Projet Xmlia

BOIVIN Benoit
LE PHILIPPE Noé
KEGBA-SANGO-SANGO Ulrich-Chancelin
WOUTERS Stéphane

22 mai 2014

Résumé

A remplir !

Remerciements

Nous remercions tout particulièrement Michel Meynard pour nous avoir brillamment encadré et soutenu tout le long de la réalisation de ce projet.

Table des matières

Chapitre 1

Introduction

Chapitre 2

Analyse du projet

2.1 Contexte

Le langage Extensible Markup Language ou XML est utilisé à des fins de stockage de données, et est structuré par un schéma qui lui est associé, il permet de definir la structure et le type de contenu du document, en plus de permettre de verifier la validité du document.

Généralement, les fichiers XML sont générés par un programme quelconque dans le but d'échanger des données ou de les stocker, XML faisant office de plateforme commune. Mais on peut également utiliser un éditeur de texte basique pour créer de toute pièce un document XML, avec des fonctionnalités propre à un éditeur de texte, sans fonctionnalités spécialement prévues pour XML.

C'est dans ce contexte que des solutions logicielles d'éditeur XML ont vu le jour : un éditeur de texte qui possède des fonctionnalités permettant une écriture d'un fichier XML beaucoup plus rapide et efficace, le tout avec un contrôle des erreurs. Plusieurs solutions sont déjà proposées, certaines étant payantes (OxygenXML) et d'autres sont gratuites et open source (Xerlin). L'objectif est ici de fournir une solution similaire aux autres logiciels.

2.2 Analyse des besoins fonctionnels

L'objectif du projet est de développer un éditeur XML multi-vues avec différentes fonctionnalités.

Les fonctionnalités liées à un éditeur de texte simple devront être présentes : la possibilité de saisir manuellement au clavier l'intégralité du fichier, la création et la sauvegarde du fichier à manipuler ainsi que l'ouverture d'un fichier déjà existant dans le but de le modifier.

Des fonctionnalités d'éditeur de texte avancées seront aussi présentes : coloration syntaxique et indentation automatique du code permettant ainsi une lisibilité claire des fichiers manipulés et une autocomplétion du code écrit permettant un gain de temps au cours de la frappe.

Pour finir, l'éditeur proposera des fonctionnalités spécifiques au langage XML : validation syntaxique du fichier, vue arborescente du fichier XML avec possibilité de modification des données via cette vue et ajout d'un schéma sur lequel la validation se basera.

TODO : FAIRE UN SCHEMA D'INTERFACE

2.3 Analyse des besoins non fonctionnels

2.3.1 Spécifications techniques

Le programme devra permettre de créer des fichiers XML structurés avec un respect des normes de balisage et, s'il est défini, du schéma de données. De plus, les données saisies ou modifiées à l'aide de l'éditeur doivent rester exploitables, sans corruption du fichier original. Pour terminer, l'éditeur aura à répondre dans des durées acceptables et de manière stable, dans la mesure où la taille et la complexité des données restent raisonnables.

2.3.2 Contraintes ergonomiques

Le logiciel devra être suffisamment simple pour qu'un utilisateur connaissant déjà le fonctionnement du XML puisse l'utiliser sans être bloqué par une courbe d'apprentissage trop élevée. On utilisera pour cela des icônes claires et des textes explicatifs. Un utilisateur avancé pourra augmenter sa productivité en utilisant les raccourcis clavier disponibles et pourra gagner de temps en réduisant les transitions souris/clavier.

Chapitre 3

Rapport technique

3.1 Conception

3.2 Architecture de l'application

3.2.1 Le modèle

3.2.2 L'arborescence

3.2.3 L'éditeur de texte

L'éditeur de texte est décomposé en deux parties, l'éditeur de schéma et d'éditeur XML. Ce sont en réalité des spécialisation de la classe TextEditor.

La classe TextEditor

C'est ici que sont toutes les méthodes servant à l'édition du texte en général, telle que l'insertion de texte, l'indentation ou la coloration. Nous avons fait le choix de représenter les données de l'éditeur de texte simplement par une QString, pas de référence vers la position d'un noeud ou d'information supplémentaire comme sa taille ou la délimitation de son contenu. Un noeud étant identifié par son chemin depuis la racine, il faut reconstruire l'arborescence XML à partir de la QString. Cela se fait à travers la classe QXmlStreamReader qui s'utilise de la manière suivante :

```
QXmlStreamReader xml(text->toPlainText());
while(!xml.atEnd())
{
  if(xml.isStartElement())
  {
    //traitement
  }
  else if(xml.isEndElement())
  {
    //traitement
  }
}
```

On utilise une structure de pile lors de parcours de l'arborescence. On empile l'indice du sommet lorsque l'on rencontre une balise ouvrante et on dépile lorsque l'on rencontre une balise fermante. On parvient ainsi à se déplacer et à se repérer dans la QString.

Voici le pseudo code de l'algorithme permettant de parcourir l'arbre pour se rendre sur le noeud désiré.

```
//noeud que l'on doit trouver dans la QString
var noeudCible
var pile

//Parseur xml de Qt
var xml

Tant que l'on a pas parcouru tout l'arbre
    Si on rencontre une balise ouvrante
        Si la dernière balise rencontrée est une balise ouvrante
            Empiler 0 dans pile
        Sinon
            //c'est que l'on a atteint le fils suivant
            Incrementer le sommet de la pile de 1
        Fin Si
    Sinon Si on rencontre une balise fermante
        Dépiler pile
    Fin Si

    Si noeudCible = pile
        Appeler la fonction qui traitera le noeud
        //on arrête le parcours de l'arbre
        retourner
    Fin Si

    Sauvegarder la derière balise renconcrée
    Aller à la balise suivante
Fin Tant Que
```

Cette méthode de parcours de l'arbre est appelée lorsqu'un noeud est inséré, déplacé ou supprimé dans l'arborescence. Elle est utlisée de la manière suivante :

3.2.4 Le logger

3.3 Résultat

Chapitre 4

Rapport d'activité

4.1 Organisation du travail

4.1.1 Commmunication

La communication s'est au départ faite au travers de réunions hebdomadaires au cours desquelles le cahier des charges a été défini auprès de M. Meynard.

Une fois le cahier des charges défini et rendu, le but suivant a été de déterminer quel langage et quel bibliothèque d'affichage graphique sélectionner pour le projet et ainsi être conscient des avantages et des limites des éléments choisis.

Ensuite, l'objectif suivant a été de définir une maquette du logiciel afin de savoir quelles seront les fonctionnalités retenues, la manière de les exploiter et quel organisation visuelle du logiciel sera retenue. C'est ainsi qu'a été définie la maquette qui comporte une interface simple avec une barre d'outils, une vue principale sur le côté droit avec le système d'éditeur de texte, une vue secondaire sur la gauche avec la vue arborescente du modèle XML du fichier présenté par l'utilisateur et enfin la fenêtre de log associé au différents messages liés à l'utilisation de l'éditeur, par exemple, des erreurs de schéma.

Il fallait enfin définir l'organisation du travail au sein du groupe avec des tâches définies et mettre un place un plan de travail, c'est donc Stéphane WOUTERS, le chef du projet, qui a mis en place le gestionnaire de projet, a décidé des moyens de communication et qui a défini le premier objectif de développement : l'apprentissage de la librairie.

Puis la phase d'apprentissage et de développement commença, il fallait alors découvrir et apprendre le framework utilisé, et commencer à développer pour le projet, la communication s'est donc faite au travers de messages sur Google Hangouts, de mails, de commentaires via le gestionnaire de projet et de communication orale dans une salle informatique de la faculté.

4.1.2 Répartition des tâches

4.2 Outils de développement

4.2.1 Langage et bibliothèques

Nous avons utilisé Qt comme bibliothèque d'interface graphique.

4.2.2 IDE

Nous avons naturellement utilisé Qt Creator comme IDE, c'est en effet l'IDE

4.2.3 Gestionnaire de projet

Le gestionnaire de projet utilisé est Bitbucket, un site Internet d'hébergement mutualisé supportant des projets utilisant Mercurial ou Git comme gestionnaire de versions. Dans le cas de Xmlia, Git a été retenu et utilisé. (A FINIR)

Chapitre 5

Manuel d'utilisation

Chapitre 6

Perspectives et conclusions

6.1 Perspectives

6.2 Conclusions