horizontal
- TILT : Déplacement vertical
- Stretch : Envergure
- Draw : Étirer
- Strobe : Stroboscope
- Lyre : Projecteur asservi pouvant tourner sur lui même horizontalement et verticalement
- PAR LED : Projecteurs basé sur la réflexion de la lumière uniquement.
- Scanner : Projecteurs composé d'un miroir tournant horizontalement et verticalement.
- Laser : Projecteurs composé d'un laser créant des formes.
- Broadcast : Mode de communication où la machine émet vers tous les autres appareils de son réseau.

Annexe

La documentation concernant la console et son protocole réseau ainsi que les projecteurs et la norme DMX est disponible en annexe de ce document.