

Projet Xmlia

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

24 mai 2014

Résumé

Ce rapport est le compte rendu du projet Xmlia, exécuté par les auteurs et proposé par Michel Meynard pour l'unité d'enseignement GLIN405 Projet Informatique du sixième semestre du parcours Licence Informatique de la Faculté de Sciences de Montpellier en 2013-2014.

Remerciements

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

Table des matières

1	Introduction	5
2	Analyse du projet	6
2.1	Contexte	6
2.2	Analyse de l'existant	6
2.3	Analyse des besoins fonctionnels	7
2.4	Analyse des besoins non fonctionnels	9
2.4.1	Spécifications techniques	9
2.4.2	Contraintes ergonomiques	9
3	Rapport d'activité	10
3.1	Organisation du travail	10
3.1.1	Communication	10
3.1.2	Répartition des tâches	10
3.1.3	Apprentissage d'une nouvelle API	11
3.2	Outils de développement	11
3.2.1	Langage et bibliothèques	11
3.2.2	IDE	12
3.2.3	Gestionnaire de projet	13
4	Rapport technique	15
4.1	Conception	15
4.1.1	L'architecture de Qt	15
4.1.2	Modèle de classe	15
4.2	Architecture de l'application	17
4.2.1	Le modèle	17
4.2.2	La vue arborescence	18
4.2.3	L'éditeur de texte	20
4.2.4	Le logger	26
4.2.5	Le gestionnaire de fichiers	28
4.3	Résultat	28
4.3.1	Fonctionnalités présentes	28
4.3.2	Captures	28
5	Manuel d'utilisation	30
5.1	Interaction	30
5.1.1	Barre de menu	30
5.1.2	Racourcis	30
5.2	Validation	30

5.3	Vue arborescente	31
5.4	A propos du schéma	31
5.5	Fonctions du logiciel	31
6	Perspectives et conclusions	32
6.1	Perspectives	32
6.2	Conclusions	32
6.2.1	Le projet	32
6.2.2	Les outils	33
6.2.3	Groupe de travail	33
7	Annexes	34
7.1	Code annexe	34
7.2	Historique des commits	34

Table des figures

2.1	Exemple de vue avec Oxygen	7
2.2	Diagramme des cas d'utilisation côté gestion des fichiers	7
2.3	Diagramme des cas d'utilisation de la vue arborescente	8
2.4	Diagramme des cas d'utilisation de l'éditeur de texte	8
2.5	Maquette d'interface	9
3.1	Apparence sous Linux	12
3.2	Apparence sous Mac Os	12
3.3	Apparence sous Windows	12
3.4	Exemple de ticket sur Bitbucket	14
4.1	Diagramme de classe	16
4.2	Vue arborescente de Qt	18
4.3	Rendu final de l'éditeur	29
4.4	Rendu final de l'éditeur	29
5.1	Différentes sections du logiciel	31
7.1	Historique des commits - Page 1	35
7.2	Historique des commits - Page 2	36
7.3	Historique des commits - Page 3	37
7.4	Historique des commits - Page 4	38
7.5	Historique des commits - Page 5	39
7.6	Historique des commits - Page 6	40

Chapitre 1

Introduction

Lorsqu'un programme nécessitait un stockage des données complexes et ordonnées, son créateur décidait de la manière dont les données étaient organisées en mémoire, permettant donc de définir le comportement spécifique de son application au moment de la lecture de données enregistrées. C'est dans ce contexte que le problème de la communication de données entre deux applications ou plus s'est posé. En effet, chaque programme avait sa manière d'interpréter des données stockées et les organisait de manière spécifique, la communication directe n'était donc pas possible, il fallait donc trouver un moyen intermédiaire afin de convertir les données destinées à une application vers un format lisible par un autre programme. Sauf que créer cet intermédiaire engendrait d'important coûts en termes de développement, d'autant plus que si une nouvelle application avait besoin de ces données, il aurait à nouveau fallu recréer un intermédiaire spécifique.

C'est ainsi que le principe d'une structure de données commune a émergé et que les langages de balisage se sont popularisés, permettant en plus d'avoir une structure stricte et normalisée. XML ou "Extensible Markup Language" fait partie de ces langages. Ce langage a de plus pour caractéristique d'être, d'où son nom, extensible, c'est-à-dire que les désignations des balises ne sont pas fixes, elles sont définies spécifiquement pour les données sauvegardées. Pour finir, il est possible de définir un XML Schema afin de restreindre et de contrôler la structure même du document XML, afin de vérifier s'il est écrit de la bonne manière et valide en termes d'organisation.

Le projet qui nous a été confié consiste à concevoir et à développer une application faisant office d'éditeur XML permettant donc à n'importe quel utilisateur qui utilisera l'interface de créer ou de modifier un document XML à sa guise, le tout en conservant le respect des normes dictées par le standard XML, incluant donc tout le processus de validation des données.

Après avoir exposé l'analyse menée pour étudier le projet même, ses spécifications et ses besoins, nous présenterons le rapport d'activité rendant compte des méthodes de travail que nous avons utilisées pour mener à bien ce projet. Le rapport technique décrit les choix de conception ainsi que des extraits plus pratiques, expliquant des portions de code. Et enfin, après avoir présenté le manuel d'utilisation de l'application, nous concluons en exposant des perspectives d'amélioration du logiciel.

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 définir la structure et le type de contenu du document, en plus de permettre de vérifier 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 outils propres 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.

2.2 Analyse de l'existant

Plusieurs solutions sont déjà proposées, certaines étant payantes et d'autres sont gratuites et open source. L'objectif est ici de fournir une solution similaire aux autres logiciels.

En se basant sur le logiciel le plus pertinent d'après les recherches autour du sujet "xml editor", le logiciel oXygen XML Editor semble être le plus présent et utilisé. C'est une solution logicielle contenant deux principaux outils : XML Developer et XML Author. Le tarif pour le package complet de XML Editor est au prix de 488\$ ou 19\$ par mois, là où chaque outil coûte à l'unité 349\$, ce qui en fait un logiciel très onéreux. L'entreprise propose cependant une version d'essai de 30 jours pour tester le logiciel. Le programme possède un nombre très important de fonctionnalités dans le but d'être utilisable par un développeur qui connaît déjà le langage et qui voudrait optimiser la saisie de fichiers XML mais aussi par un novice de XML avec un système d'assistants de création de documents.

La figure 2.1 expose par exemple une vue spéciale du document sous forme de tableur, permettant une modification beaucoup plus aisée des attributs des nœuds.

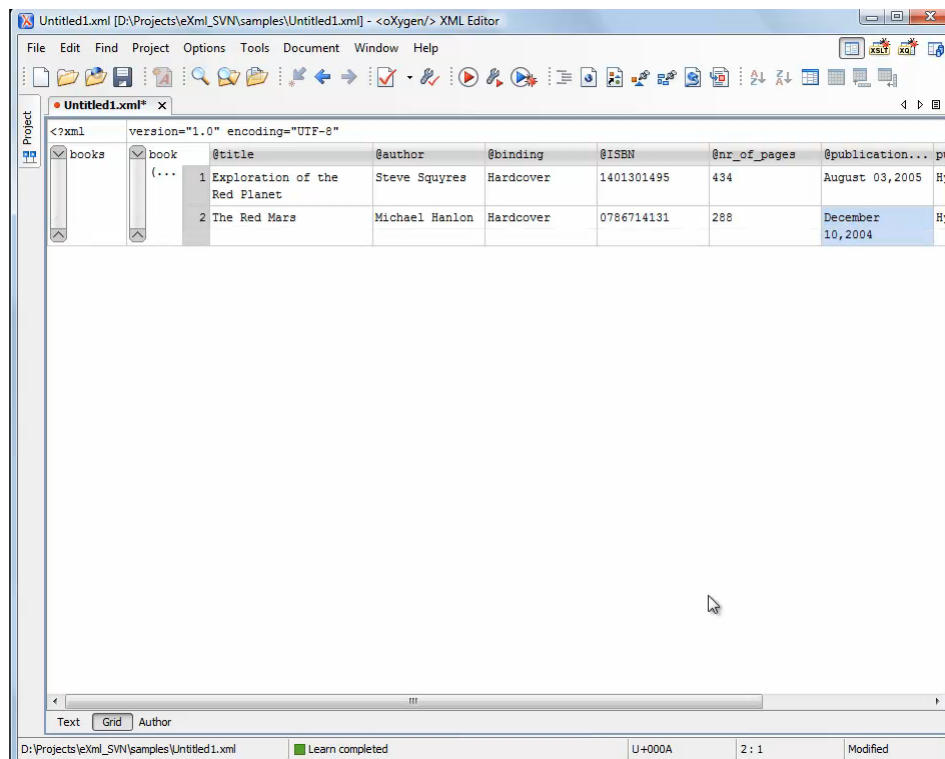


FIGURE 2.1 – Exemple de vue avec Oxygen

2.3 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. La figure 2.2 présente les différents cas d'utilisation de l'application pour ce qui est de la gestion de fichiers brute.

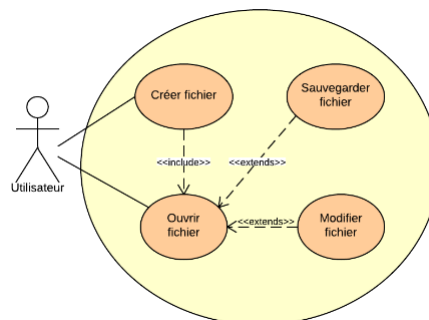


FIGURE 2.2 – Diagramme des cas d'utilisation côté gestion des fichiers

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. Les différentes actions réalisables via la vue arborescente sont exposées dans la figure 2.3. Les actions liées à l'éditeur de texte même et à la vérification syntaxique sont elles décrites sur la figure 2.4.

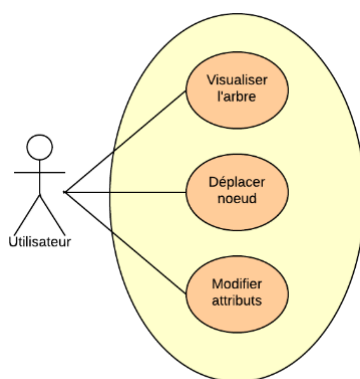


FIGURE 2.3 – Diagramme des cas d'utilisation de la vue arborescente

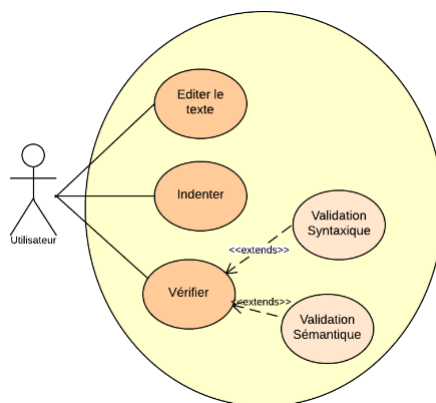


FIGURE 2.4 – Diagramme des cas d'utilisation de l'éditeur de texte

La figure 2.5 expose la maquette d'interface de l'éditeur avec chacune des parties annoncées : à gauche la vue arborescente affichant chaque noeud de l'arbre formé par les données, à droite l'éditeur de texte avec toutes ses fonctionnalités associées et en haut la barre d'outils avec la gestion de fichiers (nouveau fichier, ouvrir, sauvegarder) ainsi que la gestion du schéma.



FIGURE 2.5 – Maquette d’interface

2.4 Analyse des besoins non fonctionnels

2.4.1 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 délais acceptables et de manière stable, dans la mesure où la taille et la complexité des données restent raisonnables.

2.4.2 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 d'activité

3.1 Organisation du travail

3.1.1 Communication

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 quelle bibliothèque d'affichage graphique sélectionner pour le projet et ainsi être conscient des avantages et des limites des éléments choisis.

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 quelle 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 articulée autour 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.

3.1.2 Répartition des tâches

Nous nous sommes répartis les tâches de la façon suivante : l'arborescence pour Stéphane, l'éditeur de texte pour Noé, Benoit pour aider dans la réalisation de ces deux parties et Ulrich sur la validation du document XML à l'aide d'une DTD. L'attribution de chaque tâche a été faite en fonction de l'avancement dans l'apprentissage de Qt et des préférences

de chacun, car rappelons-le, aucun des membres du groupe n'avait d'expérience avec Qt. La réflexion sur l'architecture du programme, comment les différents modules devaient communiquer ou comment représenter les données s'est en revanche faite entre les deux développeurs des modules principaux du projet.

Étant un groupe assez restreint, il a donc été facile pour chacun de trouver sa place et de fournir le travail qu'il jugeait juste et suffisant.

3.1.3 Apprentissage d'une nouvelle API

Il nous a fallu dans un premier temps découvrir le fonctionnement intrinsèque de Qt. Pour certains ce fût même la découverte de la programmation d'interface graphique basée sur des widgets. Qt étant utilisé par un grand nombre de développeurs, un nombre égal de tutoriels et astuces ont vu le jour, en plus d'une documentation très complète fournie par les développeurs de Qt de tous leurs outils. Se mettre en route a donc été assez rapide selon le degré de motivation de chacun.

Mais Qt est énorme, et intègre un grand nombre d'outils et de classes. Nous avons à plusieurs reprises produit un bout de code pour nous rendre compte par la suite que cette fonctionnalité était déjà implémentée par une classe de Qt. Une grande partie du travail a donc été de trouver et apprendre à utiliser les classes proposées par Qt.

Un refactoring continu a été effectué tout au long du développement à mesure que nous nous améliorions en C++ et que nous connaissions Qt.

3.2 Outils de développement

3.2.1 Langage et bibliothèques

Nous avons décidé d'utiliser C++ comme langage pour ce projet, nous avons également l'option d'utiliser Java, mais C++ étant un nouveau langage pour nous, il a été retenu pour parfaire nos connaissances. Il a ensuite été décidé d'utiliser Qt comme bibliothèque d'interface graphique. Nous avons à nouveau un choix à faire, mais Qt a été préféré par rapport à GTK, l'autre grande bibliothèque d'affichage de fenêtres, il est en effet plus documenté et utilisé, et n'embarque pas uniquement des utilitaires d'affichages, mais également un grand nombre d'outils et classes qui nous faciliterons grandement la tâche.

Qt a également l'avantage d'être multi-plateforme, le code produit est en théorie compilable sur les systèmes d'exploitation majeurs. C'est alors le gestionnaire de fenêtre de la plateforme cible qui sera utilisé comme on peut le voir figures 3.1, 3.2 et 3.3, on a donc des applications qui se fondent dans l'environnement natif et respectent l'esthétique du système sans travail supplémentaire de la part du développeur.

Qt nous fournit un grand nombre de modules qui vont de la communication réseau à la gestion d'OpenGL, mais c'est surtout pour sa gestion complète et native du XML que nous l'avons préféré à GTK.

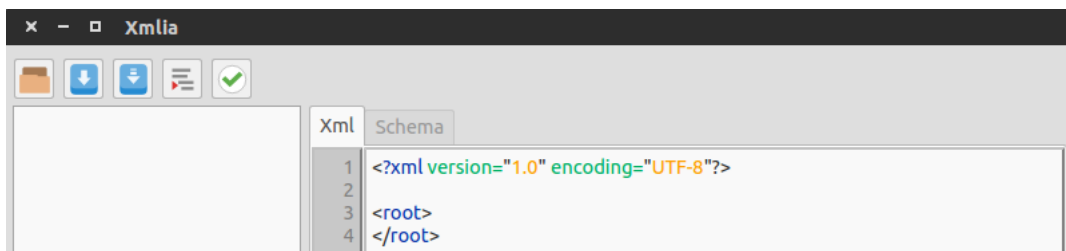


FIGURE 3.1 – Apparence sous Linux

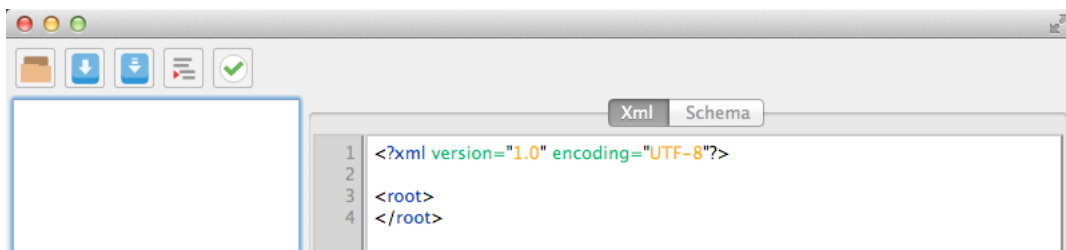


FIGURE 3.2 – Apparence sous Mac Os



FIGURE 3.3 – Apparence sous Windows

3.2.2 IDE

Nous avons naturellement utilisé Qt Creator comme IDE, c'est en effet l'IDE fourni avec Qt, il intègre donc nativement toutes les opérations spécifiques à la création et la compilation d'un projet sous Qt. Il propose également toutes les fonctionnalités que l'on pourrait attendre d'un IDE digne de ce nom, telles que la coloration syntaxique, l'autocomplétion ou encore un débogueur.

Il est tout à fait possible de réaliser un projet entier avec Qt sans passer par Qt Creator, mais l'intégration de la documentation et des fichiers d'en-tête des différentes classes de Qt constitue un gain de temps énorme et accélère grandement le processus de développement d'une part, mais également le processus d'apprentissage.

Qt propose en plus un éditeur d'interface graphique nommé Qt Designer qui permet de créer des interfaces très simplement avec du glisser-déposer notamment pour les intégrer par la suite dans une application Qt. Nous avons cependant fait le choix de ne pas l'utiliser, notre interface est en effet suffisamment simple pour ne pas avoir recours à un outil supplémentaire.

3.2.3 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.

Git a ici permis de gérer les accès et les mises à jour des différents fichiers du projet, qu'ils soient du code ou du texte brut, comme par exemple pour le rapport du projet. Son utilité aura donc été de permettre une gestion des fichiers de manière formelle, avec des gestions de conflits de versions de fichier, par exemple avec un système de gestion de fichiers basique comme un FTP, si deux développeurs travaillent en accès concurrentiel sur le même fichier, chacun aura alors sa version du fichier et au moment de renvoyer le travail effectué sur le serveur FTP, un conflit de version surviendra, c'est pour cela que Git est utile en mettant en place des barrières empêchant ce genre de problèmes et proposant des solutions pour, par exemple, réunir les deux fichiers et qu'un développeur s'occupe de "merge" ces deux fichiers en un seul qui contiendra alors le code des deux développeurs. Un dernier point important à aborder est le fait qu'un envoi de code sur le serveur peut être annulé si par exemple une erreur d'utilisation de Git a été faite et que des fichiers auraient alors été modifiés rendant le projet non fonctionnel.

Bitbucket est un service accessible depuis une page Web permettant la gestion de projets Git. Le service propose donc un serveur Git fonctionnel avec une interface Web très performante. Cette interface permet de gérer tous les projets auxquels on est rattaché, créer de nouveaux projets et gérer les droits liés à ces projets. Un nouveau projet a donc été créé sur le site au travers de l'interface puis les droits de modification du projet ont été données aux autres membres du groupe du projet qui ont alors pu commencer à utiliser le Git du projet. Ajouté à cela, Bitbucket liste l'intégralité des différentes opérations effectuées sur le Git permettant un regard global et rapide sur toutes les réalisations et offre aux différents membres du groupe la possibilité d'écrire des commentaires sur chaque opération, les membres seront donc notifiés par e-mail de ce changement, ce qui aura été un moyen de communication très présent au cours de la phase de développement. De plus, une fonctionnalité très importante de Bitbucket pour la gestion de projets est le système de tickets qui est intégré à l'interface Web où chaque membre peut ajouter soit une tâche à effectuer ou un bug à corriger et l'affecter à un autre membre, cela permet alors d'avoir une communication plus claire et concise, donnant un regard des autres membres sur les tâches effectuées par le groupe, aussi en leur donnant la possibilité de commenter ces tickets et d'ajouter des informations nécessaires à la réalisation de la tâche, donnant alors un point de vue global sur l'avancement du projet.

La figure 3.4 montre par exemple un ticket concernant une fonction à coder avec une description précise et des commentaires afin de mettre l'accent sur le sens de la tâche à réaliser, afin d'avoir une compréhension plus claire du problème.

Issue #7 **RESOLVED**

Parcours d'arbre et reconstruction avec modification d'un noeud



Doelia created an issue 2014-04-10
Implémenter la méthode suivante :

```
void updateNodeName(QDomNode dom, QString newName);
```

Il faut reconstruire le modèle QDomDocument en le remplaçant, par le même à l'exception d'un noeud qui doit changer de nom (infos en parametres).

Reconstruction complète obligatoire car on ne peut pas utiliser des pointeurs. Pour reconnaître à quel moment on atteint le noeud à modifier, on peut utiliser le nombre de parent parcourus.

Comments (4)



Doelia

- marked as *critical*

2014-04-10



Doelia

Le nom de la méthode a changé en void updateNodeName(QDomNode dom, QString newName);

2014-04-10

FIGURE 3.4 – Exemple de ticket sur Bitbucket.

Chapitre 4

Rapport technique

4.1 Conception

4.1.1 L'architecture de Qt

Qt impose un très grand nombre de contraintes par rapport aux classes à utiliser et à leur connexion. Il n'est pas possible de décider de l'organisation des classes de l'application sans connaître celle de Qt. L'étape de conception de l'application a dû d'être réalisée après l'apprentissage de la librairie. Qt est fortement basé sur le patron MVC *Modèle-vue-contrôleur*, toute l'architecture du programme en dépend.

La librairie Qt se décompose en plusieurs parties :

- *Les modèles*, contenant les données de l'application ;
- *Les vues*, qui peuvent être générées automatiquement à partir d'un même modèle ;
- *Un système d'écoute d'évènements* au travers des signaux/slots propres à Qt.

Le principe est de générer un unique modèle qui sera affiché par les différentes vues. Les évènements déclenchés sur les différentes vues par l'utilisateur sont traités pour modifier le modèle, puis la vue est rafraîchie. La partie *contrôleur* est intégrée aux classes de la vue car le système de slots/signaux que propose Qt est propre et peu encombrant.

4.1.2 Modèle de classe

Toutes les classes héritent de celles de Qt. La librairie impose que toutes les vues doivent hériter du type de vue de Qt voulu (vue arborescente, éditeur de texte, bouton...), qui héritent elles-mêmes de la classe QWidget. Le programme se compose ainsi des classes principales suivantes :

- *MainWindow*, contenant des pointeurs vers toutes les vues ainsi que la barre de menu et celle des boutons ;
- *ModeleXml*, contenant les données de l'arbre XML à jour et toutes les méthodes nécessaire à sa modification ;
- *Notepad*, la vue des éditeurs de texte (XML et Schéma) ;
- *Arbo*, la vue de l'arborescence ;
- *Logger*, la vue du logger ;
- Et un grand nombre de sous-classes conçues au fil de la phase de développement dans un soucis de clarté du code.

L'organisation de l'architecture se résume dans le diagramme de classe 4.1. Seules les composantes principales du programme apparaissent.

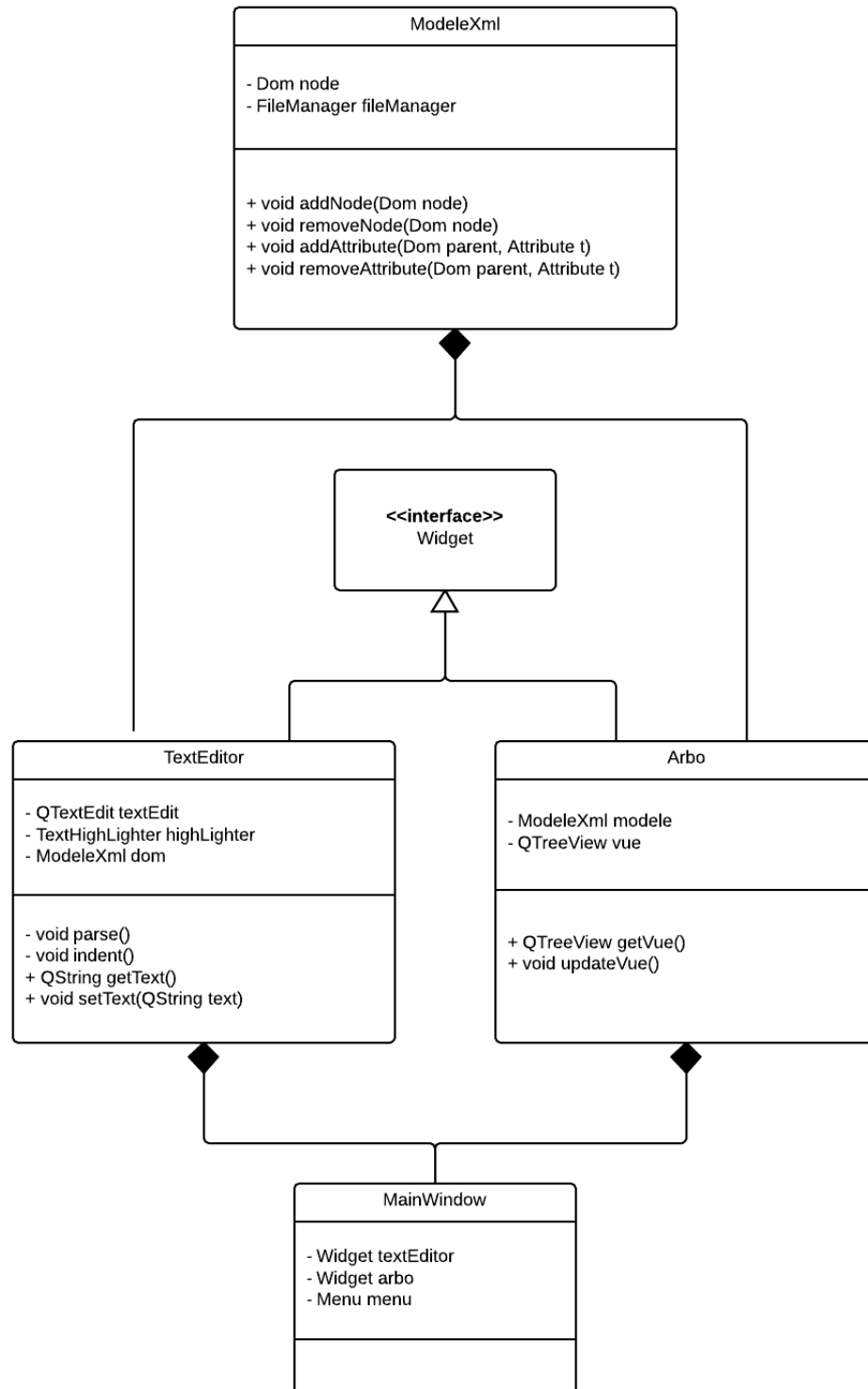


FIGURE 4.1 – Diagramme de classe

4.2 Architecture de l'application

4.2.1 Le modèle

Structure XML

Le programme est composé d'un et d'un seul modèle représentant le fichier XML arborescent. Il contient les informations sur tous les noeud de la structure XML (Nom, valeur, attributs). L'accès à l'arborescence XML se fait à partir d'un unique attribut noeud parent de type *QDomDocument*. Ce type était très contraignant dans la programmation car il n'utilisait pas des pointeurs mais des références, y compris dans tous ses fils, au contraire de tous les autres types de données utilisés ailleurs dans le programme.

Au lancement du programme, un dom vide est généré. Il est redéfini à l'ouverture d'un fichier de façon automatique grâce à une simple méthode de Qt de la façon suivante.

```
QFile file(currentFile);

if (file.open(QIODevice::ReadOnly))
{
    QString data(file.readAll());

    if (doc->setContent(data, &error, &errorLine, &errorColumn))
    {
        // Definition du dom a partir du texte du fichier
        this->modele->setFromDocument(doc);
    }

    notepad->setText(data);
}
```

Modification

Le modèle implémente toutes les méthodes nécessaires à la modification du dom. On les retrouve clairement dans le fichier entête de la classe.

```
/**
 * @action Modifie le nom du noeud passe en parametre,
 * et envoi le signal onNodeNameUpdate
 */
void updateNodeName(QDomNode n, QString newName);

/**
 * @action Insere un noeud dans le parent indique,
 * et envoi le signal onNodeInsert
 */
void insertNode(QDomNode parent, QDomNode node);

/**
 * @action Supprime le noeud et sa sous arborescente
 * et envoi le signal onNodeRemove
 */
void removeNode(QDomNode dom);
```

Toutes ces méthodes publiques sont appelées par les différentes vues. Elles modifient directement la structure XML du modèle, puis notifient les changements par un signal. Cet ensemble est minimal mais suffisant et permet de modifier à souhait l'arbre. Par exemple

une procédure de déplacement d'un noeud pourra être scriptée par une insertion/suppression.

Aspect algorithmique

Toute la subtilité technique de la classe du modèle repose sur la modification propre et optimisée de l'arborescence. Différents algorithmes de parcours d'arbres ont été utilisés pour y parvenir. Voici par exemple une méthode utilitaire permettant de récupérer la pile représentant le parcours à effectuer en largeur à partir du noeud voulu.

```
stack<int> ModeleXml::pathFromRoot(QDomNode n)
{
    stack<int> s;
    while(!n.parentNode().isNull())
    {
        s.push(rowFromNode(n)); // On recupere la position du noeud en
                                largeur
        n = n.parentNode();
    }

    return s;
}
```

4.2.2 La vue arborescence

La classe de la vue arborescente encapsule la classe QTreeView de Qt permettant de générer une interface arborescence interactive avec toutes les options nécessaires (Renommage, suppression, glisser/déposer...).

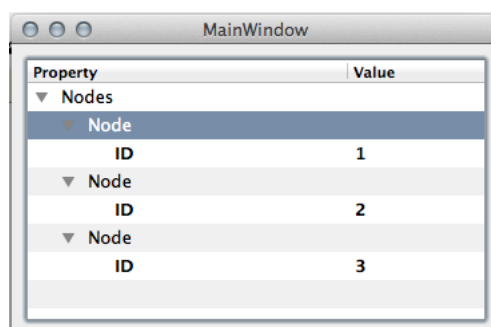


FIGURE 4.2 – Vue arborescente de Qt

Un modèle non compatible

La difficulté technique dans cette partie du programme est en rapport avec le modèle nécessaire à la construction de la vue : Il n'est pas compatible avec le modèle que nous avons conçu. Si le modèle utilise des objets de type QDomNode, QTreeView utilise des objets de type QStandardItem. Une grande partie de la programmation de la vue arborescente a donc été de réaliser des conversions pour communiquer avec le modèle.

La classe est donc composée de tout un tas de méthodes utilitaires permettant ces conversions. Quelques exemples de signatures :

— QStandardItem* getItemFromNode(QDomNode dom);

- `QDomNode getNodeFromItem(QStandardItem* item);`
- `QStandardItem* getFils(QDomNode dom);`
- `void preOrder(QDomNode dom, QStandardItemModel *model);` // Construit une arborescence de `QStandardItem` à partir d'une arborescence `QDomNode`.

La méthode de conversion reste cependant toujours la même, on passe d'un type à un autre en parcourant les structures en se basant sur une pile de parcours en largeur en utilisant les numéros des sommets, qui sont les même quelque soit le type d'arborescence.

La gestion des évènements

La vue arborescence est modifiée par les évènements quand l'éditeur de texte effectue des modifications. Pour des raisons pratiques, la vue arborescente est régénérée uniquement lors de la validation syntaxique et non tout au long de l'édition textuelle. Il n'est tout simplement pas possible de générer une arborescence d'un dom non valide. La gestion des évènements entrant et donc très simple, il suffit d'appeler la méthode de reconstruction de la vue à partir du modèle.

Pour ce qui est des évènements sortants (Les évènements déclenchés par l'utilisateur lorsqu'il modifie l'arbre), la programmation est plus complexe. La liste des possibilités de l'utilisateur est la suivante :

- Renommage d'un noeud
- Suppression d'un noeud
- Glisser/déposer d'un noeud

Le renommage et la suppression est triviale. Le renommage étant effectué instantanément sur la vue, il suffit d'envoyer le signal de modification au modèle. La suppression est légèrement plus compliqué car l'option n'est pas implémentée directement par Qt. Un simple menu contextuel a été ajouté, puis les méthodes utilitaires permettent d'effectuer simplement l'action.

```
// Lorsque l'utilisateur supprime un noeud (clic droit)
void Arbo::onRemoveNode() {
    QStandardItem* item = this->itemRoot->model()->itemFromIndex(this->
        getVue()->selectionModel()->currentIndex());
    item->parent()->removeRow(item->row());
}
```

Ce qui déclenche automatiquement un évènement par Qt :

```
void Arbo::onRowsAboutToBeRemoved(const QModelIndex &i , int x, int y)
{
    QStandardItem* item = this->itemRoot->model()->itemFromIndex(i);
    QDomNode node = this->getNodeFromItem(item);
    node = node.childNodes().at(x);
    XmlFileManager::getFileManager()->getModele()->aboutToBeRemoved(node
        );
}
```

Le glisser déposer nécessite davantage de programmation. Aucun évènement "Glisser/-déposer" n'est envoyé par Qt, mais deux évènements appelés indépendamment : Ajout d'un noeud, puis suppression d'un noeud. C'est source de beaucoup de problèmes et de conflits.

De plus, l'évènement d'ajout de noeud déclenche lors du déposer est ambiguë. Il est appelé sous la forme d'un évènement d'édition avec comme parametre unique le parent dans lequel le noeud à été déposé : Le noeud inséré n'est pas passé en paramètre. Il est donc nécessaire de parcourir l'arbre afin de trouver le noeud présent en double (l'ancien n'a pas encore été supprimé à cet instant). Une fois trouvé, un signal d'ajout de noeud est finalement envoyé au modèle pour répercuter les modifications à l'éditeur de texte. Par contre, l'évènement de suppression est le même que celui utilisé plus haut pour la suppression manuelle d'un noeud.

4.2.3 L'éditeur de texte

L'éditeur de texte est décomposé en deux parties, l'éditeur de schéma et l'éditeur XML. Ce sont en réalité des spécialisations de la classe `TextEditor`, mais parlons d'abord de l'éditeur de texte en général.

La classe `TextEditor`

C'est ici que sont toutes les méthodes servant à l'édition du texte en général, telles que l'insertion de texte, l'indentation ou la coloration. L'éditeur de texte est une sous-classe de `QTextEdit`, qui fournit toutes les fonctionnalités basiques d'un éditeur de texte riche, telles que la sélection de texte, la copie, l'annulation etc. On peut intégrer plusieurs fonctionnalités à un `QTextEdit` comme par exemple la coloration syntaxique. Nous parlerons dans un premier temps des fonctionnalités propres à notre éditeur de texte, puis nous verrons plus en détail chacune de ses implémentations, à savoir l'éditeur XML et l'éditeur de schéma.

Qt propose un `QCompleter`, il ne peut cependant pas s'intégrer et proposer l'auto-completion dans un `QTextEdit`. Même s'il ne s'intègre pas dans un `QTextEdit`, le `QCompleter` peut tout de même proposer une suggestion si on lui fournit un début de mot. On peut donc récupérer le mot sous le curseur, le donner au `QCompleter` et insérer le mot complété dans notre éditeur de texte. Cela se fait de la manière suivante :

```
//on donne le mot sous le curseur au QCompleter
completer->setCompletionPrefix(textUnderCursor());
//on recupere le mot propose
QString completion = completer->currentCompletion();
QString currentWord = textUnderCursor();

//si un mot est sous le curseur et qu'il n'est pas deja complete
if(currentWord.size() > 0 && currentWord.size() < completion.size())
{
    QTextCursor cursor = text->textCursor();
    //sauvegarde de la position actuelle du curseur
    int pos = cursor.position();
    //deplacement jusqu'a la fin du mot
    while(word.indexOf(cursor.selectedText()) == 0)
    {
        cursor.movePosition(QTextCursor::Left, QTextCursor::KeepAnchor);
    }
    //insertion de la partie qui n'est pas deja ecrite
    cursor.insertText(completion.right(completion.size() - currentWord.
        size()));
    //le curseur est replace a sa position originelle
    cursor.setPosition(pos, QTextCursor::KeepAnchor);
    text->setTextCursor(cursor);
}
```

Il n'est pas possible d'intégrer directement l'auto-complétion, mais on peut en revanche facilement lier un colorateur syntaxique à un QTextEdit. Qt propose une classe abstraite QSyntaxHighLighter qu'il faut implémenter en y indiquant nos règles de coloration. Le QTextEdit appelle automatiquement la méthode *highlightBlock* du QSyntaxHighLighter dans laquelle on aura indiqué nos règles.

```
void TextHighLighter::highlightBlock(const QString &text)
{
    setCurrentBlockState(previousBlockState());
    //l'ordre est important
    //il ne faut pas appliquer de coloration a l'interieur
    //il faut donc colorer les commentaires en premier
    for (int i = 0; i < text.length(); ++i)
    {
        //colore les commentaires
        if(cComment(last, text, i));
        //colore le texte entre quotes
        else if(cQuote(last, text, i));
        //colore les attributs
        else if(cInMarkupAttr(last, text, i));
        //colore les balises
        else if(cMarkup(last, text, i));
    }
}
```

Voici par exemple la méthode qui permet de colorer les commentaires :

```
bool TextHighLighter::cComment(int &last, const QString &text, int i)
{
    //si on se trouve deja dans un commentaire
    if(currentBlockState() == COMMENT_STATE)
    {
        //si on trouve la fin du commentaire
        if (text.mid(i, 3) == "-->")
        {
            //on colore entre last et la fin du commentaire
            setTextColor(last, i + 4, Qt::gray);
            setCurrentBlockState(DEFAULT_STATE);
        }
        setTextColor(last, i + 1, Qt::gray);
        return true;
    }
    //si on trouve le debut d'un commentaire
    else if (text.mid(i, 4) == "<!--")
    {
        //on sauvegarde la position du debut de commentaire
        last = i;
        //on marque que l'on est dans un commentaire
        setCurrentBlockState(COMMENT_STATE);
        return true;
    }
    return false;
}
```

La classe XmlEditor

C'est donc une sous-classe de TextEditor. 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 QDomStreamReader qui s'utilise de la manière suivante :

```
//creation d'un QDomStreamReader affecte du texte du QTextEdit
QDomStreamReader xml(text->toPlainText());
while(!xml.atEnd())
{
    //detection d'une balise ouvrante
    if(xml.isStartElement())
    {
        //traitement
    }
    //detection d'une balise fermante
    else if(xml.isEndElement())
    {
        //traitement
    }
}
```

On utilise une structure de pile lors d'un 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 dernière balise rencontrée
    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 utilisée de la manière suivante :

```

//methode appelee lorsque l'utilisateur renomme un noeud dans l'
//arborescence
//on passe en parametre un pointeur de fonction
xmlEditor->parseDom(n, n.nodeName(), QString(newName), &XmlEditor::
updateNodeName);

```

Voici l'implémentation de l'appel de fonction dans le pseudo code du parcours de document :

```

//si le chemin du node courant est le meme que celui
//du sommet passe en parametre
if(cmpVectors(path, nodePath))
{
    //on appelle la fonction passee en parametre
    //elle traitera le node avec des effets de bord
    if((this->*function)(nbFound, begin, end, c, oldName, newName, xml))
    {
        //on arrete le parcours de l'arbre si la fonction retourne true
        return;
    }
}
}

```

Voici le prototype de la fonction *parseDom* avec la fonction qu'elle prend en paramètre et les fonctions de manipulation de noeud qu'elle peut prendre en paramètre.

```

//parse le dom jusqu'a trouver le node target
void parseDom(QDomNode &target, QString oldName, QString newName, bool
(XmlEditor::*function)
(int &nbFound, int &begin, int &end, QTextCursor &c, QString oldname,
QString newname, QDomStreamReader &xml));

//remplace le nom de la balise oldName par newName
bool updateNodeName(int &nbFound, int &begin, int &end, QTextCursor &c
, QString oldName, QString newName, QDomStreamReader &xml);
//supprime le noeud
bool deleteNode(int &nbFound, int &begin, int &end, QTextCursor &c,
QString oldName, QString newName, QDomStreamReader &xml);
//sauvegarde les donnees du noeud dans le cas d'un deplacement
bool saveNodeData(int &nbFound, int &begin, int &end, QTextCursor &c,
QString oldName, QString newName, QDomStreamReader &xml);
//insere un noeud
bool insertNodeText(int &nbFound, int &begin, int &end, QTextCursor &c
, QString oldName, QString newName, QDomStreamReader &xml);

```

Ces fonctions manipulent le texte à travers un QTextCursor qui permet de naviguer dans un QTextEdit. Un QTextCursor permet notamment de se déplacer de mot en mot, d'aller à la fin de la ligne ou d'aller à la ligne suivante. Un outil donc indispensable dans la manipulation du texte.

La liaison vers un schéma se fait à l'aide d'expressions régulières. Le XML étant standardisé, les données auront toujours la même forme, une expression régulière semble donc le plus simple pour trouver une information dans du texte.

```
void XmlEditor::removeSchema()
{
    QString s(text->toPlainText());
    //regex du lien vers un schema
    QRegExp r("\n*xmlns:xsi.*=\".*\.xsd\"\\n*");
    suppression de la regex dans le texte
    s.remove(r);
    text->setText(s);
}
```

Il est intéressant de noter que l'on utilise une fois de plus la classe QDomStreamReader pour se déplacer dans l'arborescence XML. Le lien vers le schéma se trouve dans la balise racine, le plus simple pour la trouver est donc de parcourir le document jusqu'à la trouver.

```
while(!xml.atEnd())
{
    //lorsque l'on trouve la premiere balise ouvrante
    if(xml.isStartElement())
    {
        //on deplace le curseur jusqu'a la position de la fin de cette
        //balise
        moveCursorToLineAndColumn(c, xml.lineNumber() - 1, xml.
            columnNumber() - 1, false);
        //construction du lien vers le schema
        QString link("\nxmlns:xsi=\"http://www.w3.org/2001/XMLSchema-
            instance\"\\n xsi:noNamespaceSchemaLocation=\"");
        link.append(XmlFileManager::getFileManager()->getSchemaName()).
            append("\\");
        //insertion du lien dans le document
        c.insertText(link);
        text->setTextCursor(c);
        return;
    }
    xml.readNext();
}
```

En plus de créer et supprimer un lien vers un schéma, il est également possible de l'extraire, pour la validation par exemple. Cela se fait une fois de plus grâce à une expression régulière. Il est intéressant de noter dans l'exemple suivant que nous ne gérons que les liens locaux vers un schéma et pas les liens vers un schéma distant.

```
QString s(text->toPlainText());
//split du document pour se placer directement a l'endroit du lien
QStringList l(s.split("xsi:noNamespaceSchemaLocation=\""));
//si un lien est present dans le document
if(l.size() > 1)
{
    //split selon le caractere " pour ne garder que le lien
```

```

s = l.at(1).split("\\").at(0);
//on retourne le chemin courant du fichier XML auquel on concatene
le lien que l'on vient de trouver
return XmlFileManager::getFileManager()->getCurrentFilePath().append
    ("/").append(s);
}
//on cherche s'il n'y a pas de lien distant
l = s.split("xsi:SchemaLocation=");
if(l.size() > 1)
{
    //on notifie qu'on ne le gere pas encore s'il existe
    emit log("Cannot process http requests yet, XML will not be checked"
        , QColor("orange"));
}
return "";

```

La classe SchemaEditor

La classe SchemaEditor ne rajoute pas vraiment de fonctionnalités à la classe TextEditor mis à part la génération de schéma à partir d'un document XML. Elle fonctionne de manière très simple, les balises sont extraites du document XML et sont ajoutées au schéma. L'extraction des balises se fait à nouveau grâce à la classe QDomStreamReader, on parcourt le document et ajoute chaque balise dans une liste si elle n'est pas déjà présente.

```

while(!xml.atEnd())
{
    //on utilise une hashmap pour simplifier la non duplication
    h.insert(xml.name().toString(), 0);
    xml.readNext();
}
//on appelle la methode de generation de schema
schemaEditor->genSchema(h.keys());

```

Voici l'implémentation de la méthode de génération de schéma, on peut remarquer qu'elle est assez basique et laisse à l'utilisateur le soin de compléter la valeur et le type de chaque élément.

```

void DtdEditor::genSchema(QList<QString> l)
{
    //ajout des meta data
    text->setText("<?xml version=\"1.0\" encoding=\"UTF-8\" ?>");
    text->append("<xs:schema xmlns:xs=\"http://www.w3.org/2001/XMLSchema
        \">>");

    //ajout de chaque element
    for(QString s : l)
    {
        QString toAppend("<xs:element name=\"");
        toAppend.append(s).append("\>");
        text->append(toAppend);
    }
    text->append("</xs:schema>");
    indent();
}

```

La validation

Parlons maintenant de la validation syntaxique et sémantique. La validation syntaxique se fait très simplement à l'aide du `QXmlStreamReader`, qui permet de détecter une erreur et d'indiquer la ligne, ainsi que le type de l'erreur.

```
while(!xml.atEnd())
{
    xml.readNext();
    //si le parseur XML rencontre une erreur
    if(xml.hasError())
    {
        //affichage de la ligne de l'erreur avec son type
        emit log("Erreur ligne " + QString::number(xml.lineNumber()) + " :
                " + xml.errorString(), QColor("red"));
    }
}
```

Pour la validation sémantique, qui ne peut se faire que si un schéma est rattaché au document XML, on doit passer par une autre classe de Qt, le `QXmlSchemaValidator` et sa méthode `validate`. Nous l'utilisons de la manière suivante :

```
QXmlSchema schema;
//extraction de l'url du schema dans le document XML
QString url(extractSchemaUrl());

//si le xml contient une url valide vers un schema
if(url.size() > 1)
{
    schema.load(QString("file://").append(url));
    //on verifie au passage la validite du schema en lui meme
    if (schema.isValid())
    {
        emit log("Schema XSD valide", QColor("green"));
        QXmlSchemaValidator validator(schema);
        validator.setMessageHandler(mh);
        if (validator.validate(this->getText().toUtf8(), QUrl(
            XmlFileManager::getFileManager()->getCurrentSchema()))
        {
            //notification que la semantique est valide
            emit log("Semantique XML valide", QColor("green"));
        }
    }
    return true;
}
//si le schema n'a pas pu etre trouve
emit log("Schema XSD invalide, est-il manquant ou invalide ?", QColor(
    "orange"));
```

4.2.4 Le logger

Le logger est la partie qui reçoit tous les messages pour les afficher à l'utilisateur. Il repose fortement sur le principe de signaux et de slots de Qt. Le fonctionnement est le

suivant : Qt propose de connecter un objet émetteur, qui enverra des messages avec la macro *emit* sur le slot d'un objet receveur, qui est en réalité une fonction qui sera appelée au moment du *emit*. Voici par exemple le fonctionnement de l'envoi de messages à afficher :

Tout d'abord l'interface de notre Logger, pour montrer que les slots ne sont rien d'autre que des fonctions.

```
class Logger : public QWidget
{
    Q_OBJECT
public:
    explicit Logger();

    signals:
        //les signaux, il n'y en a aucun dans notre Logger

    public slots:
        //notre slot log qui affichera dans la zone de texte la QString de
        //la couleur QColor
        void log(QString s, QColor c);

    private slots:
        //slots prives, on ne pourra connecter que des signaux de la classe
        //sur des slots de cette meme classe

    private:
        //attributs prives
};
```

On connecte les objets entre eux grâce à la fonction *connect(sender, signal, receiver, slot)* de Qt.

```
//on connecte l'editeur de texte avec le logger sur le slot log
connect(notepad, SIGNAL(log(QString,QColor)), logger, SLOT(log(QString
    ,QColor)));
```

Et lorsque qu'il y aura un *emit* du signal *log*, la fonction correspondante, dans notre cas *log* sera appelée.

```
//l'emission du signal dans la methode de validation semantique
emit log("Semantique XML valide", QColor("green"));
//appellera automatiquement
void Logger::log(QString s, QColor c)
{
    //changement de la couleur d'affichage en la couleur desiree
    logArea->setTextColor(c);
    QTextCursor cursor = logArea->textCursor();
    //insertion du texte dans le QTextEdit qui sert de zone d'affichage
    //au logger
    cursor.insertText(s);

    logArea->setTextCursor(cursor);
    //affichage du Logger s'il etait masque
    show();
}
```

Ce système de signal/slot est énormément utilisé pour faire communiquer nos widgets les uns avec les autres, et permet d'une part de les séparer, c'est à dire que l'arborescence ne possède pas de pointeur vers l'éditeur de texte et inversement, mais également de ne pas avoir non plus de pointeur vers le contrôleur, une fois que les connexions sont faites dans le contrôleur, nos widgets peuvent communiquer de manière totalement indépendante.

4.2.5 Le gestionnaire de fichiers

Nous avons fait le choix d'avoir une seule instance du gestionnaire de fichier, et de s'en servir pour stocker le modèle et les différents documents ouverts. Le gestionnaire de fichier est donc un singleton.

Il embarque les fonctionnalités basiques d'un gestionnaire de fichier, telles que l'ouverture et la sauvegarde. C'est à travers ce gestionnaire de fichiers que l'on va gérer les deux documents ouverts, le document XML et le schéma qui lui est associé.

4.3 Résultat

Nous avons obtenus un éditeur pleinement fonctionnel apportant un lot suffisant d'outils pour permettre une édition complète, contrôlée et pratique d'un fichier XML.

4.3.1 Fonctionnalités présentes

- Ouverture / Enregistrement d'un fichier XML
- Edition du fichier
 - Coloration syntaxique
 - Auto-complétion
- Validation syntaxique
- Gestion d'un schéma
 - Liaison d'un schéma (Ouverture/sauvegarde automatique)
 - Edition du schéma
 - Génération automatique d'un schéma à partir du XML
 - Validation syntaxique avec prise en compte du schéma
- Vue arborescente avec répercussion sur la vue de l'éditeur
 - Renommage d'un noeud
 - Suppression d'un noeud
 - Glisser / Déposer d'un noeud
- Logger
- Divers raccourcis claviers

4.3.2 Captures

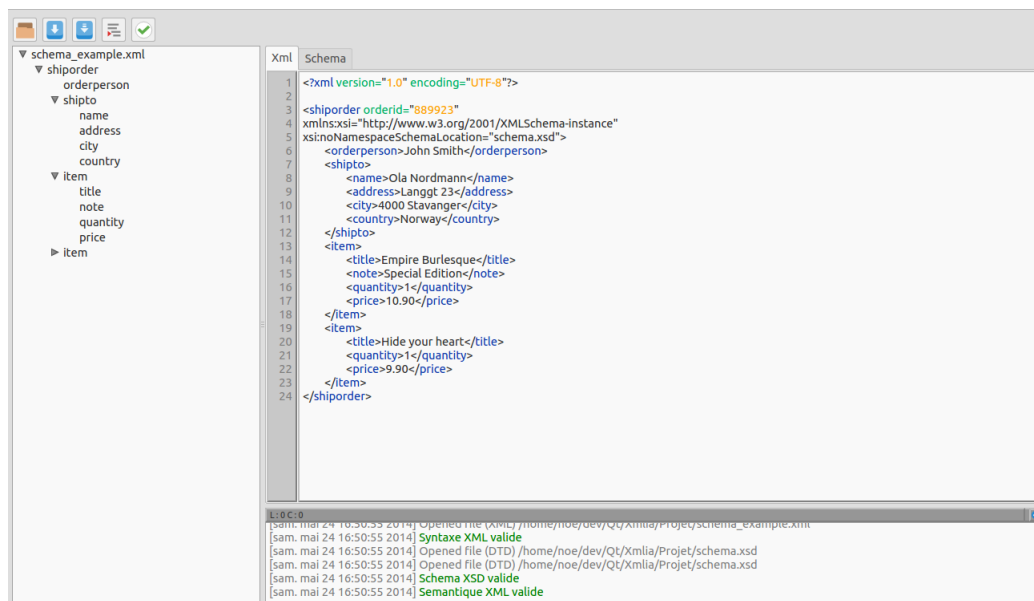


FIGURE 4.3 – Rendu final de l'éditeur

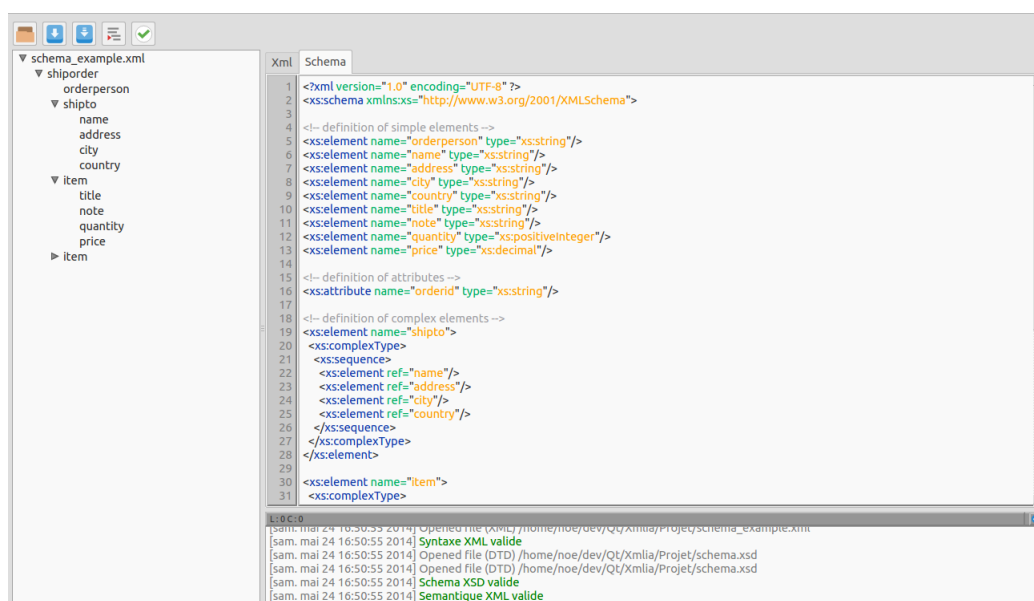


FIGURE 4.4 – Rendu final de l'éditeur

Chapitre 5

Manuel d'utilisation

L'interface de l'éditeur est assez intuitive. Les boutons sont peu nombreux et les fonctionnalités moins utilisées sont dans la barre de menu. Ce manuel liste la totalité des fonctionnalités du logiciel.

5.1 Interaction

5.1.1 Barre de menu

Les actions disponibles dans la barre de menu sont les suivantes :

- *File...* : Ouvrir / Sauvegarder / Fermer
- *Edit...* : Indenter
- *Schéma...*
 - *Load schema file* : Ouvre un schéma et le lie automatique au fichier XML.
 - *Generate schema file* : Génère automatiquement un schéma à partir du fichier XML en utilisant les balises connues.
 - *Delete schéma* : Supprime la liason entre le schéma et le fichier XML.

Les actions Ouvrir, sauvegarder, indenter et valider sont disponibles en accès rapide sur la barre des boutons.

5.1.2 Racourcis

- *ctrl + o* : Ouvrir
- *ctrl + r* : Lancer la validation
- *ctrl + s* : Enregistrer
- *ctrl + shift + s* : Enregistrer sous
- *ctrl + i* : Indenter automatiquement
- *F4* : Switch entre le fichier XML et le schéma

5.2 Validation

La validation permet de vérifier si le code XML entré est valide. Il vérifie que les balises sont correctement formés (pas de balises croisés). Et si un schéma est associé, le validateur vérifie que le fichier XML est bien en accord avec celui-ci. En cas d'erreur, le message d'erreur est indiqué dans la console et le numéro de la ligne concernée est surligné en rouge. Si la validation s'exécute correctement, la vue arborescence est générée.

5.3 Vue arborescente

Sur la vue arborescence il est possible de renommer un noeud (double clic), supprimer un noeud (clid droit / Supprimer), ou déplacer un noeud (glisser/déposer). Lors d'une modification, la vue de l'éditeur de texte est automatiquement mise à jour.

5.4 A propos du schéma

L'association d'un schéma est une option du logiciel mais il n'est pas obligatoire de lier un schéma pour travailler un fichier XML. Quand un schéma est ouvert, il est "lié" au fichier XML. La balise de lien vers le schéma est automatiquement ajoutée dans le fichier XML et un enregistrement provoque la sauvegarde des deux fichiers (xml et schéma). Lors de l'ouverture d'un fichier XML, le schéma est automatiquement ouvert si un chemin vers celui-ci est indiqué dans le XML.

5.5 Fonctions du logiciel

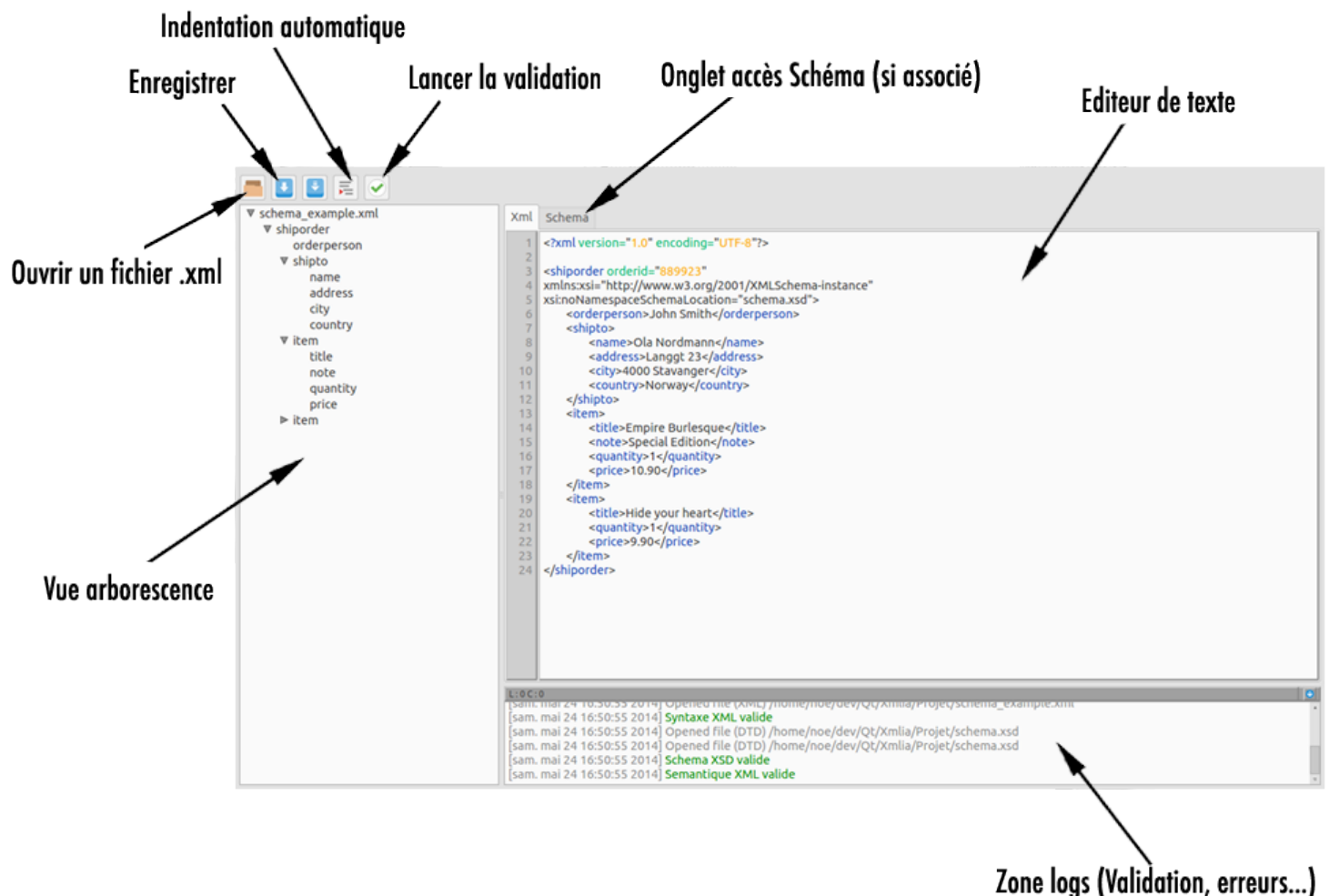


FIGURE 5.1 – Différentes sections du logiciel

Chapitre 6

Perspectives et conclusions

6.1 Perspectives

Les délais du projet et le fait que sa réalisation soit en complément des cours nous a poussé à faire l'impasse sur des fonctionnalités intéressantes, car notre but était avant tout d'arriver à un prototype fonctionnel, auquel des ajouts ont été faits. Certaines de ces fonctionnalités seront donc développées dans cette partie du rapport.

Une fonctionnalité qui d'ailleurs avait été évoquée dans le sujet en tant que fonctionnalité optionnelle est l'affichage du fichier XML dans un navigateur Web, avec l'ajout d'une feuille de style permettant de définir la disposition et l'affichage des données dans le navigateur. Cela aurait pu fournir un affichage similaire à celui d'HTML pour ce qui est des données et de CSS pour sa feuille de style.

Une autre fonctionnalité majeure aurait pu être apportée : la modification de plusieurs documents à la fois. Cela aurait permis de ne lancer qu'une seule fois l'application pour plusieurs fichiers et donc être plus organisé et consommer moins de ressources. Pour ce qui est de la méthode d'utilisation, deux manières auraient pu être possibles ; soit un affichage moderne avec des onglets pour chaque document ouvert qui aurait requis un changement du support des schémas, soit un affichage plus ancien avec un système de type MDI (Multiple Document Interface) où chaque document ouvert serait apparu comme une sous-fenêtre de l'application principale.

Dans une approche complètement différente, nous aurions pu imaginer une application beaucoup plus axée sur l'affichage graphique avec un système plus proche de l'utilisateur novice, avec des actions simples à réaliser depuis un panel léger, en s'éloignant du principe d'éditeur de texte de base.

6.2 Conclusions

6.2.1 Le projet

Le projet était en lui-même simple mais était très complet. Nous nous sommes rendu compte au fil du temps que le projet s'adaptait très bien au contexte de notre cursus universitaire. Il nous a permis d'augmenter très largement notre pratique de la programmation. (Programmation orientée objet, parcours d'arbre, validation syntaxique, expressions régulières)

6.2.2 Les outils

D'un point de vue technique, le choix du langage (C++) et de sa librairie (Qt) s'est avéré être un bon choix. Le projet se prêtait très bien à l'apprentissage et les modules proposés par Qt comblaient bien nos besoins.

Aucun de nous quatre ne connaissait cette librairie mais nous étions conscients de la difficulté à l'appréhender. C'est ainsi que plus gros du temps passé fut celui d'apprendre à utiliser Qt. D'un point de vue pédagogique nous sommes très satisfaits de ce choix, bien que les débuts furent fort fastidieux, aujourd'hui nous pouvons considérer que nous avons effectué notre première approche sur cette célèbre librairie.

Si nous devions réaliser à nouveau un projet similaire avec Qt, nous estimons que le temps de développement serait beaucoup plus court.

6.2.3 Groupe de travail

Ce projet était pour nous quatre le premier réalisé dans ces conditions. Nous ne nous connaissions tous pas et nous n'avions pas tous réalisé le même parcours ; nous avons tous des compétences différentes.

Le système de billet nous a facilité le travail de répartition des tâches en termes de communication. Toutes les tâches étaient découpées en petites parties que chacun pouvait réaliser en fonction de ses possibilités. Ceci permettait à tous les membres du groupe de participer même si tout le monde n'effectuait pas nécessairement des choix de développement.

Les premières réunions furent difficiles mais peu à peu nous avons pris conscience des compétences de chacun et nous avons pu réaliser les bons choix pour arriver au bout du projet dans de bonnes conditions.

Chapitre 7

Annexes

7.1 Code annexe

7.2 Historique des commits









































 Znoraka	08a5d86	reagencement des menus	2014-04-28
 Znoraka	824b6b8	amelioration de l'ouverture du schema	2014-04-28
 Znoraka	5c86f41	ouverture automatique du schema xml lors de l'ouverture d'un document xml avec un lien valide	2014-04-27
 Znoraka	a0b352a	ajout de l'autocompletion dans l'editeur de schema	2014-04-26
 Znoraka	0c5eaa1	correction de l'insertion de noeud dans l'editeur de texte pour les noeuds solitaires	2014-04-25
 Znoraka	bdd2b0b 	deplacement de noeud repercuté correctement dans l'editeur de texte	2014-04-25
 Beny13	d98ee93	Pseudo fix pour les bugs lors des dragndrop dans l'arbre	2014-04-25
 Znoraka	612cb72	debut d'insert de node dans l'éditeur, il faut maintenant synchroniser les signaux	2014-04-25
 Znoraka	a7ea77e 	;	2014-04-24
 Znoraka	ab672ce 	;	2014-04-24
 Doelia	c6f9e30	Permission de modification du nom des nodes	2014-04-24
 Doelia	266dcc4 	Merge branch 'master' of https://bitbucket.org/Doelia/xmlia	2014-04-24
 Doelia	713bc95	Modif arbo	2014-04-24
 Beny13	b61e798	Changement d'icône (l'ancienne n'avait pas une résolution suffisante)	 1 2014-04-24
 Doelia	ab605f9 	Merge	2014-04-24
 Doelia	ee1ec62	merge	2014-04-24
 Beny13	92f2f0c	Ajout d'une icone au programme (assets/icon.ico)	2014-04-24
 Znoraka	02281d5	;	2014-04-24
 Doelia	cee02e2	Retrait du debug	2014-04-24
 Doelia	277c838	Modification signal de suppression	2014-04-24
 Doelia	a94b47b 	Changement des signaux	2014-04-24
 Doelia	4409798	Modification modèle	2014-04-24
 Beny13	c5456b9	Indentation	2014-04-24
 Beny13	427d389 	Merge branch 'master' of https://bitbucket.org/Doelia/xmlia	2014-04-24
 Beny13	d5d3fd2	Ajout de la fonction getNonPointerRoot	2014-04-24
 Doelia	97d808d 	merge	2014-04-24
 Doelia	d46e61d	Ajout d'une méthode de comparaison de nombre total de fils	2014-04-24
 Beny13	60b3c65 	Merge branch 'master' of https://bitbucket.org/Doelia/xmlia	2014-04-24
 Beny13	91e1e5c 	Merge branch 'master' of https://bitbucket.org/Doelia/xmlia	2014-04-24

FIGURE 7.1 – Historique des commits - Page 1






























	Znoraka	8e1f24b	validation selon le schema xml operationnelle	2014-04-24
	Beny13	b8b12bb	Ajout de getNodeFromItemRecursive et equals	2014-04-24
	Doelia	82657b5	Ajout du signal d'insertion de node au modèle et du slot dans le notepad	2014-04-24
	Doelia	9e32812	Debug de la suppression de node au dnd	2014-04-24
	Doelia	7d18f98	Ajout sample 2	2014-04-24
	Znoraka	23fa518 <small>M</small>	;	2014-04-24
	Znoraka	c832ffa	passage de la dtd au schema et ajout de l'emplacement du curseur	2014-04-24
	Doelia	fbcc855	Correction d'un bug	2014-04-24
	Doelia	dd33c0c	Ajout suppression de noeud au drag n drop (ajout restant à faire)	2014-04-24
	Znoraka	95d8277 <small>M</small>	;	2014-04-23
	Znoraka	4397333	debut d'ajout d'une dtd	2014-04-23
	Doelia	6aab4ec	Ajout fonction drag n drop. Signal en construction	2014-04-23
	Znoraka	554ad04 <small>M</small>	Merge branch 'master' of https://bitbucket.org/Doelia/xmlia	2014-04-23
	Znoraka	e5b6a6a	ajout du connect entre l'éditeur xml et le notepad	2014-04-23
	Doelia	c5ebbc7 <small>M</small>	xMerge branch 'master' of https://bitbucket.org/Doelia/xmlia	2014-04-23
	Doelia	1a2f0db <small>M</small>	Merge	2014-04-23
	Znoraka	3b5941b	amelioration de la numerotation des lignes	2014-04-23
	Doelia	e61c478	Ajout d'includes dans les .h	2014-04-23
	Znoraka	76caa0f	ajout d'onglets pour l'edition de la dtd	2014-04-23
	Znoraka	8a0cec0	correction d'un bug sur la suppression de noeud dans l'editeur de texte	2014-04-22
	Znoraka	27949b1	modification de node et suppression de node refait dans l'editeur de texte	2014-04-22
	Znoraka	31e422d	deplacement du code pour enlever les noeuds indesirables pour ne pas avoir à modifier le modele	2014-04-22
	Znoraka	35af2a7	les commentaire et le texte des nodes dans l'arbo ne sont plus affichés	2014-04-22
	Znoraka	0bc540d	ajout du redimensionnement vertical	2014-04-22
	Znoraka	a400add	amelioration du logger	2014-04-20
	Znoraka	91cc8a3	modification de la fenetre de log et deplacement des methodes de lecture et ecriture	2014-04-19
	Znoraka	20b1a98	ajout d'une fenetre pour les logs	2014-04-17
	Znoraka	d26b309	ajout d'une barre de menu avec des boutons	2014-04-17
	Doelia	1dbb485	Ajout d'includes pour build depuis mon laptop	2014-04-17

FIGURE 7.2 – Historique des commits - Page 2




































 Znoraka	0fd5e09	deplacement du curseur sur l'erreur	2014-04-17
 Znoraka	6ac128c	le QTextEdit des lignes n'est plus scus scrollable	2014-04-17
 Znoraka	e55bb8a 	Merge branch 'master' of https://bitbucket.org/Doelia/xmlia	2014-04-17
 Znoraka	51e0245	ajout des numeros de ligne	2014-04-17
 Doelia	e0ba77d 	Merge branch 'master' of https://bitbucket.org/Doelia/xmlia	2014-04-17
 Doelia	a0e4d9f	Ajout d'un slot	2014-04-17
 Znoraka	432fef2	implementation de la methode pour mettre le modele à jour apres appui sur le bouton de refresh	2014-04-17
 Znoraka	d2f6a19	ajout de connect entre le bouton et uton et l'editeur	2014-04-17
 Znoraka	00fee36 	Merge branch 'master' of https://bitbucket.org/Doelia/xmlia	2014-04-17
 Znoraka	50afac3	ajout du slot pour le bouton refresh	2014-04-17
 Doelia	d1f8b6b	Ajout bouton refresh	2014-04-17
 Doelia	9319d81	Ajout du fichier de test	2014-04-17
 Doelia	0c44b6e 	Merge branch 'master' of https://bitbucket.org/Doelia/xmlia	2014-04-17
 Doelia	ff02a36	;	2014-04-17
 Doelia	8f5391a	Ajout du .pro	2014-04-17
 Znoraka	fdf54a6	;	2014-04-17
 Znoraka	78d1efd 	Merge branch 'master' of https://bitbucket.org/Doelia/xmlia	2014-04-17
 Znoraka	0044959	;	2014-04-17
 Doelia	44d7723	Ajout du readme	2014-04-17
 Znoraka	95ea733 	;	2014-04-17
 Znoraka	def5047 	correction de la suppression d'un noeud dans l'editeur de texte	2014-04-17
 Doelia	f1c210c	Ajout projet	2014-04-17
 Doelia	e68552c	Reconstruction	2014-04-17
 Doelia	e4425fa	Ajout gitignore v2	2014-04-17
 Doelia	4c6b61c	Suppresion gitignore	2014-04-17
 Doelia	3694322	Ajout gitignore	2014-04-17
 Doelia	14a5047	Réglage position widget	2014-04-17
 Znoraka	61be6d3	implementation de la methode de supression de noeud dans l'editeur de texte	2014-04-17

FIGURE 7.3 – Historique des commits - Page 3






































	Doelia	8cc4247	Répercution suppression d'un noeud -> suppression dans l'arbre	2014-04-17
	Doelia	f240371	Ajout recupeation de la pile pour getItemFromNod	2014-04-17
	Znoraka	7055328 	Merge branch 'master' of https://bitbucket.org/Doelia/xmlia	2014-04-16
	Znoraka	5c8c703	correction d'un bug de coloration	2014-04-16
	Beny13	8ae431d	Implémentation de ModeleXml::removeNode	2014-04-16
	Znoraka	1305237	ajout d'une methode permettant deretourner un QDomNode a partir de la position du curseur dans le texte	2014-04-15
	Znoraka	78ba8f7 	;	2014-04-15
	Znoraka	b306304	;	2014-04-15
	Doelia	1ee36cb	Correction de bug dans l'arbo	2014-04-15
	Doelia	1f97e54	Ajout slots de suppression de node	2014-04-15
	Doelia	f50c4b0	Ajout de documentation au modèle	2014-04-15
	Doelia	3b3edfc	Ajout documentation dans arbo + signaux d'édition de node	2014-04-15
	Znoraka	79413ed	correction de bug sur la modification du texte a partir de l'arbo	2014-04-11
	Znoraka	af8e278	mise a jour de l'editeur de texte lors de la modification de l'arbre	2014-04-10
	Znoraka	a871f29 	correction de bugs	2014-04-10
	Znoraka	243a5cc	debut de solution pour la mise a jour du texte dans l'editeur de texte	2014-04-10
	Beny13	1c7c514	Fix de updateNodeName : on peut maintenant modifier le nom des QDomElement seulement	2014-04-10
	Beny13	6ebacbd 	Implémentation de la méthode de mise à jour du modèle après renommage d'un node dans l'arbre. Merge	2014-04-10
	Beny13	7c47c4a	Implémentation de la méthode de mise à jour du modèle après renommage d'un node dans l'arbre.	2014-04-10
	Doelia	3817204 	Merge	2014-04-10
	Doelia	7eeb1c3	Ajout menu contextuel sur l'arbo	2014-04-10
	Znoraka	359ff81 	mise en place de l'architecture pour hitecture pour la modification du modele vers les vues	2014-04-10
	Znoraka	2b42799	merge	2014-04-10
	Doelia	de2710e	Debug d'un crash au démarrage	2014-04-10
	Doelia	e39a6c3 	Merge branch 'master' of https://bitbucket.org/Doelia/xmlia	2014-04-10
	Znoraka	c21d554 	insertion automatique des balises fermantes	2014-04-10
	Znoraka	e75b8d2	insertion automatique des balises fermantes	2014-04-10
	Doelia	1b7821f	Ajout méthode à faire : Modification d'un noeud du modèle	2014-04-10
	Doelia	9eeac71	Detection du node modifié à partir d'une edition dans la vue arborescente	2014-04-10

FIGURE 7.4 – Historique des commits - Page 4




































 Znoraka	6269ef3 	;	2014-04-09
 Znoraka	2f9367d	ajout d'une methode pas forcement utile	2014-04-09
 nlephilippe	4c6979a 	ajout d'une methode peut etre utile	2014-04-08
 Doelia	74ab9d6	Detection modification de l'arbre	2014-04-08
 Doelia	93babbd	Modification slot edition	2014-04-08
 nlephilippe	9ba539c	;	2014-04-08
 Doelia	b5bc1c7	Ajout signal d'édition	2014-04-08
 Doelia	f1281b7	Ajout de la méthode de mise à jour de l'arbre	2014-04-08
 Doelia	2740391 	Merge branch 'master' of https://bitbucket.org/Doelia/xmlia	2014-04-08
 nlephilippe	8818303	;	2014-04-08
 Doelia	9edf854	;	2014-04-08
 Znoraka	112c08d	amelioration de l'indentation et de la coloration	2014-04-07
 nlephilippe	1b5193d	;	2014-04-03
 Znoraka	e202989 	;	2014-04-01
 Znoraka	003d258 	;	2014-04-01
 Doelia	397d104	;	2014-04-01
 Doelia	71f00cf	;	2014-04-01
 Znoraka	344b7ed 	;	2014-04-01
 Doelia	0026cdf	Ajout de la vue arbo au projet principal	2014-04-01
 Znoraka	7cd7c6e	amelioration de l'indentation	2014-03-31
 Znoraka	37b8d29	nrs	2014-03-18
 Znoraka	59f4dd5	amelioration de la coloration	2014-03-17
 Znoraka	e3be345	amelioration de l'indentation	2014-03-17
 Znoraka	a91e4e7	amelioration de la coloration	2014-03-12
 nlephilippe	507a501	correction d'un bug sur la detection de balises	2014-03-11
 Znoraka	b12e6db	factorisation du code	2014-03-11
 Znoraka	d5cefbf 	Merge branch 'master' of https://bitbucket.org/Doelia/xmlia	2014-03-11
 Znoraka	fc0644d	correction de l'indentation avec les fichier cette fois	2014-03-11

FIGURE 7.5 – Historique des commits - Page 5












 Doelia	1abc8db <small>M</small>	Merge branch 'master' of https://bitbucket.org/Doelia/xmlia	2014-03-10
 Doelia	dcd0766	Reorganisation et création du projet principal	2014-03-10
 Doelia	2ae499a	TODO_Arbo deleted online with Bitbucket	2014-03-10
 Doelia	04aacc9	TODO_Arbo created online with Bitbucket	2014-03-10
 Doelia	3eda83d	Préparation code pour lecture XML	2014-03-10
 Doelia	59d5b79	Lecture fichier XML corrigée	2014-03-10
 Doelia	4d4bc8a	Test de lecture de fichier .xml (non fonctionnel)	2014-03-10
 Doelia	e42873a <small>M</small>	Merge branch 'master' of https://bitbucket.org/Doelia/xmlia	2014-03-10
 Doelia	0354c37	Ajout début module Arbo	2014-03-10
 Znoraka	28c63c7	correction de bugs sur l'indentation	2014-03-08
 Znoraka	0e57b7e	amelioration de l'indentation	2014-03-05
 Znoraka	8dad3ac	ajout initial de l'editeur de texte	2014-03-04
 Znoraka	bc45f4b	First commit. Adding a README.	2014-03-04

FIGURE 7.6 – Historique des commits - Page 6