|  |
| --- |
| **Rapport PSTL** |
| [Tapez le sous-titre du document] |
|  |
| [Tapez le résumé du document ici. Il s'agit généralement d'une courte synthèse du document. Tapez le résumé du document ici. Il s'agit généralement d'une courte synthèse du document.] |
|  |
| **Morvan Lassauzayet Victor Nea** |
| **30/06/2016** |
|  |

LES BOUTS DE PHRASES EN ROUGES SONT DES NOTES QUI DEVRONT ETRE SUPPRIMEES.

(PARTIES II ET III à refaire, il faudrait fusionner les deux parties)

(Il faudrait également renommer certaines sous parties)

Sommaire

[Introduction 3](#_Toc449627786)

[I - Présentation générale du projet 4](#_Toc449627787)

[1. Présentation DOT + GraphViz 4](#_Toc449627788)

[2. Veille concurrentielle 5](#_Toc449627789)

[3. Vue d'ensemble de l'application espérée 6](#_Toc449627790)

[4. Digramme de use case 7](#_Toc449627791)

[5. Outils choisis et pourquoi 8](#_Toc449627792)

[II - Architecture 9](#_Toc449627793)

[1. Architecture générale 9](#_Toc449627794)

[2. Architecture de chaque composant et description générale 9](#_Toc449627795)

[3. Liaison inter-composants: description plus fines car explication des méthodes choisies et de leur forme 9](#_Toc449627796)

[III - Interface et éléments graphiques 10](#_Toc449627797)

[1. Description des outils utilisés 10](#_Toc449627798)

[2. Gestion graphique des nœuds et arêtes 10](#_Toc449627799)

[3. Gestion vue textuelle 10](#_Toc449627800)

[IV - Maintenabilité et distribution 11](#_Toc449627801)

[1. Exemple d'extension 11](#_Toc449627802)

[2. PEP, licence, packaging 11](#_Toc449627803)

[Conclusion 12](#_Toc449627804)

[Tables des illustrations 13](#_Toc449627805)

[Annexes 14](#_Toc449627806)

Introduction

Dans le cadre de notre formation en 1ère année de Master spécialité STL, nous avons été amenés à développer une application nommée dotEd pour l'UE PSTL. (A compléter)

I - Présentation générale du projet

## 1. Présentation DOT + GraphViz

Le langage DOT est un langage de description de graphe dans un format texte (extension dot). Il est donc possible de définir des nœuds, des arrêtes, des sous-graphes, etc et de leur définir des attributs comme un label, une couleur, ..

Voici la grammaire du langage DOT (à mettre en annexe ou inutile peut-être ?) :

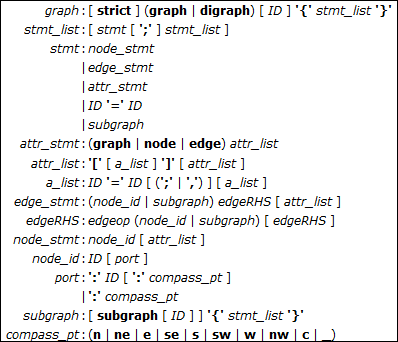


Figure : Grammaire du langage DOT

(source : http://www.graphviz.org/content/dot-language)

A l'aide du logiciel GraphViz (Graph Visualization Software), il est possible d'importer des fichiers dot afin de visualiser le graphe sous forme graphique. Il est ainsi possible de sauvegarder ce rendu graphique sous forme d'image.

Voici un exemple d'un fichier dot et de sa représentation graphique avec GraphViz :



Figure : Représentation graphique d'un fichier DOT

Utilité de notre logiciel :

GraphViz est un outil très puissant pour la génération graphique de graphe. Cependant il n'est pas possible de faire des actions (création, édition ou suppression d'un élément) directement sur la vue graphique comme c'est une image. Le seul moyen est donc de modifier le fichier dot puis regénérer le graphe. Ce que nous souhaitons c'est donc de pourvoir modifier les éléments directement sur la vue graphique et sur la vue textuelle.

## 2. Veille concurrentielle

(retrouver lien Stack Overflow puis tester chaque logiciel).

## 3. Vue d'ensemble de l'application espérée

(screenshot ou maquette + description ?)

Voici une maquette de ce que l'on voudrait obtenir :



Figure : Maquette de l'application

1: Barre de menus

2: Vue graphique pour afficher le graphe

3: Vue textuelle pour afficher le fichier dot

## 4. Digramme de use case

(à discuter sur ce diagramme, il faudra surement le changer, peut-être faire des use case sur les vues, peut-être mettre des "extends" pour créer-éditer et créer-supprimer).

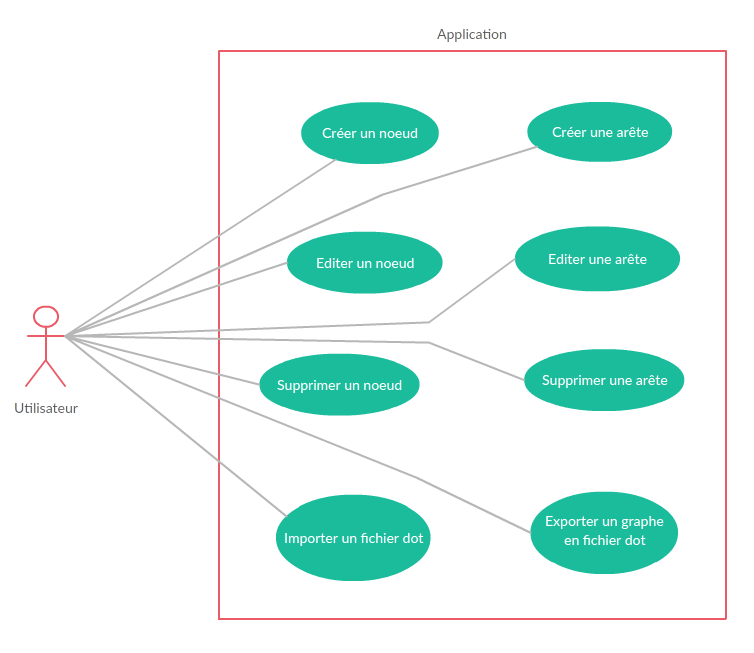


Figure : Diagramme de use case

Remarque:

Lorsque l'on parle d'éditer un nœud ou une arête, il s'agit de l'édition d'un de ses attributs (par exemple son label).

## 5. Outils choisis et pourquoi

(Insérer des images peut-être ?)

Tout d'abord, nous avons choisis le langage Python pour développer notre application. Nous avons choisis ce langage pour plusieurs raisons :

* la portabilité : comme Python est multiplateforme, il est facile de distribuer l'application sur les différents plateformes.
* la gestion de la mémoire : la désallocation d'une ressource est automatiquement gérée par Python, il n'est donc pas nécessaire de se soucier de la libération des ressources ce qui permet d'éviter les fuites de mémoires et d'avoir une application robuste.
* la documentation : Python est très bien documentée, il est facile de trouver ce que l'on cherche.
* la bibliothèque standard : elle est très vaste ce qui permet de ne pas réinventer la roue et d'avoir une application robuste.
* PEP (Python Enhancement Proposal) : nous en parlerons plus tard
* c'est un langage orienté objet (peut-être qu'il faut le retirer ou argumenter ?)

En ce qui concerne la bibliothèque graphique, nous avons décidé d'utiliser PyQt (qui est la version Python de Qt). PyQt a été développée dans le but de créer des interfaces utilisateurs et c'est ce qui nous intéresse dans notre cas. De plus elle est très bien documentée et est multiplateforme. C'est également une des seules (voire la seule) bibliothèque graphique à disposer d'une scène pour manipuler des objets graphiques. Il n'est donc pas nécessaire de recoder toute la partie qui permet de gérer l'ajout, le déplacement ou la surpression d'un élément graphique de la scène.

Ensuite, nous avons utilisé Git pour partager facilement le code source, avoir un backup en cas de problème et avoir un contrôle de version.

Enfin nous avons utilisé l'IDE LiClipse qui est une version customisée d'Eclipse. Il n'est pas primordial d'utiliser un IDE pour développer mais LiClipse apporte différents avantages :

* le plugin PyDev qui permet de développer en Python.
* le plugin EGit qui permet d'utiliser Git directement dans LiClipse via l'interface graphique.
* un débogueur pour analyser les bugs du programme.
* l'auto-complétion qui permet de gagner en rapidité.

II - Architecture

(partie à changer)

## 1. Architecture générale

## 2. Architecture de chaque composant et description générale

## 3. Liaison inter-composants: description plus fines car explication des méthodes choisies et de leur forme

III - Interface et éléments graphiques

(partie à changer)

## 1. Description des outils utilisés

## 2. Gestion graphique des nœuds et arêtes

## 3. Gestion vue textuelle

IV - Maintenabilité et distribution

## 1. Exemple d'extension

## 2. PEP, licence, packaging

PEP : c'est un document de conception qui décrit l'ensemble des conventions et bonnes pratiques pour coder en Python. En respectant ces normes, le code sera plus maintenable et une nouvelle personne qui souhaite travailler sur ce projet aura beaucoup plus de facilité à le comprendre.

Licence de base : GNU GPL

Peut-être prendre la licence MIT qui est totalement libre ?

Packaging :

Conclusion

(partie à compléter)

Tables des illustrations

[Figure 1: Grammaire du langage DOT 4](#_Toc449627449)

[Figure 2: Représentation graphique d'un fichier DOT 5](#_Toc449627450)

[Figure 3: Maquette de l'application 6](#_Toc449627451)

[Figure 4: Diagramme de use case 7](#_Toc449627452)

Annexes

(Peut-être que cette partie est inutile)