ÉTUDE ET MISE EN ŒUVRE D'UN PARSEUR POUR LE STANDARD XMLTV

THEUREL Matthias
LEDUC Théo

Dans le cadre de notre formation par alternance au sein de Polytech' Tours, nous avions un projet de développement informatique à réaliser. Notre projet, la mise en œuvre d'un parseur pour le standard XMLTV, a été tutoré par Mathieu DELALANDRE.

Ce rapport présentera le travail effectué pour ce projet. Nous commencerons par l'étude. Nous continuerons avec une seconde partie dédiée à la conception avant de terminer avec la programmation.

Sommaire

Analyse 4
I/ Introduction au XML 4
II/ Gestion de projet 6
Développement 7
III/ Conception 7
a) Algorithme : 7
b) Solutions techniques : 7
c) Document Type Definition (DTD) 8
IV/ Programmation 10
a) Tests : 10
b) Rendu final sous terminal 16
c) Interface graphique 17

Analyse

I/ Introduction au XML

Le XML ou e**X**tensible **M**arkup **L**anguage est un langage informatique standardisé de balisage générique. Ce langage de description décrit et structure un ensemble de données selon des règles et des contraintes définies. Ce sont elles qui permettent la compréhension des documents XML par les hommes et les machines.

Nous travaillons sur le XMLTV français. C'est un document XML public qui répertorie les chaines de la TNT françaises et tous les programmes de chaque chaine.

Un document XML est composé de deux éléments principaux :

- Les **balises** sont les éléments de base. Elles sont définies par des chevrons « < » et « > » qui contiennent un nom. Elles fonctionnements par paires. Chaque balise ouvrante sera associée à une balise fermante. Par exemple, <programme> </programme> est une balise contenue dans le XMLTV.
  La balise qui encadre toutes les autres est appelé balise mère.
  Le contenu des balises se limite aux chaines de caractères, aux nombres mais peut aussi contenir d'autres balises appelées balises filles. On parle alors d'arborescence.

- Pour ajouter des informations aux balises, il est possible d'ajouter des **attributs**. Ce sont des informations supplémentaires sur la balise et sont optionnels.
  Par exemple, la durée d'un programme peut être mise dans une balise <durée> telle que <durée *minutes = 60*> </durée>.

Parsing robuste du standard xmltv

Balise « TV » et « Channel »

```
<tv source-info-url="https://api.telerama.fr" source-data-url="https://api.telerama.fr"
    generator-info-name="XMLTV France" generator-info-url="https://www.xmltv.fr/">
  <channel id="C111.api.telerama.fr">
    <display-name>Arte</display-name>
    <icon src="https://television.telerama.fr/sites/tr_master/files/sheet_media/tv/500x500/111.png" />
  </channel>
```

Balise « Programme »

```
<programme start="20190303150000 +0100" stop="20190303154500 +0100" showview="" channel="C111.api.telerama.fr">
  <title>Chemins d'école, chemins de tous les dangers</title>
  <sub-title>Colombie</sub-title>
  <desc lang="fr">Saison : 3 - A Vereda la Escuela, dans le nord de la Colombie, les enfants doivent traverser
                une région marécageuse ou naviguer en pirogue pour se rendre à l'école.</desc>
  <credits>
    <director>Nora Kocks</director>
  </credits>
  <date>2018</date>
  <category lang="fr">documentaire : education</category>
  <length units="minutes">45</length>
  <icon src="https://television.telerama.fr/sites/tr_master/files/sheet_media/media/aa4dc071-4dc4-402f-ab38-386248b5d49f.jpg" />
  <country>allemand</country>
  <episode-num system="xmltv_ns">2..</episode-num>
  <video>
    <aspect>4:3</aspect>
    <quality>HDTV</quality>
  </video>
  <audio>
    <stereo>bilingual</stereo>
  </audio>
  <previously-shown />
  <rating system="CSA">
    <value>Tout public</value>
  </rating>
</programme>
```

On retrouve ici les éléments décrits des documents XML dans notre document XMLTV français.

Il y a bien notre balise mère : la balise « tv » qui contiendra toutes les chaines et tous les programmes contenus dans des balises filles « channel » et « programme ».
La balise <tv> contient des attributs qui concernent l'api et l'url de provenance du XMLTV.

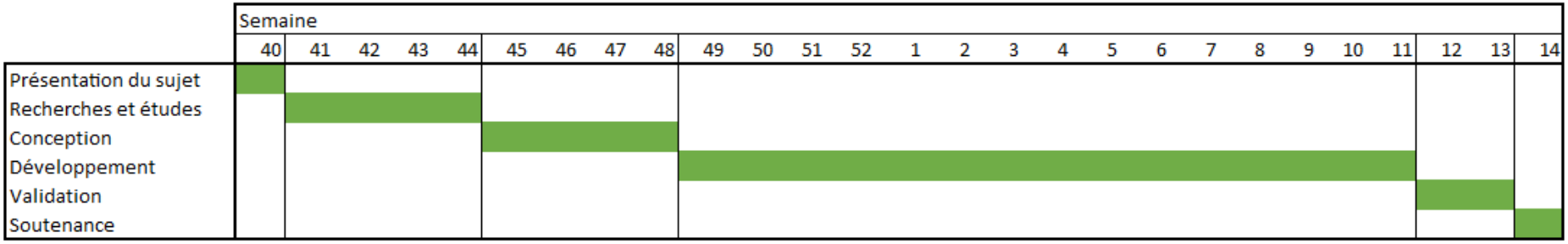
La balise <channel> a aussi un attribut. Il s'agit de son identifiant qui est unique. Elle contient deux informations dans ses balises filles : le nom de la chaine contenu dans la balise <display-name> et son logo dans la balise <icon> sous forme de lien URL.

La balise <programme> est la plus importante. Elle contient toutes les informations sur un programme, en commençant par la date de sa diffusion et la chaine à laquelle il appartient.

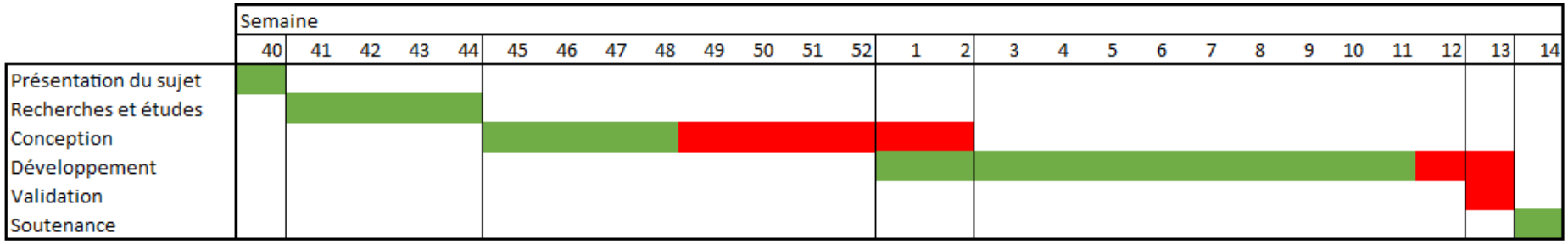
Dans les balises filles, on retrouve des informations comme le titre du programme, sa description, sa langue, sa date, sa durée, ses crédits, sa qualité sonore et vidéo, etc. Les programmes ne contiennent en général pas toutes les informations possibles. Dans notre exemple, nous n'avons pas la balise <subtitles> (à ne pas confondre avec <sub-titles>) qui contient les informations sur les sous-titres du programme.

II/ Gestion de projet

La gestion d'un projet est essentielle pour la réussite d'un projet. Après avoir étudié le XMLTV et commencé à réfléchir à nos choix techniques, nous avons défini les étapes importantes pour répondre correctement à la demande du client. Trois étapes se distinguent : les recherches liées au projet, la conception et le développement, la partie la plus importante du projet. Ces étapes seront approfondies dans la suite du rapport.



Le projet ne s'est pas déroulé de la façon convenue. Suite à des problèmes rencontrés en début de projet, notamment le langage C que nous avions initialement prévu, nous avons accumulé du retard et changé nos priorités, en accord avec notre client. La mise en œuvre d'un parseur pour le XMLTV est devenue notre priorité.



Développement

III/ Conception

La phase de conception représente le socle du développement. En effet, il permet de préciser les fonctionnalités attendues et choisir les solutions les plus pertinentes pour le projet en question.

a) Algorithme :

Afin de définir au mieux les actions que doit satisfaire le programme ainsi que leurs enchainements, un algorithme est nécessaire. Ci-dessous, ledit algorithme :

```
Charger le fichier XML;
ControlerValidation(XML,DTD);
Si ControlerValidation est vrai
        Affiche la validité;
        Obtenir les informations du fichier;
        Enregistrer les informations;
        Traiter les informations;

Sinon

        Afficher la non validité;
```

Description : *On souhaite dans un premier temps charger le fichier XML à analyser. En vérifiant sa conformité par rapport à un fichier .DTD, on confirme, ou non sa validité. En fonction de cette dernière, on effectue les fonctions associées. Cet algorithme est très simple puisqu'il n'est constitué que d'une condition de validation. Dans les parties suivantes, nous verrons l'application de cet algorithme.*

b) Solutions techniques :

Nous avions trois grandes solutions à chercher. En effet, nous devions choisir notre langage de programmation et donc notre IDE. Dans un premier temps, nous avons décidé de développer en C puisque nous avions eu des cours dans ce langage. Pour des raisons d'implémentation de la bibliothèque utile au parsing, nous avons décidé de changer de langage, une cause des retards que nous avons eu.

La solution choisie était donc le **C#**, pour plusieurs raisons. En effet, le cahier des charges ne stipulait pas de langage obligatoire. Ainsi, nous avions un très large éventail de possibilités. La deuxième raison est que l'implémentation simplifiée du parser. En effet, le C# fournit une bibliothèque interne, les fonctions convenaient très bien à nos attentes en termes de **contrôle**, de **segmentation du fichier** tvguide.xml et d'**affichage**. Ce dernier point fait du C# un choix de taille dans l'implémentation d'une interface graphique puisqu'en utilisant **Visual Studio** il est possible de créer un projet Windows **Form Application** nous permettant de lier notre projet en C# avec une bibliothèque d'objet graphique en .NET directement exécutable sous Windows.

Parsing robuste du standard xmltv

Ce choix étant fait, nous devions à présent choisir notre technologie de parsing. Comme dit précédemment, nous avons le choix entre le **SAX** et le **DOM**. C'est cette dernière technologie qui a été choisie pour plusieurs raisons. Les retards accumulés dû à notre mauvaise prise en compte de la partie concernant la gestion du projet, le temps ne nous permettait plus de partir sur une technologie SAX qui requiert plus de temps pour l'implémentation, notamment à cause de la gestion évènementielle du fichier XML à traiter. Deuxièmement, comme nous étions partis sur le développement d'une interface graphique, le DOM est plus simple à intégrer grâce à son architecture en forme d'arbre. Néanmoins, le cahier des charges stipulait un développement en SAX. Nous n'avons pas respecté ce point. Comme dit précédemment la mauvaise gestion du projet nous a posé beaucoup de problèmes. Le développement SAX est possible en C#, la même bibliothèque de parser le permet.

c) Document Type Definition (DTD)

Un fichier DTD correspond à une structure que doit suivre un fichier contenant du code en langage de balisage extensible, exemple, le **XML.** Chaque balise du fichier à vérifier va être déclarer dans le DTD en lui attribuant un type de donnée. Le tableau ci-dessous nous permet de comprendre les différents types présents dans notre fichier DTD.

Type de donnée	Description	*Type de variable*
<ELEMENT>	*Ce type de donnée va définir si un élément fichier XML doit être considéré comme une balise.* *Lors d'une déclaration d'une donnée de type <ELEMENT>, les balises filles sont déclarées entre parenthèses.*	✓ **#PCDATA** est une donnée de type caractère mais
<ATTLIST>	*Celui-ci définit un attribut d'une balise. Ce type de donnée doit être suivi d'une déclaration du type de la variable.*	✓ **#CDATA** est une donnée de type caractère.

Ce fichier va donc nous servir pour la validation de notre fichier XML avant de procéder à sa segmentation.

La question que nous devons nous poser à présent est la suivante : **Que doit contenir le fichier DTD afin de valider le fichier XML ?**

La réponse réside dans les informations que contient le fichier XML. En effet, le fichier DTD av définir tous les éléments (balises) du fichier XML à valider. Notre fichier étant très conséquent, le DTD sera proportionnellement conséquent, et donc va prendre du temps à créer. Or, une archive regroupant des informations sur le standard xmltv est disponible sur internet. A l'intérieur de cette archive, on retrouve un fichier DTD générique pour ce standard. Voici donc trois blocs définissant les éléments définissant les balises **tv,** et **programme.**

Parsing robuste du standard xmltv

➢ *Balise « tv » :*

```
<!ELEMENT tv (channel*, programme*)>
<!ATTLIST tv date    CDATA #IMPLIED
             source-info-url     CDATA #IMPLIED
             source-info-name    CDATA #IMPLIED
             source-data-url     CDATA #IMPLIED
             generator-info-name CDATA #IMPLIED
             generator-info-url  CDATA #IMPLIED >
```

On retrouve ici la définition de la balise racine tv avec les deux balises filles possibles, « channel » et « programme »

Voici donc la définition des attributs de la balise tv. Le type ATTLIST définie la liste d'attributs. Chaque nom précédent le type de variable correspond au nom d'un attribut. Viens ensuite le type de variable qui est ici de type caractère. Le **#IMPLIED** signifie que l'attribut correspondant n'a pas obligatoirement besoin d'une valeur texte pour être validé.

➢ *Balise « programme » :*

```
<!ELEMENT programme (title+, sub-title*, desc*, credits?, date?,
                     category*, keyword*, language?, orig-language?,
                     length?, icon*, url*, country*, episode-num*,
                     video?, audio?, previously-shown?, premiere?,
                     last-chance?, new?, subtitles*, rating*,
                     star-rating*, review* )>
<!ATTLIST programme start     CDATA #REQUIRED
                    stop      CDATA #IMPLIED
                    pdc-start CDATA #IMPLIED
                    vps-start CDATA #IMPLIED
                    showview  CDATA #IMPLIED
                    videoplus CDATA #IMPLIED
                    channel   CDATA #REQUIRED
                    clumpidx  CDATA "0/1" >
```

On déclare ici la balise programme. Comme pour la balise « tv », celle-ci possède des balises filles qui sont plus nombreuses. Chaque balise fille correspond à un paramètre caractéristique d'un programme télé.

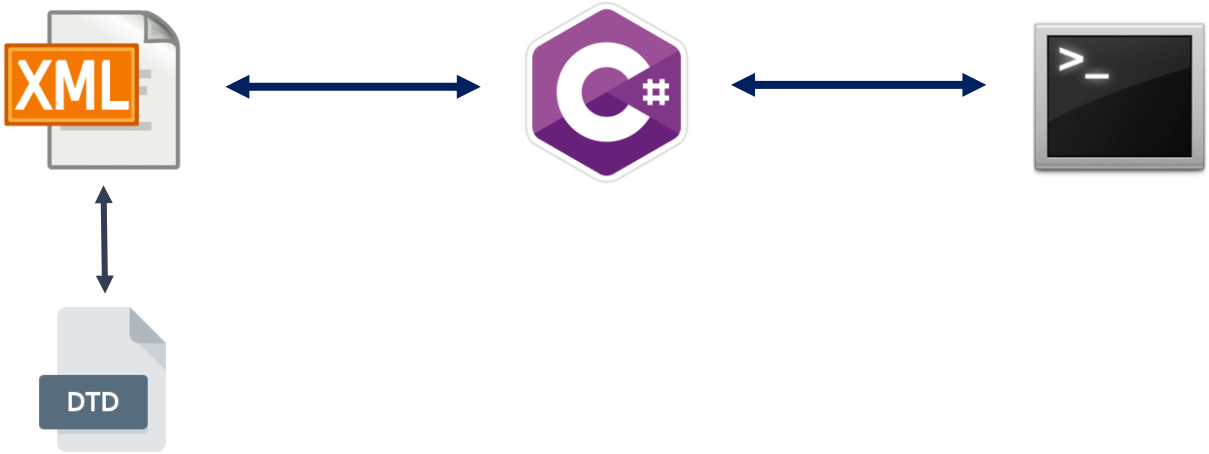
Comme pour la balise « tv », on vient déclarer tous les attributs de la balise programme. On retrouve aussi le type de variable CDATA et la déclaration #IMPLIED. Cependant, on retrouve ici la déclaration **#REQUIRED ;** celle-ci veut dire que les attributs correspondant doivent obligatoirement avoir une valeur texte à l'intérieur pour que la structure soit validé. En effet, un programme doit avoir une heure de début et une chaine, au minimum, pour être déclaré.

IV/ Programmation

Une fois notre conception terminée, nous avons pu dégager les éléments nécessaires au développement de notre projet, en termes de langage, d'IDE et de technologie de parsing. Il est temps de commencer la programmation. Nous allons orienter cette dernière selon deux axes. Premièrement, une partie **Test** pour valider la partie extraction de données du fichier, ainsi que la partie vérification.

a) Tests :

La première chose que nous avons fait en termes de test était de comprendre comment nous allons restituer les informations du fichier. Voici donc, ci-dessous, les différents acteurs au sein de notre programme.



Voici donc le principe d'application que nous allons suivre. Comme dit précédemment dans notre algorithme, la première étape est la validation de la structure du fichier .xml contre un fichier .dtd. Après validation, un fichier C# assure la gestion de données provenant du fichier.xml validé. Ce même fichier C# viendra gérer l'affichage sur une invite de commande dans un premier temps.

Vérification :

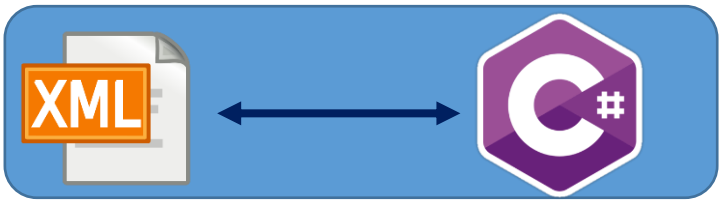
La ligne ci-dessous vient définir le fichier contre lequel le fichier xml sera validé. Elle est incluse en entête de ce dernier. On y voit le nom du fichier .dtd (ici **xmltv.dtd**).

```
<!DOCTYPE tv SYSTEM "xmltv.dtd">
```

C'est l'instruction DOCTYPE qui permet ce lien.

Le but à présent est de spécifier au fichier C# que les deux fichiers sont validés, ou non, afin de procéder à la suite de l'algorithme.

Parsing robuste du standard xmltv



Voici donc la partie sur laquelle nous allons nous concentrer. Au sein de notre programme, la fonction **ReadXMLWithDTD** permet cette validation.

```
public static void ReadXMLwithDTD()
{
    // Configuration du vérificateur
    XmlReaderSettings settings = new XmlReaderSettings();
    settings.ValidationType = ValidationType.DTD; //Désignation type de validation
    settings.DtdProcessing = DtdProcessing.Parse; //On choisi d'utiliser le dtd pour parser le fichier
    settings.XmlResolver = new XmlUrlResolver();
    settings.ValidationEventHandler += new ValidationEventHandler(DisplayErrors);
    settings.IgnoreWhitespace = true;
    //=====================================
```

- La ligne contenue dans le cadre bleu permet d'instancier le nouveau gestionnaire de paramétrage pour la lecture du fichier xml.
- La première ligne du cadre rouge permet de définir le type de fichier nécessaire pour la validation du fichier xml. En l'occurrence, la méthode ValidationType est suivie de la désignation .DTD car nous validons le fichier xml contre un fichier dtd. La deuxième ligne désigne le traitement voulu avec le dtd. On choisit le parsing du fichier xml.
- La ligne dans le cadre vert permet de définir un gestionnaire d'erreur que nous allons détailler à présent.

```
    // Parse parser le fichier
    while (reader.Read())
    {

        switch (reader.NodeType) //Type de noeud
        {
            case XmlNodeType.Element: //Compte le nombre d'élément dans le fichier
                cptElement = cptElement + 1;
                BarProgression.Value++;
                break;
            case XmlNodeType.Text: //Compte le nombre de valeurs de text
                cptText = cptElement + 1;
                break;

        }

        switch (reader.Name) //Type de text
        {
            case "channel":
                cptChaine = cptChaine + 1; //Compte le nombre de balises chaine
                break;
            case "programme":
                cptProgramme = cptProgramme + 1; //Compte le nombre de balises programme
                break;
        }

    }

}
```

La partie de code ci-contre correspond à la suite de la fonction **ReadXMLwithDTD.** Cette boucle « while » vient contrôler les différents éléments dans le fichier xml TANT QU'il reste un élément à lire dans le fichier.

Dans le premier switch, le but est ici de compter le nombre de balises présentes dans le fichier ainsi que les valeurs textes afin de contrôler que tous les éléments soient reconnus. Tandis que le deuxième switch vient compter le nombre de chaines et de programmes qui vont nous aider dans les fonctions suivantes. Le switch est une instruction de sélection qui choisir une section d'opérations à exécuter en fonction d'une liste de possibilités (« case »).

Parsing robuste du standard xmltv

Comme dit précédemment, il existe un gestionnaire d'erreurs (« ValidationEventHandler »). Ce gestionnaire va nous permettre de détecter une erreur dans la comparaison entre le xml et le fichier dtd. La fonction **DisplayErrors** ci-dessous, permet d'afficher les erreurs rencontrées. De plus, à chaque erreur rencontrée, un compteur vient s'incrémenter afin de compter le nombre total d'erreur. Enfin, nous savons qu'en cas de non validité de la structure du fichier xml, les fonctions de récupération de données ne peuvent se faire. L'algorithme vu précédemment nous informe sur ce contrôle. Le booléen **IsValid** permet le contrôle et donc va exécuter des fonctions différentes selon son état. Dans le cas d'une détection d'erreur, ce dernier passe à faux (ici « false »).

Fonction DisplayErrors :

```
//Gestionnaire des messages d'erreur
1 référence
private static void DisplayErrors(object sender, ValidationEventArgs e)
{

    if (e.Severity == XmlSeverityType.Warning)
        MessageBox.Show("Warning: Matching schema not found.  No validation occurred." + e.Message);
    else // Error
        IsValid = false;
        cptErreurs = cptErreurs + 1;


}
```

Main :

La fonction main est très courte. En effet, elle fait appel aux différentes fonctions nécessaires dans l'algorithme.

```
//=============  MAIN  ====================
0 références
static void Main()
{

    ReadXMLwithDTD();
    if (IsValid == true)
    {
        DocIsValid();

        DisplayInfos();
        TVGuideSerializer();

    }
    else
    {
        DocIsNotValid();
    }

}

//=====================================
```

Remarque : *La structure de cette fonction est identique à l'algorithme élaboré. Le booléen IsValid détaillé ci-dessus intervient ici, et on distingue bien les deux axes que peut prendre le projet.*

Nous allons donc voir maintenant les différentes fonctions qui composent ce main().

Parsing robuste du standard xmltv

Notifier l'utilisateur que l'analyse du fichier XML contre DTD s'est bien déroulé :

```
//Affichage document valide
1 référence
private static void DocIsValid()
{

    Console.ForegroundColor = ConsoleColor.Green;
    Console.WriteLine("Document is valid ");
    Console.ResetColor();
    Console.WriteLine(" Errors occured :" + cptErreurs);
    Console.ForegroundColor = ConsoleColor.Gray;

}
```

Cette fonction permet donc de notifier l'utilisateur de la validation du document analysé et contrôlé. On vient afficher un message et aussi le nombre d'erreurs rencontrées lors du contrôle contre le DTD. Le compteur que nous voyons ci-contre est le même compteur que l'on vient incrémenter à chaque erreur occasionné dans la fonction **DisplayErrors().**

Lors de la consultation d'un programme TV, on souhaite savoir le nombre de chaines mises à disposition de l'utilisateur. C'est ce que nous faisons ci-dessous. De plus, on rajoute le nombre d'émissions disponibles au total pour le xmltv France exclusivement.

De plus, pour des questions de contrôle du fichier, on vient rajouter le nombre de balises reconnues ainsi que le nombre de valeurs textes.

```
private static void DisplayInfos()
{
    System.Console.WriteLine("=============== INFORMATIONS ===============");
    //Affichage du nombre d'éléments
    Console.Write("Number of Elements:");
    Console.ForegroundColor = ConsoleColor.Yellow;
    System.Console.WriteLine(cptElement);
    Console.ResetColor();


    //Affichage des nombres respectifs de chaines et de programmes
    Console.WriteLine("     with...");
    Console.ForegroundColor = ConsoleColor.Blue;
    Console.Write("     Number of Channels:");

    /*==========================================================================================
     * Division du compteur par deux car pour déclarer une chaine il faut une balise ouverture et fermeture
     * Or ici on veut le nombre de chaines. Il faut donc diviser le compteur par deux
     ==========================================================================================*/
    System.Console.WriteLine(cptChaine/2);
    Console.Write("     Number of Programs:");
    System.Console.WriteLine(cptProgramme/2);
    Console.ResetColor();


    //Affichage du nombre de valeurs text
    Console.Write("Number of Text values:");
    Console.ForegroundColor = ConsoleColor.Yellow;
    System.Console.WriteLine(cptText);
    Console.ResetColor();


    System.Console.WriteLine("==========================================");
}
```

*En **jaune**, les compteurs correspondants respectivement au nombre de balises présentes dans le fichier, le nombre de chaine (divisé par deux car il existe une balise ouvrante et une fermante par chaine, et l'analyse de différencie pas les deux), le nombre de programmes (même cas que pour les chaines), et enfin le nombre de valeurs textes (présent entre une balise ouvrante et une fermante).*

Parsing robuste du standard xmltv

Récupérer les informations des chaines/programmes dans le fichier :

Maintenant que nous avons le nombre de chaines ainsi que le nombre de programmes disponibles, la problématique est d'extraire les informations correspondantes à chaque programmes/chaines. C'est le rôle de la fonction **TVGuideSerializer().**

Le plus complexe dans cette fonction est la mise en place de la liaison entre le programme C# et le fichier xml. La première étape est de définir les types de balises présentes dans notre fichier. Nous allons donc prendre l'exemple de la définition des balises filles de la balise même « programme ».

```
[Serializable]
2 références
public class programme
{
    [XmlElement(ElementName = "title")]
    1 référence
    public string title { get; set; }

    [XmlElement(ElementName = "sub-title")]
    0 références
    public string SubTitle { get; set; }

    [XmlElement(ElementName = "desc")]
    1 référence
    public string desc { get; set; }

    [XmlElement(ElementName = "credits")]
    0 références
    public List<credits> credits { get; set; }

    [XmlElement(ElementName = "length")]
    1 référence
    public int length { get; set; }

    [XmlElement(ElementName = "icon")]
    1 référence
    public List<icon> icon { get; set; }
```

*Ci-contre, ces balises filles précédemment énoncées. Si l'on reprend la structure DTD présentée précédemment, on remarque que toutes les balises filles sont présentes. La classe **XmlElement** permet d'instancier une balise en lui précisant donc son nom. On lui affecte ensuite une variable que le fichier C# va pouvoir manipuler sans toucher au contenu du fichier XML. Les éléments entre accolades permettent de définir les permissions de chaque variable. La permission **get** permet la lecture de la valeur de cette variable et la permission **set** permet son écriture.*

Ce principe s'applique aussi pour les balises « tv » et « chaine » du fichier.

Maintenant que le fichier C# reconnait les balises du fichier XML, il faut définir le flux de données qui va transmettre les valeurs de chaque balise du fichier XML au fichier C#.

La portion de code ci-dessous représente cette liaison. On vient ici définir une méthode **XMLSerializer** qui permet d'extraire les données afin de restituer des objets du type donné en argument. Ainsi, on définit ici des objets de type « tv » qui représente le nom de la classe définissant les balises filles de la balise racine tv (*c.f définition de la classe programme ci-dessous*).

```
XmlSerializer xmlSerializer = new XmlSerializer(typeof(tv));
StreamReader SerializeReader = new StreamReader("tvguide.xml");
tv xmltv = (tv)xmlSerializer.Deserialize(SerializeReader);
```

Parsing robuste du standard xmltv

La ligne numéro 2 instancie le flux de données. On vient seulement préciser le nom du fichier que depuis lequel extraire les données. Enfin, on définit une variable de type tv que viendra prendre comme valeur toute donnée du flux.

```
foreach (programme prog in xmltv.programme)
{

    Console.WriteLine("=====================================");
    //====================Affichage titre du programme=======================
    Console.ForegroundColor = ConsoleColor.Green;
    Console.Write("Programme: ");
    Console.ForegroundColor = ConsoleColor.Gray;
    Console.WriteLine(prog.title);

    //===================Affichage directeurs/ presentateurs
    foreach(credits cred in prog.credits)
    {
        Console.ForegroundColor = ConsoleColor.Yellow;
        Console.Write("     Directeurs : ");
        Console.WriteLine(cred.director);

    }


    //====================Affichage Description du programme=================
    Console.ForegroundColor = ConsoleColor.DarkGreen;
    Console.Write("     Description: ");
    Console.ForegroundColor = ConsoleColor.Gray;
    Console.WriteLine(prog.desc);
}
```

*Le code ci-dessus affiche le titre ainsi que le titre de chaque programme. Le plus important ici est de comprendre l'accès aux différentes informations. La boucle **foreach** ci-dessus permet d'afficher le titre et la description **pour chaque** programme présent dans la balise tv. En effet, c'est ce que nous renseigne la condition de la boucle.*

Précédemment, nous avions défini une variable « xmltv » de type tv. On la retrouve ici, suivi de la méthode programme qui est une des variables définies dans la classe tv correspondante. De plus, on vient instancier une nouvelle variable prog de type programme. Comme pour la variable xmltv, la variable « prog » aura accès aux méthodes de la classe programme définie précédemment. On y retrouvera donc les méthodes « .title », « .credits » pour les directeurs, et « .desc » pour les descriptions.

Parsing robuste du standard xmltv

b) Rendu final sous terminal

```
Document is valid
 Errors occured :0
============== INFORMATIONS ==============
Number of Elements:135348
     with...
     Number of Channels:28
     Number of Programs:8106
Number of Text values:135349
==========================================
Chaine 1: Arte
Chaine 2: LCI
Chaine 3: M6
Chaine 4: W9
Chaine 5: Chérie 25
Chaine 6: RMC Découverte
Chaine 7: L'Equipe
Chaine 8: RMC Story
Chaine 9: 6ter
Chaine 10: TF1 Series Films
Chaine 11: Paris Première
Chaine 12: France Ô
Chaine 13: TF1
Chaine 14: TMC
Chaine 15: Franceinfo
Chaine 16: CNEWS
Chaine 17: La Chaîne parlementaire
Chaine 18: Canal+
Chaine 19: France 2
Chaine 20: NRJ 12
Chaine 21: C8
Chaine 22: TFX
Chaine 23: CSTAR
Chaine 24: France 5
Chaine 25: BFM TV
Chaine 26: Gulli
Chaine 27: France 4
Chaine 28: France 3
=====================================
Programme: X:enius
     Directeurs :
     Description: En action 24 heures sur 24, les reins constituent la station d'épuration du corps, filtrant les toxines da
ns le sang, contrôlant l'équilibre des fluides et régulant la tension artérielle. En cas de problèmes rénaux, les conséquenc
es peuvent se révéler dramatiques. Les présentateurs cherchent à comprendre ce qui affecte leur fonctionnement et partent en
 quête de solutions pour les protéger.
=====================================
Programme: Des monuments et des hommes
     Directeurs : Véronique Legendre
     Description: A Kyoto, le célèbre temple de Daitoku-ji et ses jardins constituent un hymne à la nature, qui joue le rôle
 d'intermédiaire entre le profane et le sacré.
=====================================
```

En haut de l'écran, nous retrouvons bien les informations sur les erreurs rencontrées, le nombre de balises présentes ainsi que le nombre de chaines et de programmes présents.

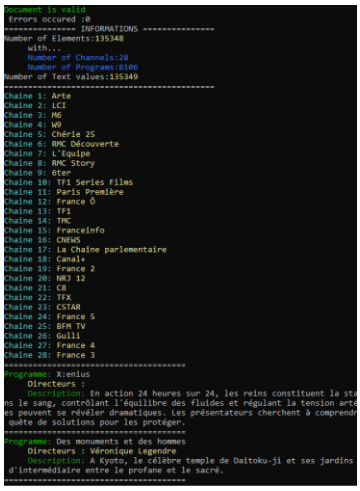
Au milieu de l'écran, toutes les chaines répertoriées dans le fichier, suivie de tous les programmes (impossible d'afficher tous les programmes au vu du nombre conséquent d'éléments).

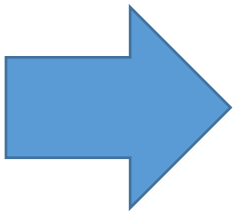
Nous avons donc bien répondu aux attentes de tests, la vérification du fichier xml contre une structure dtd fonctionne et l'extraction de données aussi. Cependant, c'est une liste d'informations, il est impossible de parcourir efficacement le fichier. Les interactions homme machines sont nécessaires. C'est pourquoi, nous allons voir à présent le développement de l'interface graphique correspondante.

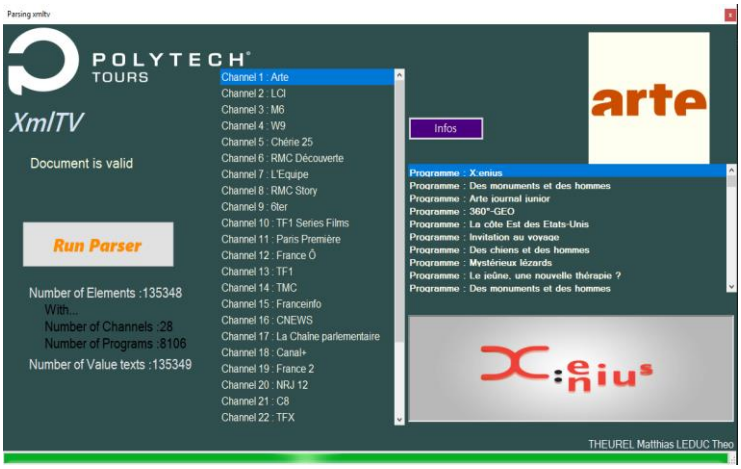
Parsing robuste du standard xmltv

c) Interface graphique

Nous avons donc vu précédemment qu'il était possible de sérialiser (extraire) les données du fichier et d'en valider sa structure. Il est nécessaire à présent d'optimiser la navigation dans toutes ces informations, et donc de passer d'une interface terminale, à une interface homme machine.







Comme dit dans la phase de conception, l'IDE Visual Studio fournit un outil de conception d'interface simplifié. En additionnant les fonctionnalités VB.NET à nos fonctions C#, nous obtenons une interface **Windows Forms.**

Le but ici est d'optimiser la navigation de l'utilisateur tout en ajoutant les éléments vus dans l'interface sous terminale. Nous allons donc voir l'affichage des programmes correspondants à chaque chaine, ainsi que la gestion de la bascule d'image entre deux programmes différents.

<u>Afficher les programmes en fonction de la chaine voulue</u> :

Afin de répondre à notre problématique, nous devons tout d'abord afficher les chaines. Rien de plus simple. Nous avons besoin d'une **ListBox()** qui correspond à un listing. Chaque ligne de cette ListBox se verra attribué avec un élément de la liste de chaines précédemment évoqué.

La ListBox présente une fonctionnalité intéressante. En effet, chaque élément de cette liste se verra attribué un identificateur unique. Ainsi, la chaine **Arte** se verra remettre l'identificateur **0**, et ainsi de suite. Cette fonctionnalité va nous permettre de détecter l'action de l'utilisateur et d'afficher les éléments de la chaine en conséquence, autrement dit l'icône de la chaine et ses programmes.

Le problème maintenant est d'afficher le programme **si** la chaine à laquelle il est rattaché a été cliqué par l'utilisateur. Dans le fichier XML, chaque chaine est dotée d'un identifiant, stockée dans la balise « id ». On vient enregistrer chaque identifiant de la chaine dans un tableau. Le choix du tableau se justifie car chaque identifiant sera ajouté dans l'ordre dans le tableau, incrémentant par la même occasion son index. On aura donc à chaque clique d'un élément de la ListBox, l'identifiant correspondant à la chaine dans un tableau.

Parsing robuste du standard xmltv

Ex : la chaine Arte aura son identifiant de chaine stockée à la case 0 du tableau et ainsi de suite.

```
1 référence
private void AffichChaine_SelectedIndexChanged(object sender, EventArgs e)
{
    ClearTabsForProgram();
    /*==========================================
     * on récupère l'id de chaque élément de la liste
     * ==========================================*/
    id = this.AffichChaine.SelectedIndex;
    InfosProgr.Show(); //Afficher le bouton "Infos"
    PrintIconeChannel(id); //Afficher les icones des chaines
    PrintPrograms(id); //Afficher les programmes

}
```

*Ci-contre, la représentation des fonctions appelées lors de la sélection d'un élément dans la ListBox. On retrouve donc la fonction PrintIconeChannel(id) qui, comme son nom l'indique, affiche les icones des chaines pour l'utilisateur. La variable **id** correspond à l'identificateur unique de chaque élément présent dans la ListBox.*

Un des problèmes actuels est que les tableaux stockant les informations sur les programmes saturent vite. Cela est dû au fait que l'utilisateur change une chaine et que les tableaux ont toujours les valeurs des programmes des chaines précédemment cliqués en mémoire. Pour résoudre ce problème, nous avons créé la fonction **ClearTabsForProgram()** permettant de réinitialiser toutes les valeurs des tableaux à 0. Cela permettant donc de stocker seulement les informations concernant la chaine sur laquelle l'utilisateur à sélectionner.

Nous allons à présent nous concentrer sur l'affichage des programmes. Cela se fait dans la fonction **PrintPrograms().**

```
1 référence
public void PrintPrograms(int IconIdChannel) //Cherche les programmes en fonction de la chaine voulue
{
    XmlSerializer xmlSerializerPrograms = new XmlSerializer(typeof(tv));
    StreamReader SerializeReaderImage = new StreamReader("tvguide.xml");
    tv xmltvPrintPrograms = (tv)xmlSerializerPrograms.Deserialize(SerializeReaderImage);

    AffichProgr.Items.Clear(); //Nettoyage des informations dans la listbox

    foreach (programme prog in xmltvPrintPrograms.programme) //on récupère les programmes dans le fichier xml
    {
        foreach (icon iconprogr in prog.icon)
        {

            if (prog.channel == idChannelTab[IconIdChannel]) {// on compare la valeur de l'attribut channel de chaque programme à

                AffichProgr.Items.Add("Programme : " + prog.title);
                UrlIconProgramTab[compteurForUrlProgram] = iconprogr.src; //Enregistrer l'url de chaque programme affiché
                DescrTab[compteurForUrlProgram] = prog.desc; //Enregistre la description de chaque programme de la chaine choisie
                LengthTimeTab[compteurForUrlProgram] = prog.length; //Enregistre la durée de chaque programme de la chaine choisie
                compteurForUrlProgram++;


            }
        }

    }

}
```

Parsing robuste du standard xmltv

*Comme pour l'affichage des chaines, on vient parcourir le fichier xml, et **pour chaque** programme dans la balise tv, on vient rechercher son icone (en plus des informations déjà récoltées comme la description, le titre et la durée d'une émission).*

*La condition **if** correspond ici à la condition voulue pour afficher un programme, c'est-à-dire que l'identifiant de la chaine stockée corresponde à la chaine associée au programme.*

Le compteur à la fin de la condition if permet de passer à l'emplacement suivant dans les différents tableaux pour stocker toutes les informations du programme suivant.

Afficher les informations de chaque programme :

Maintenant que nous savons comment manipuler l'affichage des informations selon le programme choisi, il ne nous reste plus qu'à afficher les informations de chaque programme (description, titre, durée, et Icone de chaque programme) depuis leur tableau respectif.

Comme pour le stockage des identifiants des chaines, la boucle **foreach** vient parcourir le fichier ligne par ligne. Ainsi les données stockées dans l'emplacement 1 de chaque tableau correspondent aux informations du 1er programme identifié. La ListBox affichant les programmes va donc nous permettre d'afficher les informations en fonction du programme cliqué. Ci-dessous, les fonctions enclenchées lorsque l'événement d'appuie sur un élément de la ListBox est détecté.

```
1 référence
private void AffichProgr_SelectedIndexChanged(object sender, EventArgs e) //Actions à effectuer sur click d'un programme
{

    idAffichProgr = this.AffichProgr.SelectedIndex;
    PrintIconeProgram(idAffichProgr);


}
```

Les informations seront affichées dans une fenêtre pop-up sur appuie du bouton Infos. Ci-dessous l'affichage des informations sur le programme sélectionné dans la ListBox.

```
1 référence
public void PrintInfosProg(int idInfosProg)
{
    MessageBox.Show(String.Format("Description : " + DescrTab[idInfosProg] + "\n\n" +
                                  "Durée : " + LengthTimeTab[idInfosProg] + "minutes"));
}
```

L'interface graphique remplie donc bien nos objectifs. Nous voulions afficher les informations de manière plus intuitive pour l'utilisateur tout en améliorant sa navigation. Maintenant que nous savons manipuler les données et les afficher, toute donnée présente dans le fichier peut être affichée.

Conclusion

Nous retirons beaucoup de ce projet. D'un côté nous avons pris conscience des efforts nécessaires à la réussite d'un projet en école d'ingénieur. Nous avons trop mésestimé certains points, en particulier la gestion de projet. Le changement du cahier des charges et une interaction réduite avec le client en sont les principales conséquences.

De l'autre nous avons beaucoup découvert concernant le développement informatique. Malgré nos difficultés en langage bas niveau, notamment l'intégration d'éléments externes tels que les bibliothèques, nous avons pu travailler sur un langage que nous ne connaissions pas.

Apports	Difficultés
• Auto formation en langage C# • Découverte du VB.NET • Découverte de l'outil de création d'interface sur Visual Studio • Expérience en programmation	➢ Pauvre gestion de projet ➢ Interaction avec le client ➢ Respect du cahier des charges

Les difficultés rencontrées ne sont pas nombreuses mais sont très handicapantes, surtout au niveau attendu. Ce sont des erreurs que nous corrigerons dans les futurs projets dont nous aurons la responsabilité au sein de Polytech ou en entreprise.