MASTER EN SCIENCES INFORMATIQUES

Vers un nouvel éditeur graphique de modèles de variabilité

Talon, Jean-François

Award date:
2015

Awarding institution:
Universite de Namur

Université de Namur
Faculté d'informatique
Année académique 2014-2015

Vers un nouvel éditeur graphique de modèles de variabilité

Génération d'IHM de configuration à partir de TVL

Auteur:
Jean-François Talon

Promoteur:
Pr. Patrick Heymans
Co-promoteur:
Germain Saval

Mémoire présenté en vue de l'obtention du grade de
Master en informatique

Remerciements

A Mr Patrick Heymans,
Professeur en Génie logiciel,
Pour ses idées de mémoire

A Mr Germain Saval,
Chercheur au laboratoire PReCISE,
Pour son suivi et ses conseils avisés

Au corps enseignant,
Pour m'avoir supporté durant toutes ces années

A Alexandra Schmitt,
Pour son soutien indéfectible

A Anamaria Ionescu,
Pour sa relecture

A Tony Leclercq,
Pour ses encouragements

A Ma famille,
Pour leur soutien

Contents

Remerciements **1**

1 Introduction **5**

2 Introduction au modèle de variabilité et TVL **7**
2.1 Qu'est ce que la variabilité et à quoi sert-elle? 7
2.2 Introduction à TVL 8
2.3 Modèle graphique contre modèle textuel 12
2.4 Conclusion 12

3 État de l'art de la visualisation de graphes **14**
3.1 Définitions autour du graphe 14
3.2 Visualisation de graphes 16
3.2.1 Utilisations 17
3.3 Méthode de dessin 17
3.3.1 Contraintes de la visualisation 18
3.3.2 Style de dessins 19
3.3.3 Algorithmes d'agencement de graphes 21
3.3.4 Avantages et inconvénients des agencements 24
3.4 Rendu final 26
3.4.1 Les rendus verticaux 26
3.4.2 Les rendus horizontaux 26
3.4.3 Les rendus circulaires 26
3.4.4 Avantages et inconvénients 27
3.4.5 Les autres rendus 27
3.5 Données présentes sur le graphe 29
3.5.1 Les nœuds 29
3.5.2 Les arêtes 29
3.6 Conclusion 31

4 État de l'art des outils de visualisation/design dédiés à la variabilité **33**
4.1 Technologie WYSIWYG 33
4.2 Types de représentations 33
4.3 Catégories d'outils 34
4.4 Outils existants 35

4.4.1 Scénarios de tests 36
4.5 Résultats 38
4.6 Enseignements 41
4.6.1 Installation 41
4.6.2 Interface Homme-Machine 42
4.6.3 Prise en main 42
4.6.4 Fonctionnalité principale: La construction de graphes 43
4.6.5 Fonctionnalités externes aux processus métier 43
4.6.6 Fonctionnalités avancées 44
4.6.7 Support 44
4.7 Conclusions 44

5 Manquements et fonctionnalités essentielles des outils de visualisation dédiés à la variabilité 46
5.1 Fonctionnalités essentielles 46
5.1.1 Afficher le graphe orienté 47
5.1.2 Ajouter des noeuds ou sommets au graphe 47
5.1.3 Sauvegarder dans un fichier et ouverture depuis ce fichier 47
5.1.4 Editer des propriétés des éléments du graphe 47
5.1.5 Supprimer des éléments 47
5.1.6 Déplacer tout ou une partie du graphe 47
5.1.7 Exporter le graphe dans une image 47
5.1.8 Naviguer dans le graphe 48
5.1.9 Créer un nouveau graphe 48
5.1.10 Support des contraintes 48
5.2 Manquements détectés dans les outils 48
5.2.1 Support des attributs 48
5.2.2 Automatisation de l'optimisation 48
5.2.3 Pouvoir changer le style de dessin 48
5.2.4 Introduire la notion de sous-graphe avec zoom et nouvel onglet . 49
5.2.5 Styler les noeuds ou les arcs 49
5.2.6 Filtre sur le graphe 49
5.3 Conclusions 49

6 Vers un nouvel outil dédié à la variabilité 51
6.1 Fonctionnalités retenues 51
6.1.1 Couper/Copier/Coller un noeud 51
6.1.2 Zoomer dans le graphe 51
6.1.3 Ajouter un segment dans un arc multiligne 53
6.1.4 Consulter les noeuds environnant d'un noeud sélectionné 53
6.2 Proposition d'écran 53
6.3 Prototype et retour d'expérience 54
6.3.1 Choix pour le développement 54
6.3.2 Choix technologiques 55
6.3.3 Développement du prototype 55

6.4 Critiques du prototype 56
6.5 Conclusions 57

7 Conclusion **58**

A Annexe 1 **64**
A.1 Captain Feature 64
A.2 Feature Mapper 66
A.3 Feature Modeling 69
A.4 Feature IDE 71
A.5 Hypersense 74
A.6 MetaEdit 75
A.7 PureVariants 77
A.8 SPLOT 80
A.9 VMC 82
A.10 XFeature 84

B Annexe 2 **86**
B.1 Global package 86
B.2 View Package 98
B.3 User Action Package 106
B.4 Provider Package 106
B.5 Listener Package 110
B.6 Interface Package 114
B.7 Input Package 115
B.8 Factory Package 116
B.9 editing Package 120
B.10 Component Package 127
B.11 Adapter Package 131
B.12 Projects Files 143

Chapitre 1

Introduction

Dans ce mémoire qui conclut le cursus de master en informatique, le sujet abordé est en rapport avec le laboratoire PReCISE de l'université de Namur. En effet, ce laboratoire a créé un nouveau langage, appelé TVL ou "Text-based Variability Language". Il est utilisé dans le cadre de la variabilité appliqué au développement de lignes de produits logiciels. Il permet de concevoir des artefacts réutilisables entre différents logiciels sous forme de feature models.

Ce langage est purement textuel, tout y est décrit à l'aide de lignes de code. Il n'y a actuellement aucun outil pour transformer le code en représentations graphiques. Celles-ci offrent des avantages dans la compréhension et l'accessibilité des concepts utilisés. Il est en effet plus facile de comprendre un dessin que de lire des lignes de code.

Le but de ce mémoire est donc de trouver un moyen de rendre accessibles ces lignes de code. Les graphes sont le moyen d'y arriver car la représentation graphique de ce concept mathématique correspond aux besoins que nous avons, c'est-à-dire un ensemble d'objets hiérarchisés. L'objectif du mémoire est de définir ce dont l'outil de visualisation aura besoin comme fonctionnalités, qu'elles soient principales ou secondaires.

Les contributions apportées seront :

- Un état de l'art des représentations de graphe;
- Un état de l'art des outils existants pour créer de la variabilité;
- Les fonctionnalités essentielles d'un outil et les manquements constatés dans les outils testés;
- Une proposition d'outil avec un test sous la forme d'un prototype;

Ce mémoire est structuré comme suit:

- Le premier chapitre traite d'une introduction sur les différents concepts décrits ici comme TVL, la variabilité, les feature models.

- Le deuxième chapitre propose une analyse de ce qu'est un graphe et ses méthodes de visualisation.
- Dans le troisième chapitre, une analyse des différents outils existants pour modéliser graphiquement la variabilité est présentée.
- Le quatrième chapitre réalise un synthèse des deux analyses précédentes pour en retirer l'essentiel et ce qui manque aux outils.
- Le dernier chapitre contient une proposition d'outil avec ses fonctionnalités et une interface graphique ainsi qu'un test grandeur nature sous forme d'un prototype.
- Une conclusion reprenant les résultats des précédents chapitres clôturera ce mémoire.

Chapitre 2

Introduction au modèle de variabilité et TVL

Ce chapitre contient les présentations

- du concept de variabilité et de ce qui l'entoure
- de TVL
- et d'une comparaison entre les modèles graphique et textuel

2.1 Qu'est ce que la variabilité et à quoi sert-elle?

Un *ligne de produits logiciels* [Uni15] est un ensemble de logiciels qui partagent une base commune de fonctionnalités connues, satisfaisant des besoins spécifiques d'un segment particulier de marché, et qui sont développés depuis un ensemble commun d'assets d'une manière déterminée.
Un *feature model* [Wik15a] [Figure 2.1] est une représentation compacte de tous les produits d'une ligne de produits logiciels en termes de features. Les feature models sont visuellement représentés par des diagrammes de fonctionnalités. Ils sont largement utilisés durant le processus du développement de toute la ligne de produits et sont communément utilisés comme entrées pour produire d'autres assets comme des documents, des définitions d'architecture ou des parties de code. La variabilité est, dans le cadre des lignes de produits logiciels, l'ensemble des paramètres qui peuvent être modifiés sur les différents assets afin de construire la ligne de produits.

Certains *assets* peuvent légèrement fluctuer entre les produits et les comportements modifiés par [FB15]:

- des configurateurs pour modifier certaines variables de l'asset
- des générateurs pour générer un asset à partir de sa spécification
- de l'héritage pour étendre un asset avec des comportements supplémentaires ou redéfinis

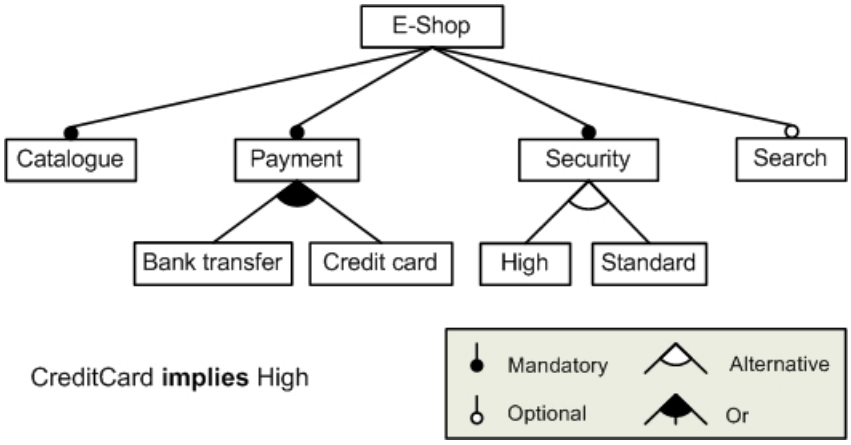


Figure 2.1: Modèle de fonctionnalités [Seg15]

- des paramètres internes pour varier un asset à la compilation grâce à des variables de pré-processeurs (#ifdef)

La variabilité sert donc à permettre aux différents assets d'être réutilisés dans plusieurs produits avec peu ou pas de modifications et ainsi de gagner du temps lors du développement de lignes de produits.

Appliquée à la figure 2.1, la variabilité se placera au niveau des possibilités optionnelles, comme la recherche (search), ou alternatives, comme la sécurité standard ou high.

2.2 Introduction à TVL

TVL, pour "Text-based Variability Language", est un langage de modélisation textuel, léger et compréhensible et couvre la plupart des constructions et langages existants, incluant les décompositions basées sur des cardinalités et les attributs de features.

La plupart des auteurs utilisent des notations graphiques basées sur l'analyse de domaines orientés fonctionnalités ou "Feature-oriented domain analysis" (FODA). Ces diagrammes sont représentés sous forme d'arbres et sont supposés plus accessibles pour les intervenants non-techniques. Malheureusement, ces diagrammes, surtout quand ils sont très grands, deviennent peu lisibles et les comprendre devient une tâche complexe à cause, entre autres, du manque de navigabilité ou de possibilité de recherche entre les fonctionnalités.

TVL utilise du texte pour décrire les modèles, ce qui a comme avantage de pouvoir profiter d'outils existants traitant déjà avec du texte, comme des programmes d'édition de code. De plus, il a une syntaxe proche du langage C, ce qui permet à tous ceux qui ont déjà développé dans ce langage ou un des langages s'en étant inspiré de s'y retrouver rapidement.

TVL est extensible car il est succinct et offre un nombre de mécanismes pour modulariser et séparer strictement les problèmes. Le langage est compréhensible parce qu'il intègre la plupart des constructions existantes de modèles de fonctionnalités.

TVL utilise des mots clés pour spécifier ces différentes fonctionnalités. **root** définit la feature racine.

Pour les décompositions:

- **group** définit une décomposition
- **allOf** est utilisé pour une décomposition dans laquelle tous les éléments sont obligatoires
- **oneOf** pour une décomposition dans laquelle un seul élément doit être sélectionné
- **someOf** pour une décomposition dont au moins un des éléments doit être sélectionné
- **group[i..j]** pour une décomposition basée sur les cardinalités avec i comme minimum et j comme maximum. Une des limites peut être remplacée par * pour dénoter que ce nombre n'est pas une limite
- **opt** pour les éléments optionnels

Exemple:

```
root Computer {
        group allOf{
                Motherboard,
                CPU[1..2],
                GraphicCard,
                opt Accessories
        }
}
```

Certains paramètres peuvent être partagés entre plusieurs features grâce au mot-clé **shared**

Exemple:

```
root A {
        group oneOf {
                B group allOf{D},
                C group allOf{shared D}
        }
}
```

Une feature peut avoir des attributs

- **int** pour une variable entière
- **real** pour une variable réelle
- **bool** pour une variable booléenne
- **enum** pur une énumération

Les valeurs par défaut peuvent être assignées par le mot-clé **in**.
Dans le cas d'une valeur optionnelle, des gardes existent pour spécifier la valeur par défaut, si elle est sélectionnée, **ifIn**, ou non, **ifOut**.
Une variable peut être la somme d'attributs sélectionnés grâce à la combinaison de mots-clés **sum** et **selectedChildren**.
Exemple:

```
Motherboard {
        real price is sum(selectedChildren.price};
        int usbNumber is 4;
        bool hasVGAPort;
        enum socket in {LGA252, BIN215};
}
```

Des expressions sont utilisées pour déterminer les valeurs d'un attribut ainsi que pour exprimer des contraintes dans le *Feature Model*, et chaque expression a un type car le langage TVL est fortement typé (type *bool*, *int* ou *real*).
Les opérateurs sont : **+**, **-**, **/**, *****, **abs** pour les valeurs numériques, **!**, **&&**, **||**, **->**, **<->** pour les booléens et **>**, **<=**, **<**, **<=** pour les comparaisons.
Il existe aussi des fonctions d'agrégation comme **sum, mul, min, max, avg, count, and, or** et **xor** qui peuvent être utilisés sur des listes ou en conjonction avec les mots-clés **children** ou **selectedChildren**.

Les contraintes sont juste des expressions booléennes qui sont ajoutés dans les définitions d'une feature. Les gardes **ifIn** et **ifOut** peuvent être utilisées sur ces contraintes.
Les mécanismes de modularité sont:

- des types définis par l'utilisateur au début du fichier
- la définition des structures à l'aide du mot-clé **struct**
- la définition des constantes à l'aide du mot-clé **const**
- l'inclusion des références vers d'autres fichiers grâce au mot-clé **include**
- l'étension des types déjà définis en ajoutant des blocs portant le même nom au début d'une nouvelle hiérarchie

Exemples:

```
struct dimension {
        int height;
        int width;
}
Motherboard {
        dimension size;
}
```

```
const in maxRamBlocks 4;
```

```
include (./some/other/file);
```

```
root Computer {
        group allOf {
                Motherboard,
                CPU,
                GraphicCard,
                opt Accessories
        }
}
Computer {
        int price is sum(selectedChildren.price);
)
```

2.3 Modèle graphique contre modèle textuel

Chaque modèle présente des avantages et des inconvénients [IDcsc15]:

	Avantages	Inconvénients
Modèle graphique	• Plus facile à comprendre pour les intervenants non-techniques • Navigation plus facile • Plus facile à apprendre	• Difficile à développer et maintenir • Format sauvegardé différent du format affiché
Modèle textuel	• Existe beaucoup d'outils pour gérer le texte plein • Développeurs habitués à utiliser ce type de modèle • Rapidité de développement grâce à de l'automatisation (raccourcis, macros) • Compact par rapport à la densité d'informations	• Difficile de comprendre la structure • Verbosité parfois trop importante • Navigation pas aussi intuitive que sur les modèles graphiques

Le modèle textuel est donc plus adressé à des utilisateurs avertis et le modèle graphique à des utilisateurs moins expérimentés. Ces deux modèles sont donc complémentaires car le modèle graphique apporte des fonctionnalités que le modèle textuel n'a pas et vice-versa.

2.4 Conclusion

Le concept de variabilité permet de créér des lignes de produits logiciel à partir des composants existants avec peu ou pas de travail supplémentaire. Ces lignes de produits sont décrites au moyen de feature models.

TVL est un langage qui permet de créer des feature models qui mettent en oeuvre la variabilité et qui sont décrits par des concepts à partir de décompositions, attributs, contraintes, opérateurs et mécanismes de modularité. Il est uniquement textuel.

Les modèles textuel et graphique sont des vues différentes et complémentaires d'un problème identique.
Elles s'adressent chacune à des besoins précis de catégories distinctes d'utilisateurs.

Chapitre 3

État de l'art de la visualisation de graphes

Ce chapitre décrit:

- ce qu'est un graphe et les notions qui l'entourent
- ce que l'on entend par visualisation de graphes et ses applications
- les différents algorithmes permettant de représenter un graphe en deux dimensions
- comment trier les graphes en fonction de leur rendu visuel
- comment représenter des informations sur un graphe

3.1 Définitions autour du graphe

Dans la théorie des graphes, les définitions suivantes s'appliquent:

Un *graphe non orienté* [RL15] [Figure 3.1] est un triple(V,E,φ) tel que :

- V est un ensemble dont les éléments sont appelés sommets ou nœuds
- E est un ensemble dont les éléments sont appelés arêtes
- φ est une fonction, dite fonction d'incidence, qui associe à chaque arête un sommet ou une paire de sommets

Un *graphe orienté* [Ver15] [Figure 3.2] est un graphe dont les arêtes sont définies par leur origine et leur extrémité, c'est-à-dire dont les arêtes sont orientées, munies d'un sens.
Une *arête* [Mü15] relie deux sommets dans un graphe.
Un *arc* [Wik15f] est un couple (ensemble ordonné de deux éléments) de sommets reliés par une arête dans un graphe orienté. Si le couple est noté (A,B), la connexion ira de A vers B.

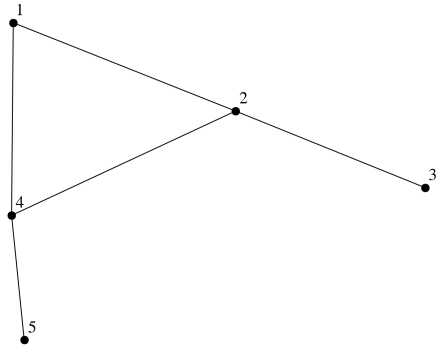


Figure 3.1: Graphe non-orienté

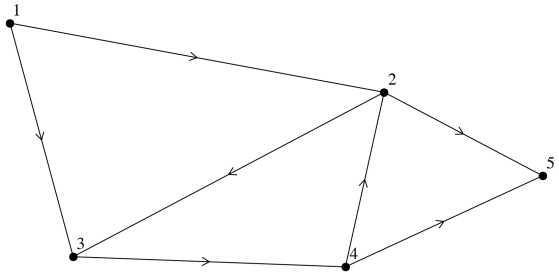


Figure 3.2: Graphe orienté

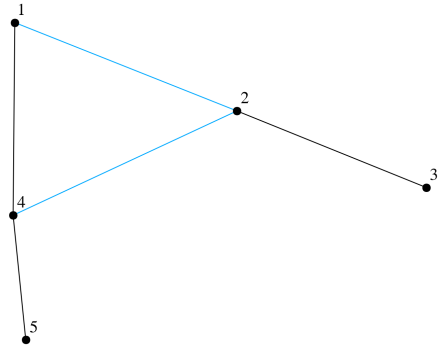


Figure 3.3: Parcours

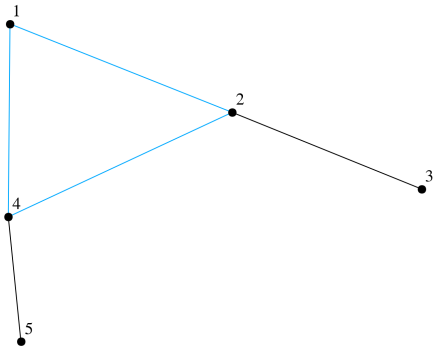


Figure 3.4: Cycle

Quelques définitions pour parcourir les graphes:
Un *parcours* [RL15] [Figure 3.3] est une suite $v_0e_1v_1e_2$...en v_n où v_0v_1... sont des sommets et e_1e_2... sont des arêtes reliant les sommets consécutifs du parcoursnewline Un *cycle* ou *circuit* [Wik15f] [Figure 3.4] est un parcours dont les sommets de départ et de fin sont les mêmesnewline Un graphe non-orienté est *connexe* [RL15] si, pour chaque paire de points, il existe un parcours qui les relie. La figure 3.3 est un graphe *connexe*.
Un *arbre* [Wik15f] est un graphe connexe sans cycle.
Un noeud X est *prédécesseur* [Pol15a] d'un noeud Y si un arc va de X vers Y.
Une *racine* [Rap15] [Figure 3.5] est le sommet sans prédécesseur dans un arbre.
Un graphe sans circuits peut être hiérarchisé par *niveaux* [Del15] [Figure 3.6] de sorte que tout arc du graphe mène d'un sommet vers un sommet de niveau supérieur.

Des définitions spécifiants certaines matrices d'un graphe:
Étant donné un graphe G = (V, E) contenant n sommets, la *matrice des degrés D* [Wik15d] de G est la matrice carrée $n \times n$ définie par :

$$d_{i,j} := \begin{cases} \deg(v_i) & \text{si } i = j \\ 0 & \text{sinon} \end{cases} \tag{3.1}$$

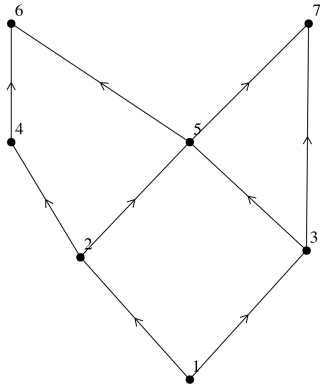


Figure 3.5: Racine

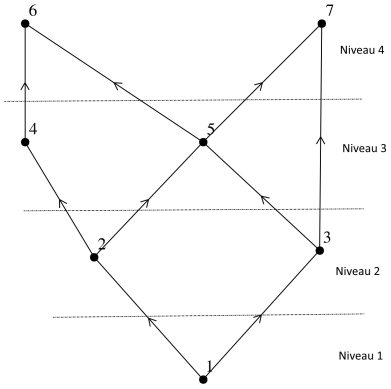


Figure 3.6: Niveau

$$D = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.4}$$

Figure 3.7: Matrice des degrés du graphe 3.1

$$M = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{3.5}$$

Figure 3.8: Matrice d'adjacence du graphe 3.1

où $\deg(v_i)$ est le nombre d'arêtes ou d'arcs reliés au sommet courant
Une structure de données simple pour représenter un graphe est la *matrice d'adjacence* M [Pol15b].
Pour obtenir M, on numérote les sommets du graphe de façon quelconque:

$$X = \{x_1, x_2 ... x_n\} \tag{3.2}$$

M est une matrice carrée $n \times n$ dont les cœfficients sont 0 et 1 telle que:

$$M_i, j = 1 \ si(x_i, x_j) \in A, M_i, j = 0 \ si(x_i, x_j) \notin A \tag{3.3}$$

où A est l'ensemble des arcs du graphe
La *matrice laplacienne* [Wik15e] d'un graphe G non orienté est définie par : L = D - A où D est la matrice des degrés de G et A la matrice d'adjacence de G1. Elle vérifie :

$$L_{i,j} := \begin{cases} \deg(s_i) & \text{si } i = j \\ -1 & \text{si } i \neq j \text{ et si i et j sont reliés par une arête} \\ 0 & \text{sinon} \end{cases} \tag{3.6}$$

3.2 Visualisation de graphes

La visualisation de graphes est la manière de présenter de visu un graphe sur un document, un écran ou un autre support.

$$M = \begin{pmatrix} 2 & -1 & 0 & -1 & 0 \\ -1 & 3 & -1 & -1 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ -1 & -1 & 0 & 3 & -1 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix} \quad (3.7)$$

Figure 3.9: Matrice de Laplace du graphe 3.1

Il y a plusieurs manières de représenter un graphe visuellement:

- sous forme de matrice [Figure 3.8]
- sous forme de dessin [Figure 3.1]

C'est sous la forme de dessin que le graphe nous intéresse le plus. En effet sa compréhension est fortement facilitée dans cette représentation.

3.2.1 Utilisations

La visalisuation de graphes est usitée dans de nombreux domaines pour présenter des informations connectées entre elles:

- Dans le domaine électrique, pour les schémas de circuits imprimés ou la modélisation de circuits électriques
- En chimie, pour modéliser les molécules et leurs compositions atomiques
- En biologie, pour représenter les espèces, leurs lieux de répartitions
- En mathématiques, dans la géométrie
- En génie logiciel, UML est basé sur des graphes
- Les bases de données sont modélisées grâce à des représentations de graphes
- ...

3.3 Méthode de dessin

Pour tracer un graphe, la méthode se résume, la plupart du temps, à dessiner des points reliés entre eux par des lignes, les points faisant office de noeuds et les lignes d'arêtes ou d'arcs. Cela parait simple, ça l'est pour des petits graphes comportant peu de points ou d'arcs, mais il devient vite compliqué de dessiner des graphes comportant beaucoup plus d'informations.

L'informatisation des processus permet dorénavant d'automatiser le dessin de graphes en leur appliquant un algorithme pour le convertir d'une matrice vers une représentation graphique. Une fois les calculs effectués, chaque noeud aura une série de coordonnées à deux (ou trois) dimensions. Les algorithmes seront décrits dans la partie suivante.

3.3.1 Contraintes de la visualisation

Plusieurs contraintes sont à prendre en compte quand on dessine un graphe:

- Le support cible
- Le style de dessin
- L'esthétique
- La sémantique

Le support cible

En fonction de la place disponible sur le support, le graphe aura ou non de l'espace pour s'étendre. Plus la place sera petite, moins d'informations et de détails seront affichés sur le dessin afin de ne pas le surcharger. A l'inverse, plus la superficie est grande, plus de données peuvent être visualisées.

Le support cible dépend du format visé comme un écran d'ordinateur avec sa résolution et sa taille, une feuille imprimée...

Le style de dessin

En fonction des conventions, des données à visualiser et du support, un style de dessin adéquat doit être choisi. Les styles sont décrits dans la section 3.3.2.

L'esthétique

Afin de présenter le graphe de la meilleure manière possible et de rendre les informations facilement compréhensibles et mémorisables, l'esthétique est une composante majeure. Certaines parties vont diminuer la compréhension comme des intersections entre les arcs, des longueurs d'arcs trop importantes. D'autres vont permettre d'assimiler les données plus facilement comme la symétrie, l'espace entre les points.

La sémantique

La sémantique permet d'ajouter des informations sous forme de symboles afin de ne pas surcharger le graphe. Elle peut prendre la forme d'un format de noeud précis, d'arcs dessinés en pointillés, du rapprochement des noeuds...

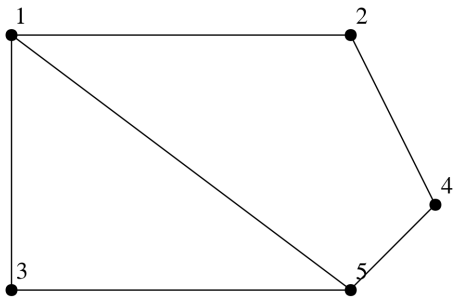


Figure 3.10: Dessin planaire

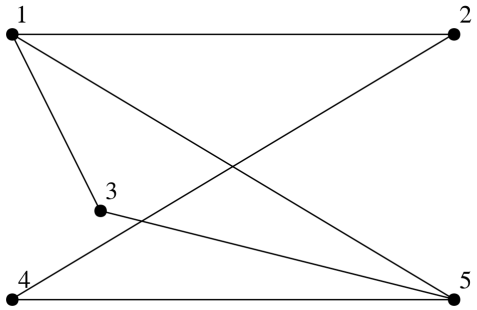


Figure 3.11: Dessin non-planaire

3.3.2 Style de dessins

Chaque représentation de graphe suit un style qui lui donne sa cohérence et sa simplicité. Il est choisi suivant le contexte d'utilisation, les simplifications que le dessin peut apporter et les informations à ajouter au graphe.

Dessin planaire

Le dessin d'un graphe est planaire [Figure 3.10] s'il ne possède aucune intersection entre ses arcs lorsqu'il est représenté sur un plan [Wik15b] [TNTUT04]. Donc pour qu'un graphe soit planaire, il suffit de lui trouver un dessin planaire. Malheureusement il n'est pas possible de trouver une représentation planaire pour chaque graphe. Il s'agit donc d'un cas particulier dans le dessin de graphes.

Trouver un dessin planaire pour un graphe est préférable étant donné que le graphe devient très facilement compréhensible.

Dessin en lignes droites

Dans le style [Figure 3.11] en "ligne droites", chaque arc est représenté par une ligne droite ou courbée mais sans brisure. Tout graphe planaire peut également être représenté par un dessin de ce style.

Dessin convexe

Un dessin est convexe si chaque cycle extérieur est dessiné comme un polygone convexe. C'est un cas particulier du dessin en lignes droites. La figure 3.10 est un dessin convexe.

Dessin en segments multiples

Dans cette représentation [Figure 3.12], chaque arc est composé d'un ou plusieurs segments de droites distincts. C'est une généralisation du dessin en lignes droites.

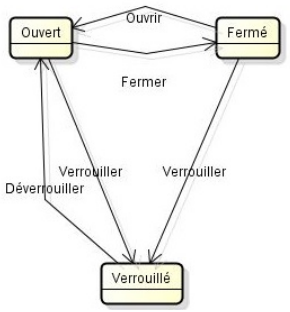


Figure 3.12: Dessin en segments multiples

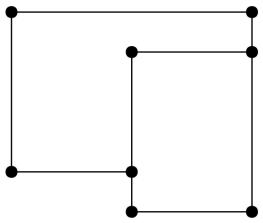


Figure 3.13: Dessin orthogonal

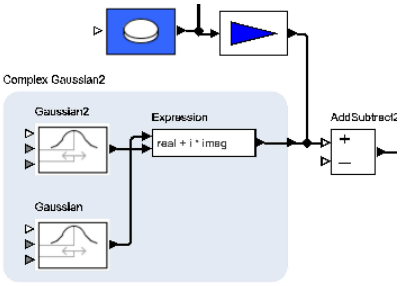


Figure 3.14: Dessin en boite orthogonale [UR15]

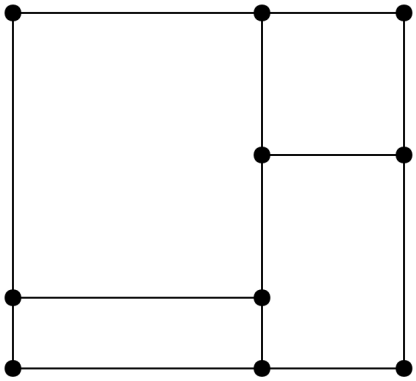


Figure 3.15: Dessin rectangulaire et en grille

Dessin orthogonal

Ce dessin [Figure 3.13] est un dessin en segments multiples où chaque segment est soit vertical soit horizontal.

Dessin en boite orthogonale

C'est un dessin orthogonal [Figure 3.14] où chaque sommet est représenté par une boite au lieu d'un point.

Dessin rectangulaire

Dans ce dessin [Figure 3.15], chaque arc est soit vertical soit horizontal sans croisement, le graphe est donc planaire. De plus chaque cycle est dessiné comme un rectangle.

Dessin boite rectangulaire

C'est un dessin rectangulaire où chaque sommet est représenté par une boite au lieu d'un point.

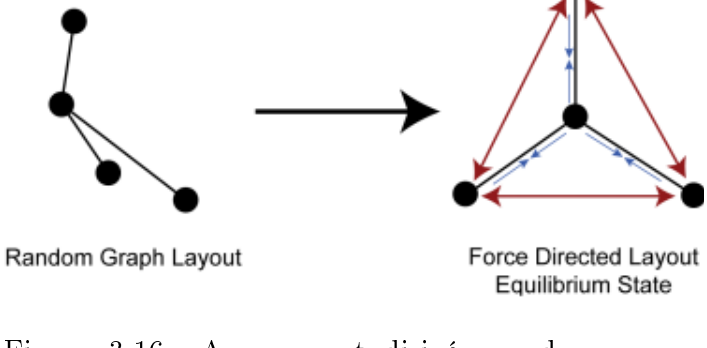


Figure 3.16: Agencement dirigé par des forces [JHS15]

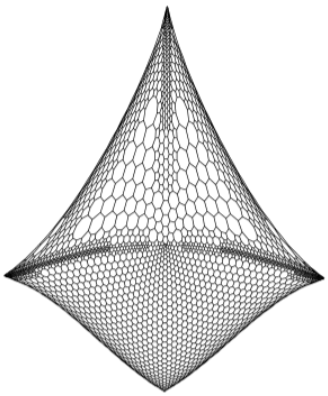


Figure 3.17: Agencement spectral [Kor15]

Dessin en grille

Les noeuds de ce dessin [Figure 3.15] sont comme alignées sur une grille imaginaire.

3.3.3 Algorithmes d'agencement de graphes

Dans cette partie, des algorithmes d'agencement de graphes en deux ou trois dimensions seront présentés:

- Agencement dirigé par des forces
- Agencement spectral
- Agencement orthogonal
- Agencement symétrique
- Agencement en arborescence
- Agencement en arc
- Agencement hiérarchique
- Agencement circulaire

Agencement dirigé par des forces

Cet agencement [Figure 3.16] est un système de forces basé sur un système de ressorts ou de la mécanique moléculaire et qui agit sur les noeuds et les arcs. Il combine des forces d'attractions entre les sommets adjacents avec des forces répulsives entre toutes les paires de sommets. Le but est de trouver l'état d'énergie minimum en simulant le système ou en résolvant une série d'équations différentielles.

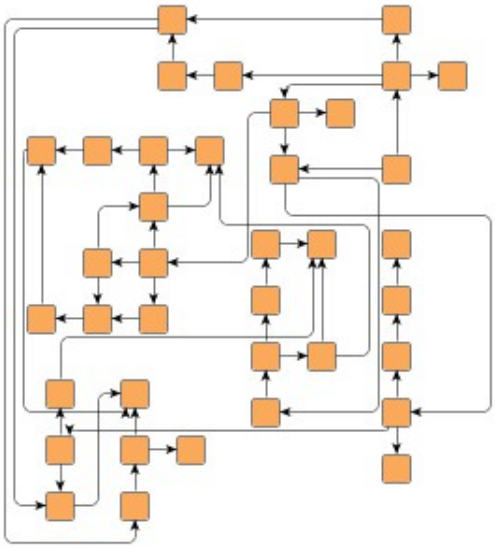


Figure 3.18: Agencement orthogonal [Ywo15a]

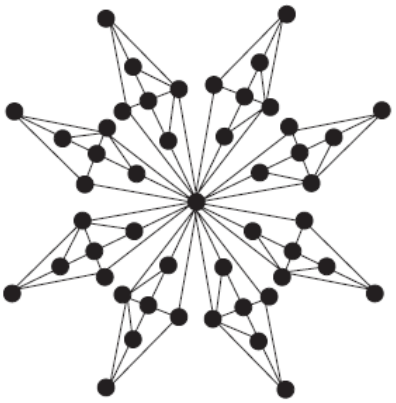


Figure 3.19: Agencement symétrique [HE15]

Agencement spectral

Cet arrangement [Figure 3.17] utilise les vecteurs propres d'une matrice, comme la matrice de Laplace du graphe, afin de trouver les coordonnées cartésiennes du graphe.

Agencement orthogonal

Cet agencement [Figure 3.18] joue uniquement sur des arcs horizontaux et verticaux. Il procède en 3 étapes:

1. Trouver les arcs qui se croisent
2. Calculer les limites du dessin
3. Déterminer les coordonnées

Des étapes peuvent être ajoutées pour optimiser et réduire la taille du dessin.

Agencement symétrique

Le graphe [Figure 3.19] est dessiné en faisant ressortir des éléments de symétrie entre des ensembles d'arcs et de sommets le composant.

Agencement en arborescence

Cet agencement [Figure 3.20] est surtout utile quand le graphe possède une racine, comme un arbre. Plusieurs approches [Rus15] sont possibles:

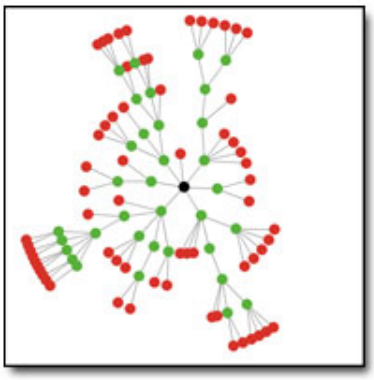


Figure 3.20: Agencement en arborescence ici circulaire [Bra15]

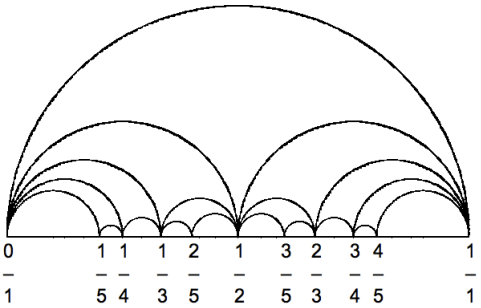


Figure 3.21: Agencement en arc [Hya15]

- Approche basée sur des niveaux: Les noeuds à la même distance de la racine sont alignés horizontalement
- Approche horizontale-verticale: Elle est basée sur une stratégie "diviser pour régner" et construit un arbre orthogonal
- Approche basée sur des chemins: Elle utilise un paradigme récursif sinueux suivant l'axe des x pour dessiner un arbre binaire, un arbre dans lequel chacun des noeuds a, au plus, 2 noeuds enfants)
- Approche circulaire: Chaque ensemble de noeuds de niveau inférieur à un sommet est placé sur un cercle dont le centre est le noeud d'origine
- Approche basée sur la séparation: Elle est basée sur une stratégie "diviser pour régner" et construit un graphe récursivement grâce aux étapes suivantes:
  1. Trouver un arc ou un noeud qui sépare le graph en sous-ensemble
  2. Séparer le graphe suivant l'arc ou le noeud venant d'être trouvé
  3. Assigner un ratio à chacun des sous-ensembles
  4. Dessiner les sous-ensembles en arbres
  5. Recomposer les dessins

Agencement en arc

Les sommets sont placés sur une ligne droite [Figure 3.21]. Les arcs sont ensuite ajoutés, représentés par des demi-cercles pour les sommets non-adjacents sur la ligne et par un segment de droite s'ils sont adjacents.

Agencement hiérarchique

L'agencement hiérarchique [Figure 3.22] est presque identique à celui en arborescence à l'exception qu'il autorise les cycles dans le graphe.

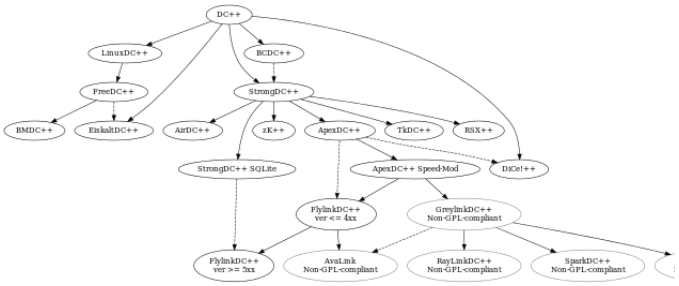


Figure 3.22: Agencement hiérarchique [Teh15]

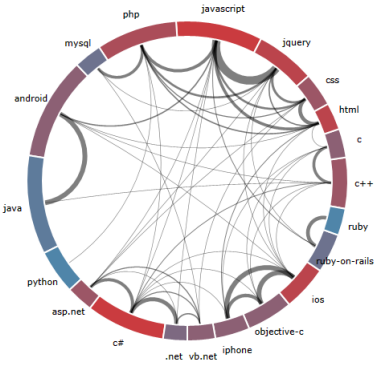


Figure 3.23: Agencement circulaire [Ebe15]

Agencement circulaire

Dans cet arrangement [Figure 3.23], les sommets sont placés sur un cercle où les arcs sont ensuite compris. Le graphe peut être séparé en sous-ensembles, chacun étant alors représenté par un agencement circulaire.

3.3.4 Avantages et inconvénients des agencements

Après cette présentation des différents agencements existants, voici leurs avantages et inconvénients:

Agencement	Avantages	Inconvénients
Agencement dirigé par des forces	• Flexibilité • Intuitivité • Simplicité	• Haute complexité de calcul • Résultat final peu optimisé
Agencement spectral	• Rapidité • Rendu optimal	• Manque d'intuitivité
Agencement orthogonal	• Adpaté aux grands graphes	• Pas plus de 4 degrés par noeud
Agencement symétrique	• Esthétique • Intuitivité	• Pas prévu pour les grands graphes • N'évite pas les intersections d'arcs
Agencement en arborescence	• Clareté • Intuitivité • Faible complexité	• Pas prévu pour les grands graphes
Agencement en arc	• Faible complexité	• Pas prévu pour les grands graphes
Agencement hiérarchique	• Réduit les croisements • Intuitivité	• Complexité élevée voire impossible pour les grands graphes
Agencement circulaire	• Faibles proportions • Pas de position privilégiée pour un noeud	• Difficilement compréhensible

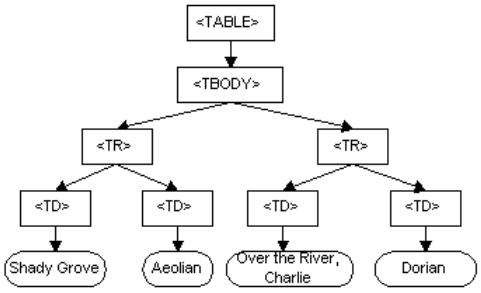


Figure 3.24: Rendu vertical [JR15]

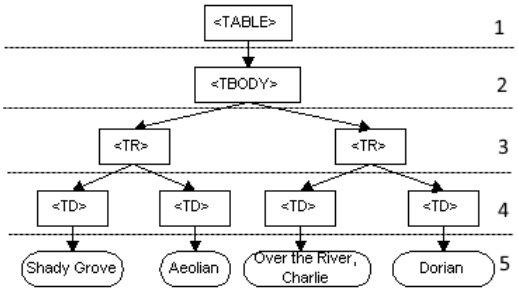


Figure 3.25: Rendu vertical avec des niveaux

3.4 Rendu final

Les rendus de graphes peuvent être classés selon plusieurs types :

- verticaux
- horizontaux
- circulaires
- multiples

3.4.1 Les rendus verticaux

Ils [Figure 3.24] représentent une hiérarchie verticale. Ils sont donc dirigés de bas en haut ou inversément. Ils possèdent une racine qui est la base du graphe.
Le rendu [Figure 3.25] peut être divisé par niveaux en fonction du parcours de chacun des nœuds jusqu'à la racine.

3.4.2 Les rendus horizontaux

Les rendus horizontaux [Figure 3.26] sont dessinés de gauche à droite ou inversément. Ils ont également une racine.
Tout comme le rendu vertical, ce type-ci peut être divisé en niveaux.

3.4.3 Les rendus circulaires

Les rendus circulaires [Figure 3.27] sont organisés autour d'un centre. Il peut faire partie du graphe auquel cas il est la racine de celui-ci et est divisible en niveaux. Si le centre ne fait pas partie du graphe, il est alors structuré autour du cercle, Les arcs sont alors inclus dans le cercle

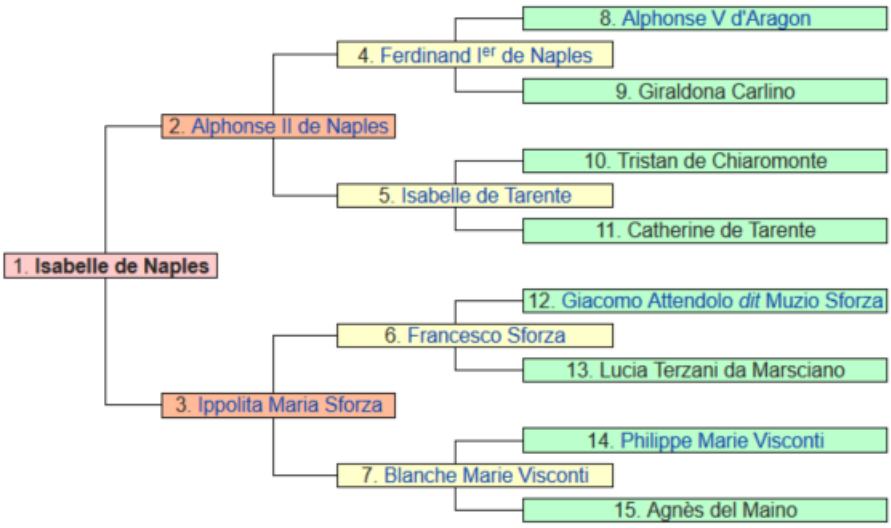


Figure 3.26: Rendu horizontal [Wik15c]

3.4.4 Avantages et inconvénients

L'usage des trois précédents rendus graphiques ne diffère véritablement que par le contexte, le support et la sémantique des informations à transmettre. Le choix du type de dessin est donc important.

Type de graphe	Avantages	Inconvénients
Vertical	Chaque niveau est visible. Autorise une grande profondeur dans la hiérarchie	Soit les noeuds comportent peu d'informations textuelles et sont nombreux sur un même niveau, soit c'est l'inverse pour des raisons de lisibilité du graphe
Horizontal	Chaque niveau est visible. Les noeuds peuvent contenir beaucoup d'informations textuelles	Peu de profondeur dans la hiérarchie
Circulaire	Afficher beaucoup de noeuds du dernier niveau sur une surface réduite. Possibilité de grouper les noeuds en famille	Pas de détails entre la racine et les noeuds du dernier niveau

Note : ces avantages et inconvénients sont pensés pour un affichage sur un écran de 19 pouces sans avoir recours au scrolling ou pour une impression sur une page de taille A3/A4 en mode portrait.

3.4.5 Les autres rendus

Le cas du graphe multiple est un peu plus complexe. Il est plus proche du nuage de points que d'une structure clairement hiérarchisée. Il ne dispose en effet pas d'une seule et unique racine, il peut en posséder aucune voire plusieurs. Les nœuds ne

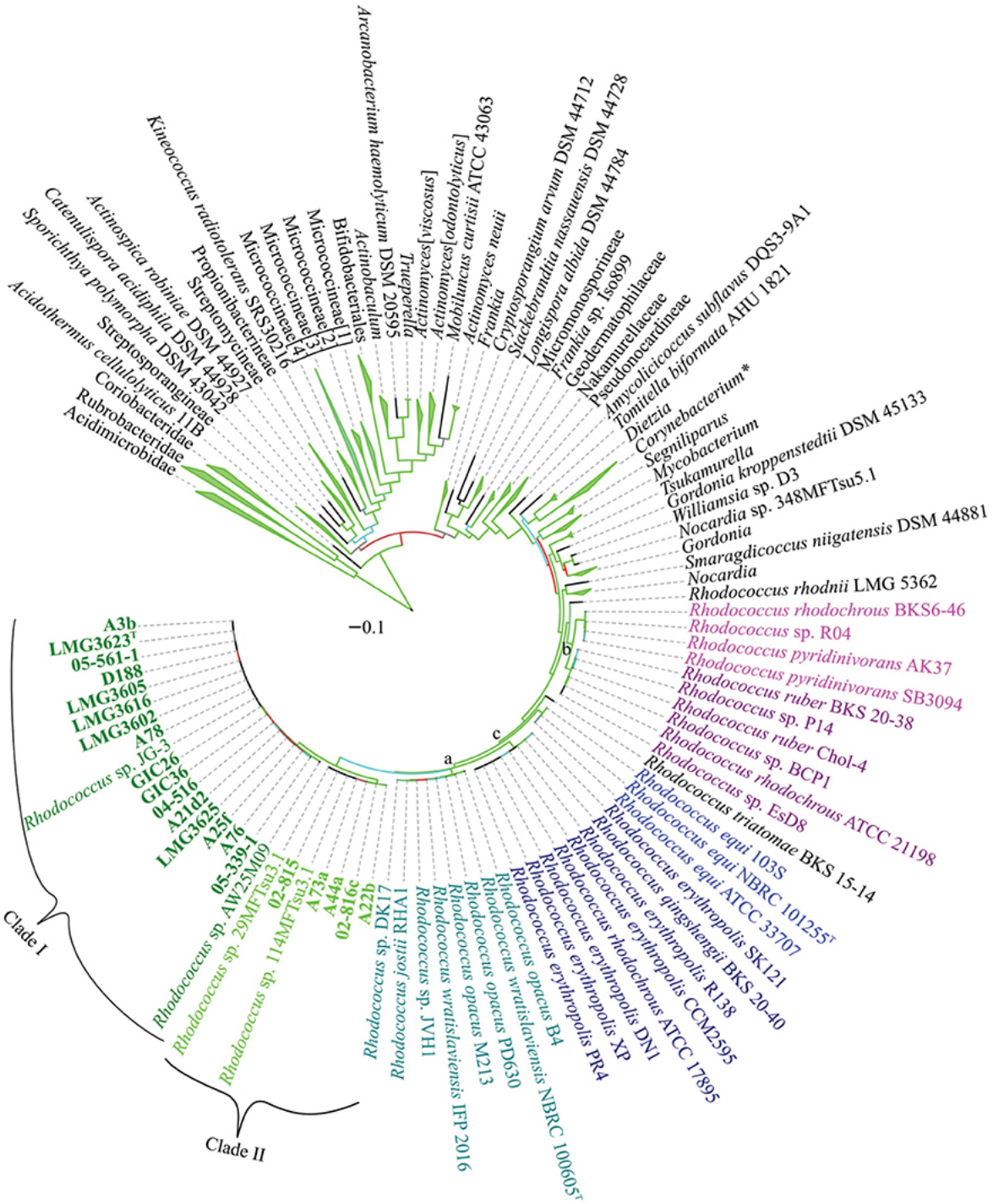


Figure 3.27: Rendu circulaire [GP15]

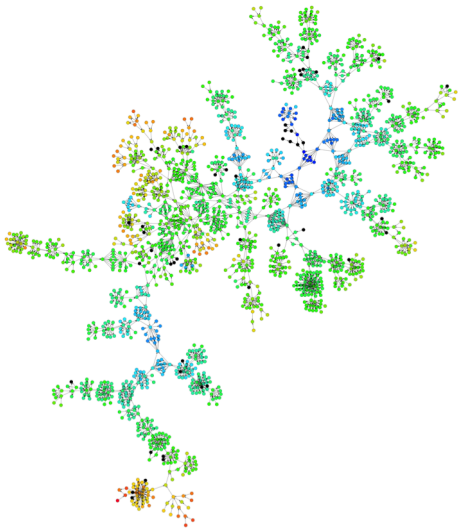


Figure 3.28: Rendu multiple [Zhu15]

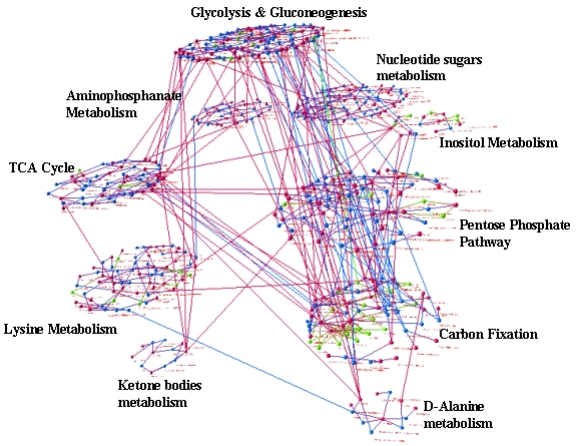


Figure 3.29: Autre rendu multiple [HH15]

peuvent pas être facilement classés par niveaux. Ce type de rendu est par contre plus facile à comprendre s'il est divisé en sous-ensembles, chacun reflétant une information particulière.

3.5 Données présentes sur le graphe

3.5.1 Les nœuds

Chaque nœud peut comporter des informations sur les données à l'origine du graphe. Elles sont plus ou moins parcellaires selon le niveau de détail voulu.
Elles se présentent sous forme de texte, symboles ou mots qui peuvent être placés dans ou autour du nœud [Figure 3.30].
La police est également importante dans la présentation des informations ainsi que la mise en forme des caractères (gras, souligné,...) pour indiquer les éléments essentiels du contenu du nœud [Figure 3.33].
La forme et la couleur des nœuds est aussi une manière de proposer de l'information [Figures 3.31 3.32].

3.5.2 Les arêtes

Les liens peuvent inclure des informations sur les relations entre les nœuds.
L'utilisation d'un arc ou d'une arête est déjà une information en soi : un arc [Figure 3.35] pour une relation hiérarchisante et une arête [Figure 3.34] pour une liaison égalitaire.
Une relation hiérarchisante peut également être représentée par une arête ayant des cardinalités [Figure 3.36].
Un texte [Figure 3.37] représentant le contexte ou la raison du lien peut lui être

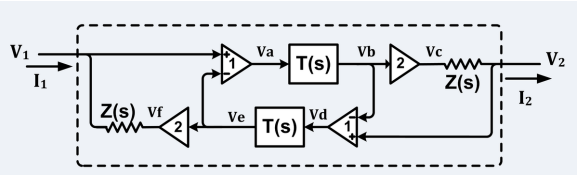


Figure 3.30: Informations contenues dans un nœud [Con15]

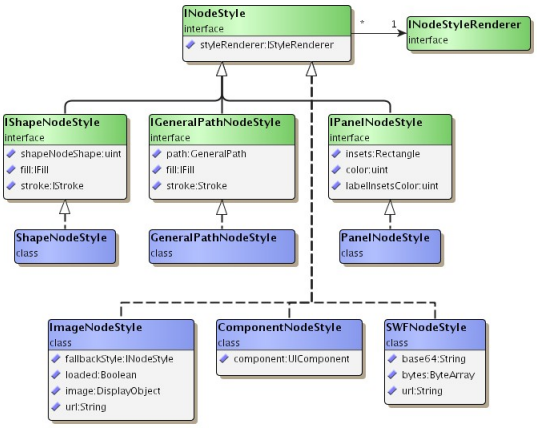


Figure 3.31: Format et couleur [Ywo15b]

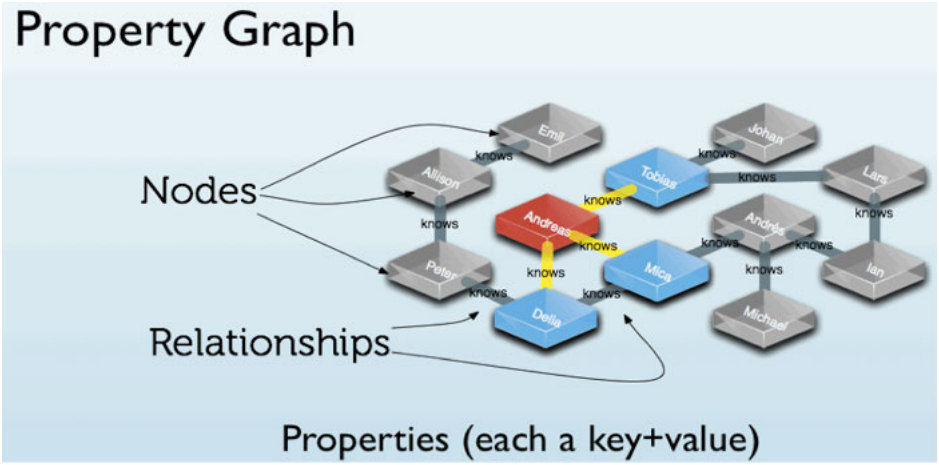


Figure 3.32: Format et couleur [Hun15]

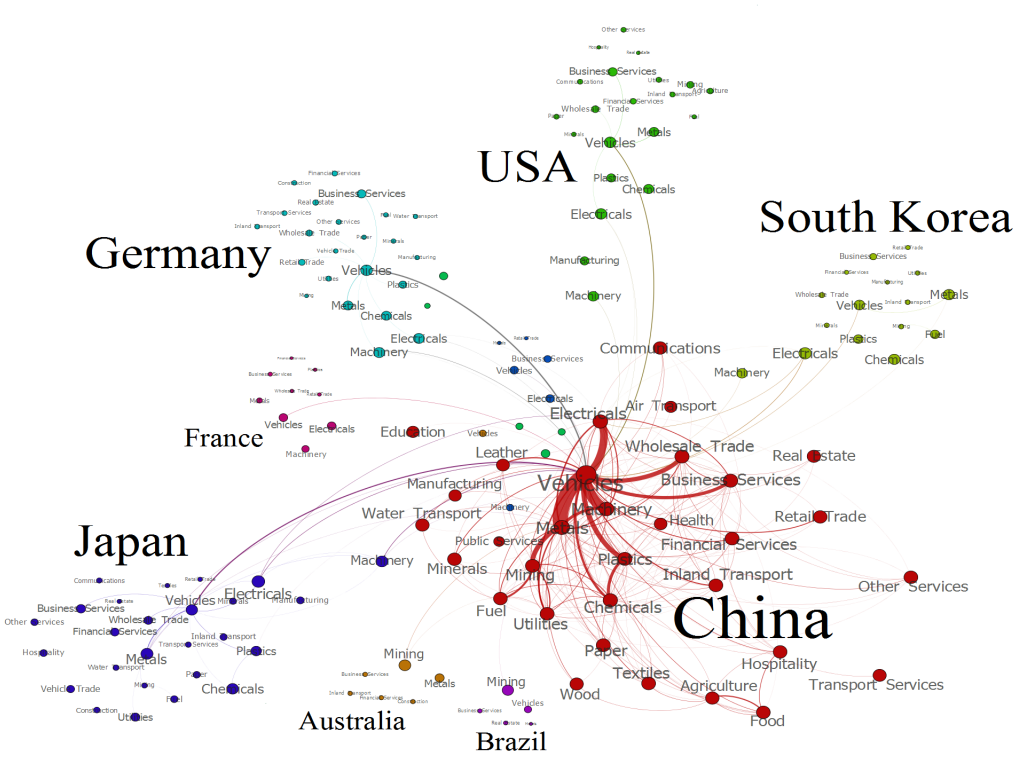


Figure 3.33: Police des noeuds [rob15]

assigné.
D'autres informations peuvent aussi être attribuées à des liens comme des ordres de grandeur, des poids, un ordre croissant, une valeur pour franchir le lien,...

3.6 Conclusion

Dans ce chapitre, nous avons pu constater que la visualisation d'un graphe dépend de plusieurs critères (par ordre d'importance) :

1. Les données à afficher
2. Le domaine d'utilisation
3. Le support cible
4. La sémantique
5. Le style de dessin

Chaque critère influence les suivants.
Le plus important reste, tout de même, d'avoir un rendu clair, lisible, compréhensible et complet.

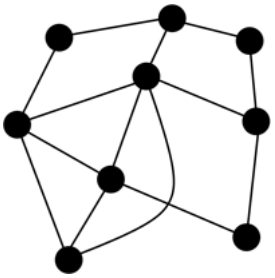


Figure 3.34: Exemple d'arête [Pan15]

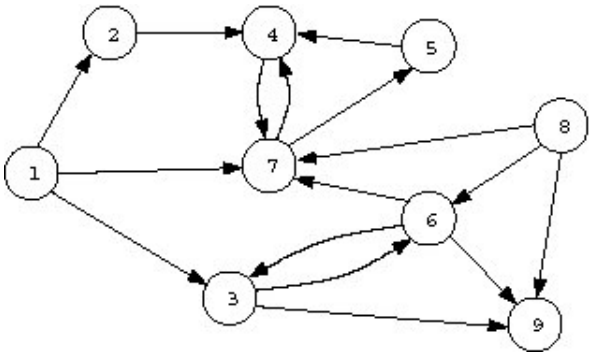


Figure 3.35: Exemple d'arc [Epi15]

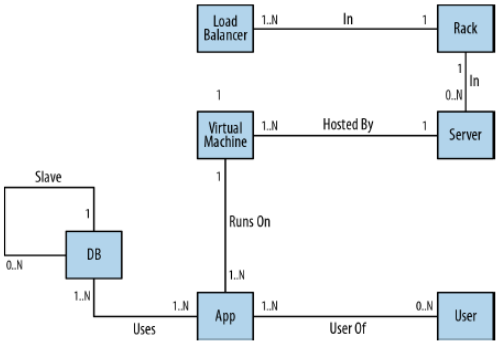


Figure 3.36: Cardinalités [IR15]

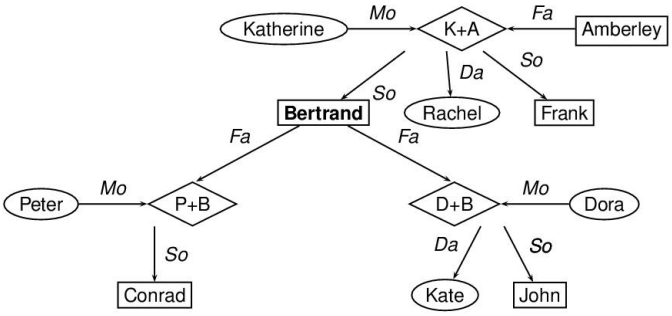


Figure 3.37: Texte sur un arc [ei15]

L'information doit être transmise sans ambigüité aux personnes qui le consultent.
L'algorithme sera par contre sélectionné selon ces critères par le programme.

Chapitre 4

État de l'art des outils de visualisation/design dédiés à la variabilité

Les outils de visualisation pour les modèles de variabilité sont des outils dédiés à afficher une représentation graphique des modèles de variabilité. De plus ces outils peuvent être combinés avec des éditeurs graphiques qui, eux, permettent l'édition du graphe directement dans la représentation graphique.

4.1 Technologie WYSIWYG

Un éditeur graphique est un logiciel qui permet de créer des interfaces, des documents, des graphiques et autres grâce à la technologie WYSIWYG.

Cette technologie, dont le nom vient d'un acronyme d'une locution anglaise signifiant « What you see is what you get », ou en français « vous voyez ce que vous réalisez », est un programme informatique dans lequel on voit directement ce que l'on est occupé de concevoir.

Comme exemples de programme utilisant cette technologie, on peut citer les programmes de traitement de texte, de création de maquettes, de dessin assisté par ordinateur, etc.

Dans le cas de la variabilité, l'utilisateur, grâce aux fonctionnalités du programme, pourra définir de manière visuelle la hiérarchie des types et leurs contraintes qui prendront place dans un graphe orienté, et les informations qu'il contiendra.

4.2 Types de représentations

Les représentations graphiques sont réalisées principalement de deux manières différentes :

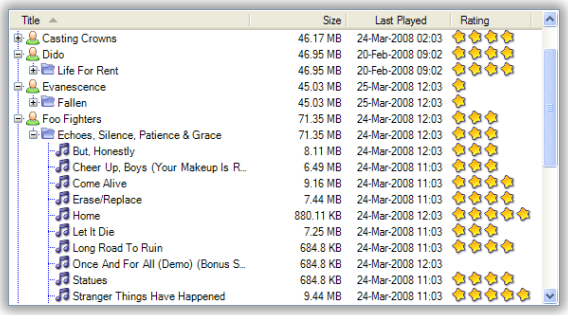


Figure 4.1: Treeview [Gra15]

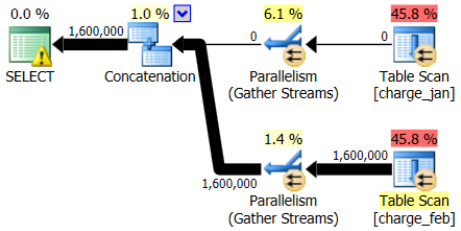


Figure 4.2: Graphe [Sac15]

- Treeview : le *treeview* [Figure 4.1] est un contrôle graphique servant à représenter une hiérarchie verticale entre des entités suivant des liens parent-enfant. Chaque entité est reproduite sous forme d'un nœud comportant un texte et parfois une icône.

  C'est la solution la plus courante pour la représentation des données. Pratiquement tous les langages de programmation disposant de bibliothèques contenant des contrôles graphiques en possèdent un de ce type. De plus il est relativement simple à utiliser.

- Graphe : le *graphe* [Figure 4.2] est une généralisation du treeview car il n'est qu'une représentation alignée à droite ou à gauche d'un graphe vertical. Le graphe offre par ailleurs les mêmes fonctionnalités en rajoutant, entre autres, le fait qu'un nœud peut posséder plus d'un parent, que les liens peuvent avoir des cardinalités différentes que 1-N, etc.

Certains logiciels combinent ces deux représentations avec l'une pour l'édition des propriétés des nœuds et l'autre pour la mise à jour de l'arborescence.

4.3 Catégories d'outils

Chaque outil peut être classé parmi une des catégories suivantes :

- Programme: l'outil est un programme complet et utilisable sans autre logiciel
- Plugin: l'outil est une extension pour un programme existant
- Application web: l'outil est une application disponible depuis internet

Chaque catégorie a ses avantages et ses inconvénients. Ils sont repris dans le tableau suivant :

	Avantages	Inconvénients
Programme	• Ne requiert aucun ou très peu de prérequis (framework) • Contrôle des fonctionnalités du programme	• Nécessite un grand investissement dans le développement de l'outil (fonctionnalités supplémentaires comme l'édition et sauvegarde, portabilité) • Besoin de tester la compatibilité du programme avec différents systèmes d'exploitation et leurs versions
Plugin	• Profite des fonctionnalités du logiciel pour lequel il est développé	• Besoin de suivre les évolutions et le cycle de vie du logiciel pour lequel il est développé
Application web	• Facilité de mise à jour et de déploiement • Facilité de test pour l'utilisateur final • Accessibilité de l'outil	• Sensiblité aux probèmes d'internet (sécurité, disponibilité, ...) • Nécessite un grand investissement dans le développement de l'outil (fonctionnalités supplémentaires comme l'édition et sauvegarde, portabilité)

Voici quelques exemples d'outils dédiés à la variabilité:

- Programmes: MetaEdit+ Modeler, HyperSenses
- Plugins: Xfeature, Feature Modeling, Pure::Variants, FeatureMapper
- Application web: SPLOT, VMC

4.4 Outils existants

Afin d'apporter un avis global sur les outils existants, une batterie de tests a été réalisée sur les programmes suivants:

- Feature IDE http://wwwiti.cs.uni-magdeburg.de/iti_b/research/featureide/\#documentation
- Feature mapper http://featuremapper.org/
- Captain feature http://sourceforge.net/projects/captainfeature/?source=navbar
- VMC http://ercim-news.ercim.eu/en93/ri/vmc-a-tool-for-the-analysis-of-variabilit
- Purevariants http://www.pure-systems.com/Variant_Management.49+M54a708de802.0.html
- HyperSense: http://www.d-s-t-g.com/fr/produits-head/hypersenses.html
- MetaEdit+ http://www.metacase.com/mep/
- Xfeature http://www.pnp-software.com/XFeature/Home.html
- Splot http://www.splot-research.org/
- Feature modeling http://sourceforge.net/projects/fmp/

4.4.1 Scénarios de tests

Pour estimer les performances des outils, différents scénarios ont été reproduit avec chacun d'eux.

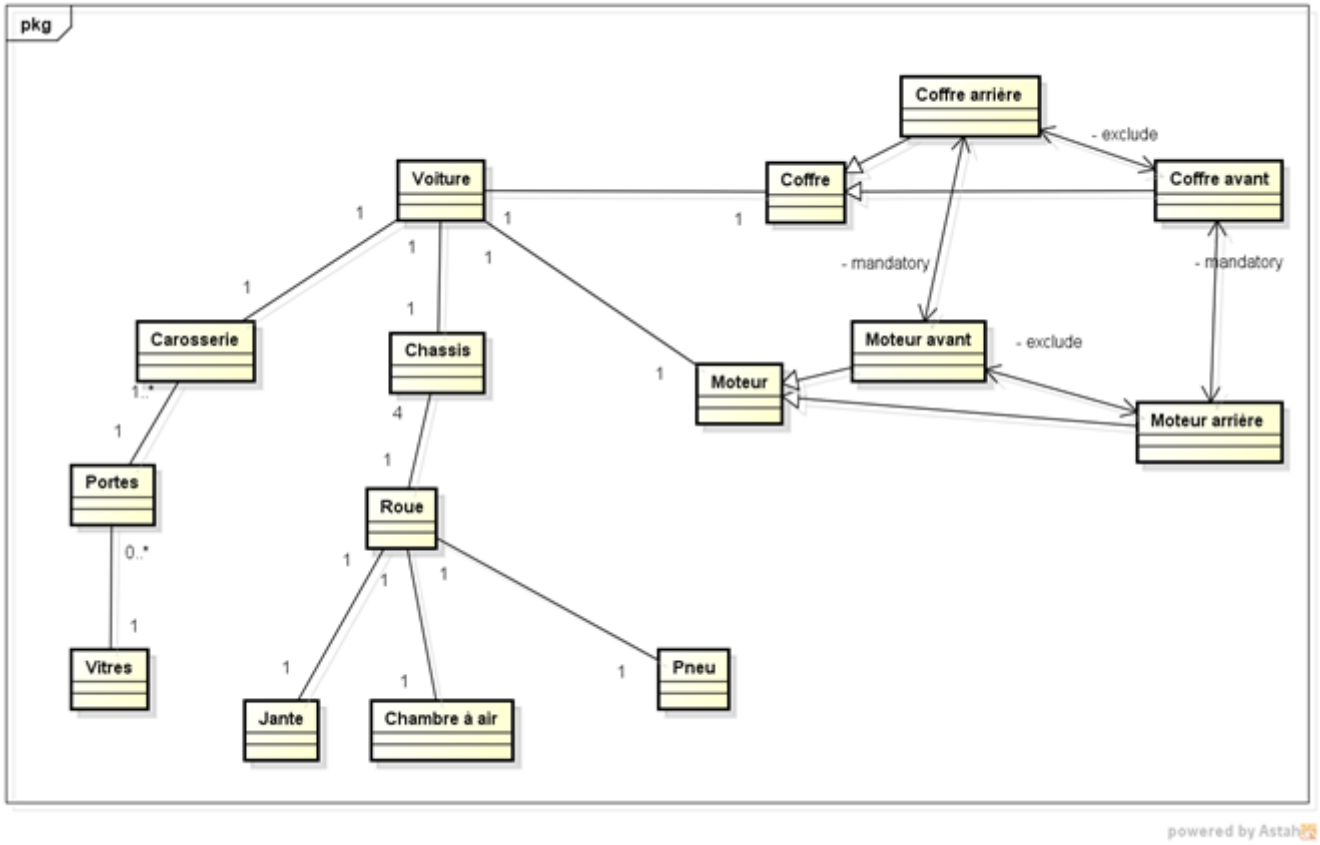
Scénario 1: Installation

1. Installation du programme
2. Premier lancement

Scénario 2: Création d'un projet

1. Création du projet
2. Sauvegarde
3. Ajout d'un noeud A
4. Renommer le noeud
5. Ajout d'un noeud B
6. Ajout d'une arête
7. Transformer l'arête en arc (A vers B)
8. Nommer l'arc
9. Sauvegarder

Scénario 3: Reproduire le schéma ci-dessous



Scénario 4: Ressources utilisées

1. Lancer le programme
2. Vérifier sa consommation mémoire
3. Charger le schéma fait en 3
4. Revérifier la consommation mémoire

Scénario 5: Fonctionnalités externes aux processus métiers

Vérifier si le programme offre :

1. Une fonctionnalité faire-défaire pour plusieurs opérations contiguës
2. Une fonctionnalité de zoom
3. Une fonctionnalité de recherche sur le schéma
4. Une fonctionnalité d'export du schéma sous forme d'image
5. Une fonctionnalité de sauvegarde du schéma

6. Une fonctionnalité d'ouverture d'un schéma sauvegardé interne au programme
7. Une fonctionnalité d'ouverture d'un schéma sauvegardé externe au programme
8. Une fonctionnalité de couper/copier/coller

Autres questions

1. Le programme propose-t-il des extensions par plugin?
2. Le programme est-il portable sur d'autres plateformes (linux,...)?
3. Les fichiers sauvegardés sont-ils utilisables pour d'autres programmes?
4. Le programme est-il encore supporté par une équipe? Bugfixing? Nouveaux développements?
5. Depuis combien de temps le programme existe-t-il?
6. Les fonctionnalités du programme comportent-elles encore des bugs?
7. Est-ce qu'une opération non appropriée/autorisée fait fermer le programme?
8. Le travail d'une session est-il récupérable après une fermeture non prévue du programme?
9. L'apprentissage est-il aisé?
10. L'interface homme-machine est-elle facilement compréhensible?
11. Est-elle attractive?
12. Existe-il une version de production?
13. Le programme propose-t-il des fonctions de décorations du schéma (couleur, type de noeuds, type de traits)?
14. Quelles sont les vues proposées par le programme (graphe, treeview,...)?
15. Est-il possible de rajouter un attribut/propriété dans le modèle?
16. L'ajout des contraintes est-il possible?

4.5 Résultats

Pour ne pas alourdir la présentation de ce mémoire, les résultats bruts ont été renvoyés en annexe.

Voici une synthèse des résultats obtenus:

	Scénario 1	Scénario 2	Scénario 3	Scénario 4	Scénario 5
Captain Feature	Plugin d'Eclipse, pas de problème rencontré	Ok mais on ne crée que des arcs	Ok mais pas de support des contraintes	86mb	Pas de zoom, de faire-défaire, d'ouverture de fichier, de copier-coller, ni de recherche
Feature Mapper	Plugin d'Eclipse, pas de problème rencontré	Ok mais l'affichage en treeview ne facilite pas les choses	Long car navigation pas optimale, difficulté à comprendre les contraintes ajoutées	Pas d'impact sur la mémoire	Pas de zoom, pas d'export, ni de recherche; attention à la sauvegarde et aux accents
Feature Modeling	Plugin d'Eclipse, pas de problème rencontré	Ok mais pas d'arcs visibles	Ok, pas de problème rencontré	Pas d'impact sur la mémoire	Pas d'export, ni de copier-coller; Les fonctionnalités d'ouverture et de recherche sont fournies par eclipse
Feature IDE	Plugin d'Eclipse, pas de problème rencontré	Très bon outil, tutoriel existant; Attention, noms des noeuds uniques	Pas de problème lors de la construction, contraintes aisément ajoutables mais pas de cardinalités	Pas d'impact sur la mémoire	Pas de copier-coller, le reste est implémenté
Hypersense	Impossible car pas de licence	Non réalisé	Non réalisé	Non réalisé	Non réalisé

	Scénario 1	Scénario 2	Scénario 3	Scénario 4	Scénario 5
MetaEdit	Programme complet, pas de problème rencontré	Trop complet donc on s'y perd, impossible de nommer des arcs	Schéma créé mais impossible d'ajouter des contraintes	36 Mb	Pas de recherche ni d'import
PureVariants	Programme complet (c'est une version modifiée d'Eclipse), pas de problème rencontré, ne fonctionne qu'avec un os 32 bits	Pas de problème mais pas d'arcs, que des arêtes	Pas de problème mais contraintes en prolog ou pvscl, pas de cardinalités	Pas d'impact	Pas de zoom, pas d'export pour la version gratuite; Attention au copier-coller qui duplique également toute la hiérarchie qui suit le noeud sélectionné
SPLOT	Application web, pas d'installation	Pas de problème mais pas de nommage d'arc, arcs à orientation unique	Pas de problème mais pas de cardinalités spécifiques	Non applicable	Pas de zoom, de faire-défaire, d'ouverture de fichier, de copier-coller, ni de recherche
VMC	Application web, pas d'installation	Ca n'est pas un éditeur WYSIWYG, on passe par la création de code avant d'avoir un rendu graphique	Impossible de créer le schéma car on dessine un diagramme d'état	Non applicable	Fonctionnalités de faire-défaire et de copier-coller sur le code; Pas d'import; Export possible en sauvegardant l'image depuis le navigateur; Les fonctionnalités de zoom et de recherche sont apportées par le navigateur
XFeature	Installation trop longue	Non réalisé	Non réalisé	Non réalisé	Non réalisé

4.6 Enseignements

Après avoir synthétiser les résultats, Les enseignements que l'on peut en retirer sont listés dans cette partie. Les enseignements seront triés par ordre d'importance dans le chapitre suivant.

4.6.1 Installation

Comme décrit au point 4.3, il existe 3 types de programmes, dont 2 seuls sont installables sur un pc local.

Plugin

Le plugin est dépendant d'un autre programme dont il faut avoir déjà installé une version compatible. Après, le plugin peut être ajouté via un gestionnaire interne ou en copiant les fichiers nécessaires dans le répertoire dédié aux plugins.
Il va sans dire que passer par un gestionnaire est beaucoup plus pratique que la copie des fichiers pour plusieurs raisons:

- Facilité de prise en main du gestionnaire
- Résolution automatique des dépendances avec d'autres librairies et plugins
- Validation de la licence d'utilisation
- Accès centralisé à la bibliothèque des plugins disponibles pour ce logiciel

Le plugin nécessite la plupart du temps un redémarrage du programme principal pour devenir actif.

Programme

Le programme s'installe seul, le plus souvent à partir d'un fichier exécutable (.exe) et/ou extractable (de type zip). Il n'y a donc pas de manipulations compliquées à réaliser. Il faut juste prêter attention aux applications indésirables de type adware qui peuvent venir avec l'installation de l'application.

Dans les explications détaillant la procédure d'installation, il est également important de notifier les prérequis nécessaires et les systèmes d'exploitation avec lequels le programme est compatible (testé et validé).
Dans tous les cas, un tutoriel pour guider l'utilisateur est un plus surtout s'il n'est pas familiarisé avec ces procédures.

Application web

La question de l'installation ne se pose pas pour une application disponible depuis internet. Il faudra tout au plus s'inscrire sur le site web l'hébergeant pour en profiter.

4.6.2 Interface Homme-Machine

L'interface se compose d'un cadre global pouvant être un navigateur internet, un programme principal comme Eclipse ou le programme entièrement créé.
Dans le cas présent, l'interface décrite est celle d'une application installée sur un ordinateur.
Elle contient une fenêtre principale, dans laquelle est représenté le graphe, et est accompagnée de fenêtres secondaires de plus petites tailles qui :

- soit affichent le graphe ou une partie sous une autre forme comme par exemple les suivants ou prédecesseurs du noeud sélectionné
- soit proposent une liste de propriétés associées à l'élément sélectionné à des fins de consultation et de modification
- soit offrent des outils permettant d'ajouter des nouveaux composants au graphe

Qu'elles soient intégrées à un outil existant ou faisant parti d'un nouveau programme, toutes les fenêtres bénéficient des fonctionnalités de base des composants de type fenêtre (redimensionnables, mobiles...).

Différentes fonctionnalités sont rendues facilement accessibles via une barre d'outils disposée en haut du cadre global ou d'une autre fenêtre et/ou de menus contextuels ouverts via un clic-droit depuis la souris ou d'une pression sur une touche appropriée.

Si le concepteur du plugin a respecté la charte graphique et la logique du programme principal, l'intégration dans un logiciel existant apporte comme avantage que l'utilisateur peut garder ses habitudes de manipulation des fonctionnalités déjà existantes (raccourcis, emplacement des fenêtres...).

Dans le cas d'un outil web, à l'origine, certaines limites apparaissent du point de vue de l'interface dues à la nature "sans état" (stateless) du protocole http.
En effet à chaque rechargement de la page, tous les éléments non sauvegardés sont perdus (modification de la position de fenêtres, des outils affichés) et l'interface revient à son état d'origine. Mais vu qu'avec l'avènement du Web 2.0, le rechargement de page n'est plus nécessaire, il est donc possible de concevoir des outils de modélisation entièrement en ligne possédant les mêmes possibilités que les programmes installables.

4.6.3 Prise en main

Lors des premières utilisations d'un nouveau logiciel, la prise en main est un élément important. En effet la courbe d'apprentissage qui doit mener d'un niveau novice à un

niveau intermédiaire voire expert doit être bien dosée et progressive.

Pour ce faire, la compréhension du fonctionnement de l'application et de l'interface doit être la plus intuitive possible.
Elle doit correspondre aux usages courants de logiciels existants (raccourcis, menus et leurs organisations, fonctions globales comme ouverture d'un fichier, etc).
Les termes définissant les différentes fonctionnalités doivent être choisis de manière à décrire précisément l'action et l'impact induit par celle-ci sur le schéma.

Si une fonctionnalité mérite plus d'explications afin d'être utilisée correctement, une aide incluse au programme peut être ouverte par l'utilisateur pour répondre à ses questions.
Un tutoriel écrit ou sous format vidéo ou encore une aide en ligne peut aussi remplir ce rôle et favoriser l'apprentissage.

En cas d'erreur ou d'emploi incorrect d'une fonctionnalité, un message devant aider l'utilisateur doit être affiché afin qu'il comprenne où il l'a commise, comment la résoudre et ne pas la reproduire.

4.6.4 Fonctionnalité principale: La construction de graphes

La partie graphique du dessin de graphes doit proposer des outils modulaires, adéquats et rapides d'accès. Cela peut être apporté grâce à des menus contextuels disponibles depuis un raccourci ou un clic de souris ou des barres d'outils comme expliqué précédemment.

Chaque opération sur le graphe doit être retranscrite presque instantanément. La performance et la rapidité de calcul sont donc importantes afin de ne pas laisser dubitatif l'utilisateur quant au résultat obtenu.

Ce point se recoupe avec la prise en main car le dessin d'un graphe est la fonctionnalité principale du programme. Elle doit donc avoir la meilleure prise en main possible et être très intuitive.

Comme expliqué dans un paragraphe précédent, afin de faciliter le dessin des graphes, un tutoriel compris dans le logiciel ou sur un autre support en ligne est un excellent ajout pour expliquer le mode d'utilisation de cette fonctionnalité.

4.6.5 Fonctionnalités externes aux processus métier

A côté du dessin de graphes, une série d'autres fonctionnalités enrichissent le logiciel.

Un copier/couper et coller peut être ajouté pour accélerer la construction du graphe en dupliquant les noeuds.

Un zoom pour agrandir une partie du graphe ou avoir une vue globale peut compléter l'application.

Le programme peut aussi offrir un export du graphe dans un autre format, comme image ou pdf.

Bien sûr, toutes ces fonctionnalités dépendent de la plateforme à partir de laquelle l'application est accessible.

4.6.6 Fonctionnalités avancées

Les utilisateurs avec un niveau intermédiaire ou expert apprécient de pouvoir automatiser certains groupes de fonctionnalités grâce au scripting ou aux macros.

Afin d'être plus productif et plus rapide, le graphe doit pouvoir être éditable depuis le fichier de sauvegarde disponible sous format texte ou xml. Cette demande implique une vérification du contenu du fichier lors de l'ouverture de celui-ci pour vérifier qu'il ne contient pas d'erreur.

4.6.7 Support

L'application doit être soutenue par la société ou la personne responsable pour encourager le public à l'utiliser.
Afin de remonter les différents bugs qui seront trouvés dans le logiciel et des propositions d'amélioration, un moyen simple et efficace doit être proposé aux utilisateurs, comme un forum ou mieux un logiciel dans lequel on peut reporter des bugs.

L'application doit également avoir un suivi sous forme de mises à jour afin de corriger les bugs remontés et d'apporter de nouvelles fonctionnalités.

4.7 Conclusions

Dans ce chapitre, nous avons pu spécifier ce qu'est la technologie WYSIWYG, soit une visualisation directe d'actions sur une interface graphique. Elle permet d'éviter de passer par des opérations spécifiques afin de regénérer cette interface avec les résultats de nos opérations.

Nous avons aussi constaté les différents types de représentations possibles dans les outils existant à savoir le graphe et le treeview, le premier étant une généralisation du second. Leur utilisation dépend des choix de conception de chaque outil.

Pour finir, nous avons utilisé différentes catégories d'outils que sont les applications web, plugins et logiciels complets dans le cadre de la variabilité. Chacun a ses avantages et inconvénients que nous avons pu observés durant les tests qui ont été réalisés sur base d'un questionnaire.

La création en WYSIWYG de modèles de variabilités pourrait paraitre utopique sur les applications web mais le développement de nouvelles technologies changera la donne avec l'émergence d'applications riches accessibles directement depuis internet.

Chapitre 5

Manquements et fonctionnalités essentielles des outils de visualisation dédiés à la variabilité

Dans ce chapitre, nous réaliserons une combinaison des deux chapitre précédents.
Nous analyserons les résultats des tests pour en dégager les fonctionnalités essentielles nécessaires à un outil dédié à la variabilité.
Ensuite, nous comparerons les possibilités offertes par la visualisation de graphes et les diverses caractéristiques de chaque outil afin de déterminer quels sont les aspects manquants.

5.1 Fonctionnalités essentielles

Dans chaque logiciel, certaines fonctionnalités sont plus importantes que d'autres, bien que toutes font la richesse du logiciel et amènent le fait que l'utilisateur vient se resservir du programme.

Voici celles qui ont été spécifiées comme essentielles au logiciel par ordre décroissant d'importance:

1. Afficher le graphe orienté
2. Ajouter des noeuds ou sommets au graphe
3. Sauvegarder dans un fichier et ouverture depuis ce fichier
4. Editer des propriétés des éléments du graphe
5. Supprimer des éléments
6. Déplacer tout ou une partie du graphe
7. Exporter le graphe dans une image

8. Naviguer dans le graphe
9. Créer un nouveau graphe
10. Ajouter des contraintes

5.1.1 Afficher le graphe orienté

Une des fonctionnalités principales, l'affichage du graphe, sert à visualiser le dessin représentant des lignes de code TVL. C'est la fonctionnalité essentielle car c'est la seule sans qui le logiciel n'a aucun intérêt.

5.1.2 Ajouter des noeuds ou sommets au graphe

Une fois que le graphe est affiché, ajouter des éléments à celui-ci est la suite logique pour étendre le graphe.

5.1.3 Sauvegarder dans un fichier et ouverture depuis ce fichier

Sans cette fonctionnalité, le logiciel n'a pas grand intérêt. La création de graphes peut toujours être accomplie sans, mais toutes les sessions de travail seront indépendantes. Cette fonctionnalité permet donc la pérénité des travaux.

5.1.4 Editer des propriétés des éléments du graphe

Ajouter des informations sur les noeuds et arcs permet d'enrichir le graphe et de le placer dans un contexte particulier. Cette fonctionnalité se recoupe avec TVL car les noms des noeuds définiront les types dans les lignes de code.

5.1.5 Supprimer des éléments

C'est le pendant de la fonctionnalité "Ajouter des éléments", elle permet de retirer du graphe les éléments inutiles.

5.1.6 Déplacer tout ou une partie du graphe

Afin de pouvoir réorganiser le graphe ou de replacer les éléments mal placés, il faut pouvoir bouger les parties et les repositionner correctemet. Cette fonctionalité permet de refondre le graphe pour le rendre plus compréhensible.

5.1.7 Exporter le graphe dans une image

Dès que le graphe est construit, l'exporter permet de distribuer le rendu à d'autres personnes afin de partager les informations qu'il transmet.

5.1.8 Naviguer dans le graphe

Afin de s'y retrouver entre les noeuds et autres éléments du graphe ainsi que de survoler le graphe rapidement, le déroulement de l'écran doit être facile. Il peut être représenté par des barres de déroulement ou encore par le glissement du graphe. On peut également ajouter une fonction de recherche entre les intitulés des noeuds.

5.1.9 Créer un nouveau graphe

Cette fonctionnalité sert à créer un nouveau graphe et donc un nouveau feature model.

5.1.10 Support des contraintes

TVL supporte les contraintes. Le graphe ne serait pas complet sans le support de cette fonctionnalité.

5.2 Manquements détectés dans les outils

Durant les tests sur les différents outils, certains manquements aux logiciels existants ont été relevés.

5.2.1 Support des attributs

Dans TVL, chaque entité peut avoir des attributs de type *boolean, int, real*. Une fonctionnalité, pour assigner des variables de ces types à un noeud, permettra de prendre en compte cette partie de TVL.

5.2.2 Automatisation de l'optimisation

Quand le graphe aura atteint une certaine complexité, un certain nombre d'arcs se croiseront, rajoutant encore de la difficulté à le comprendre. Proposer une fonctionnalité qui permette de réagencer le graphe pour limiter le nombre d'intersections de noeuds ou d'arc serait une addition intéressante. C'est un ajout à double sens car si le concepteur du graphe a déjà regroupé certains noeuds selon une logique, le réagencement risque fortement de casser cette organisation.

5.2.3 Pouvoir changer le style de dessin

Plusieurs styles de dessins ont été présentés dans la partie 3.3. Pouvoir changer de dessin permet de donner un autre but au graphe, d'avoir un autre point de vue sur les données contenues. Cette fonctionnalité complète un peu la précédente. En effet, le changement de graphe conduira à une réorganisation des noeuds dépendant du style choisi.

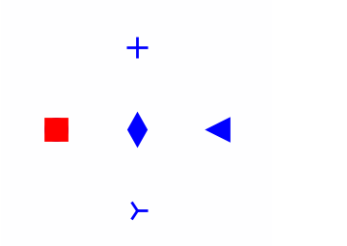


Figure 5.1: Style de noeuds

5.2.4 Introduire la notion de sous-graphe avec zoom et nouvel onglet

Lors de la création de feature model important, la lisibilité du graphe peut devenir très compliquée. Le but de cette fonctionnalité est de permettre de mettre un focus sur une partie relativement petite du graphe afin de pouvoir la travailler sans être dérangé par la complexité et la taille du graphe. Le sous-graphe serait affiché dans un autre onglet, concept qui permet d'avoir, dans une seule fenêtre, plusieurs interfaces différentes et complémentaires. Toute modification du sous-graphe serait répercutée sur le graphe principal.

5.2.5 Styler les noeuds ou les arcs

Dans une volonté d'apporter plus d'informations sur les noeuds ou les arcs, au lieu de les laisser tous de la même forme, il est intéressant de pouvoir changer l'aspect de certains éléments [Figure 5.1]. Cette opportunité peut rejoindre les sous-graphes car ils pourraient être générés sur base de ce critère.

5.2.6 Filtre sur le graphe

En plus des sous-graphes et des styles de noeuds, d'autres filtres peuvent aussi s'appliquer sur des conditions différentes comme des propriétés ou autres valeurs. C'est encore une fois pour faciliter la lecture du graphe et diminuer la complexité qu'il peut engendrer.

5.3 Conclusions

Dans ce chapitre, les fonctionnalités essentielles sont celles liées à la réalisation du graphe et ses relations avec le language TVL. C'est normal étant donné que la construction du graphe est la fonction de base du logiciel. De plus, les autres fonctionnalités retenues sont celles qui enrichissent l'application et qui permettent aux utilisateurs de pouvoir profiter pleinement des graphes créés.

Pour les manquements, ce sont des aspects qui permettent de simplifier l'utilisation du logiciel lors de la création de graphes comprennant de nombreux noeuds et arcs. Un autre aspect permet de transmettre l'information différemment aux différentes parties

prenantes pour mettre en exergue des caractéristiques spécifiques du domaine étudié.

Chapitre 6

Vers un nouvel outil dédié à la variabilité

Dans ce chapitre, nous allons aborder les spécifications des fonctionnalités à apporter à un nouvel outil dédié à la variabilité, et plus particulièrement à TVL. Après avoir également proposé un concept d'interface, nous suivrons le développement d'un prototype.

6.1 Fonctionnalités retenues

Pour ce nouvel outil, les fonctionnalités retenues sont les fonctionnalités essentielles (section 5.1), les manquements (section 5.2) et certains autres aspects non encore abordés.

Le diagramme de cas d'utilisations [Figure 6.1] résume les fonctionnalités à ajouter à l'outil. Dans ce schéma, les fonctionnalités en bleu sont celles essentielles, celles en vert sont les manquements relevées et celles en jaune, les autres aspects. Seuls ces derniers seront expliqués dans cette partie.

6.1.1 Couper/Copier/Coller un noeud

Pour accélerer la création de noeuds, surtout ceux qui possèdent des propriétés particulières, il est intéressant de pouvoir les dupliquer rapidement grâce une fonctionnalité du type "copier-coller".

6.1.2 Zoomer dans le graphe

Si le graphe devient trop encombré ou trop étalé, il devient intéressant de pouvoir zoomer, que ce soit pour avoir plus de détails et mettre le focus sur une partie de celui-ci ou prendre du recul pour avoir une vue d'ensemble.

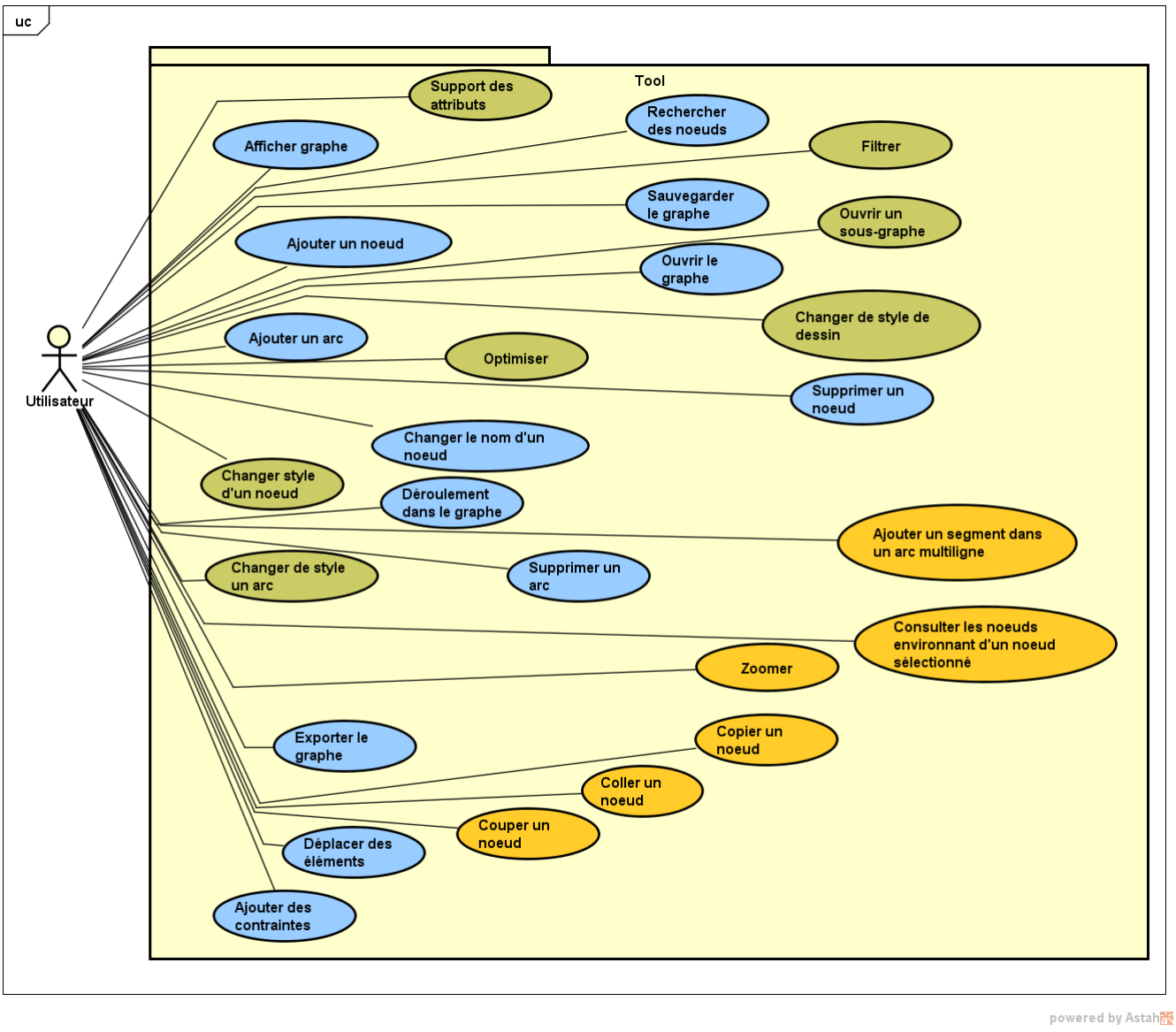


Figure 6.1: Cas d'utilisations

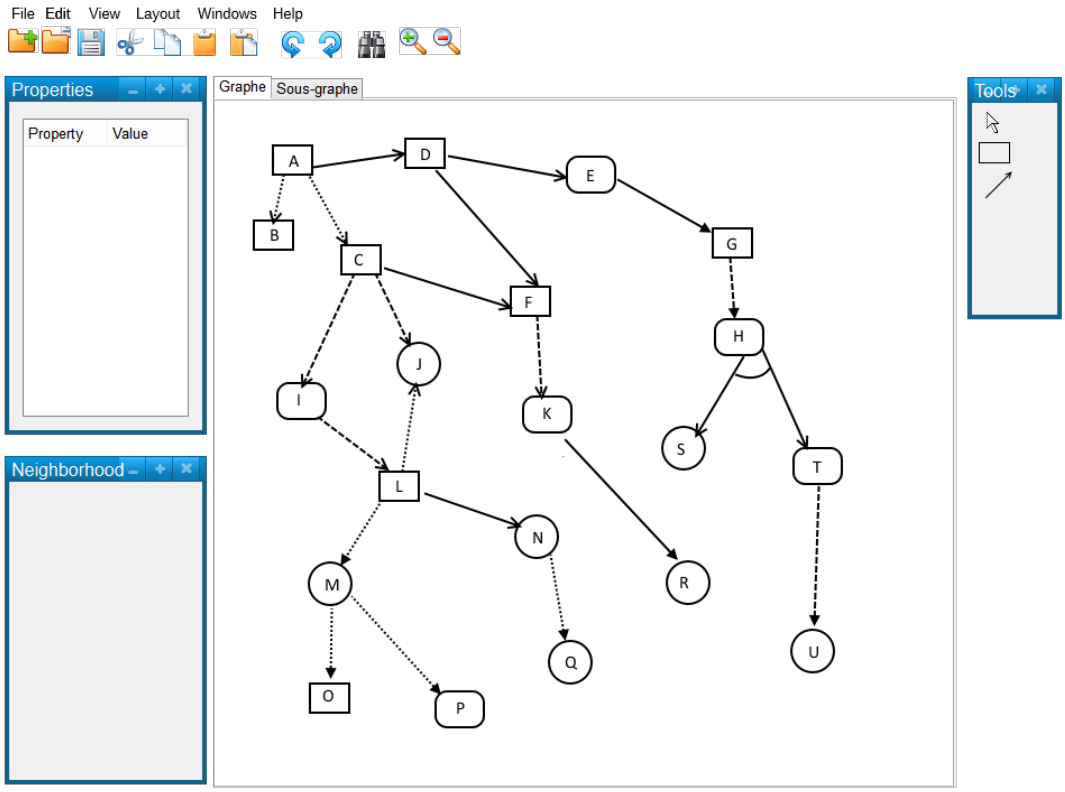


Figure 6.2: Proposition d'écran

6.1.3 Ajouter un segment dans un arc multiligne

Lors de l'ajout d'un arc, celui-ci est représenté par une ligne droite. Pour des raisons de lisibilité, il peut être profitable de casser cette ligne droite en segments contigües mais non rectilignes. Le diagramme des cas d'utilisations [Figure 6.1] comporte des exemples clairs.

6.1.4 Consulter les noeuds environnant d'un noeud sélectionné

Pour des raisons de facilité de lecture, un sous-graphe comportant uniquement les noeuds liés par un arc au noeud sélectionné peut être affiché dans une fenêtre secondaire.

6.2 Proposition d'écran

Pour le nouvel outil, voici une proposition d'écran référencé à la figure 6.2.

Dans cet écran, plusieurs parties peuvent être aperçues:

- Le menu en haut avec les sous-menus suivants:
  - Menu File pour les opérations sur les fichiers (ouvrir, nouveau, exporter)

  - Menu Edit pour les opérations de couper, copier, coller, faire et défaire
  - Menu View pour le zoom
  - Menu Layout pour changer le style de dessin du graphe
  - Menu Windows pour ouvrir des fenêtres supplémentaires
  - Menu Help pour l'aide et la version du logiciel
- Les raccourcis pour les options du menu les plus utilisées avec, dans l'ordre:
  - Nouveau
  - ouvrir
  - enregistrer
  - couper
  - copier
  - coller
  - dupliquer le style
  - faire
  - défaire
  - rechercher
  - zoomer
  - dézoomer
- La fenêtre des propriétés
- La fenêtre des noeuds environnants
- La fenêtre des outils
- La fenêtre principale contenant le graphe et les onglets pour les sous-graphe

6.3 Prototype et retour d'expérience

6.3.1 Choix pour le développement

En rapport avec les enseignements des chapitres précédents, voici une liste de fonctionnalités qui seront implémentées dans le prototype:

- Afficher un graphe comprenant des noeuds et arcs
- Ajouter des noeuds
- Ajouter des arcs
- Renommer un noeud

- Déplacer des noeuds
- Ajouter des contraintes "And" et "Or"
- Ajouter des cardinalités

6.3.2 Choix technologiques

Pour l'implémentation du logiciel, l'environnement de développement sera le logiciel Eclipse dans sa version 4.5, nommée Mars.
Le gestionnaire de plugin est celui intégré à Eclipse.

Afin de pouvoir développer un plugin, nous avons besoin de la sous-version RCP (rich client platform) d'Eclipse.
Cet environnement de développement supporte le langage Java qui a été choisi pour implémenter le programme.

L'environnement d'Eclipse comporte un plugin qui permet de construire des graphes. Il se nomme "Zest" et est disponible à l'adresse suivante: `https://www.eclipse.org/gef/zest/`. Ce plugin offre également plusieurs agencements:

- agencement basé sur des forces
- agencement en arborescence
- agencement circulaire
- agencement orthogonal

Apport avec Eclipse

L'utilisation d'Eclipse donne accès à plusieurs fonctionnalités:

- Les fenêtres
- Les perspectives qui gèrent l'emplacement des fenêtres
- Les commandes pour les fonctions couper/copier/coller

Apport avec Zest

Zest offre des fonctionnalités intégrées de zoom et de déplacements des noeuds.

6.3.3 Développement du prototype

Une capture d'écran du prototype est disponible à la figure 6.3. Le code est dans l'annexe B.

Le logiciel propose les fonctionnalités supplémentaires suivantes:

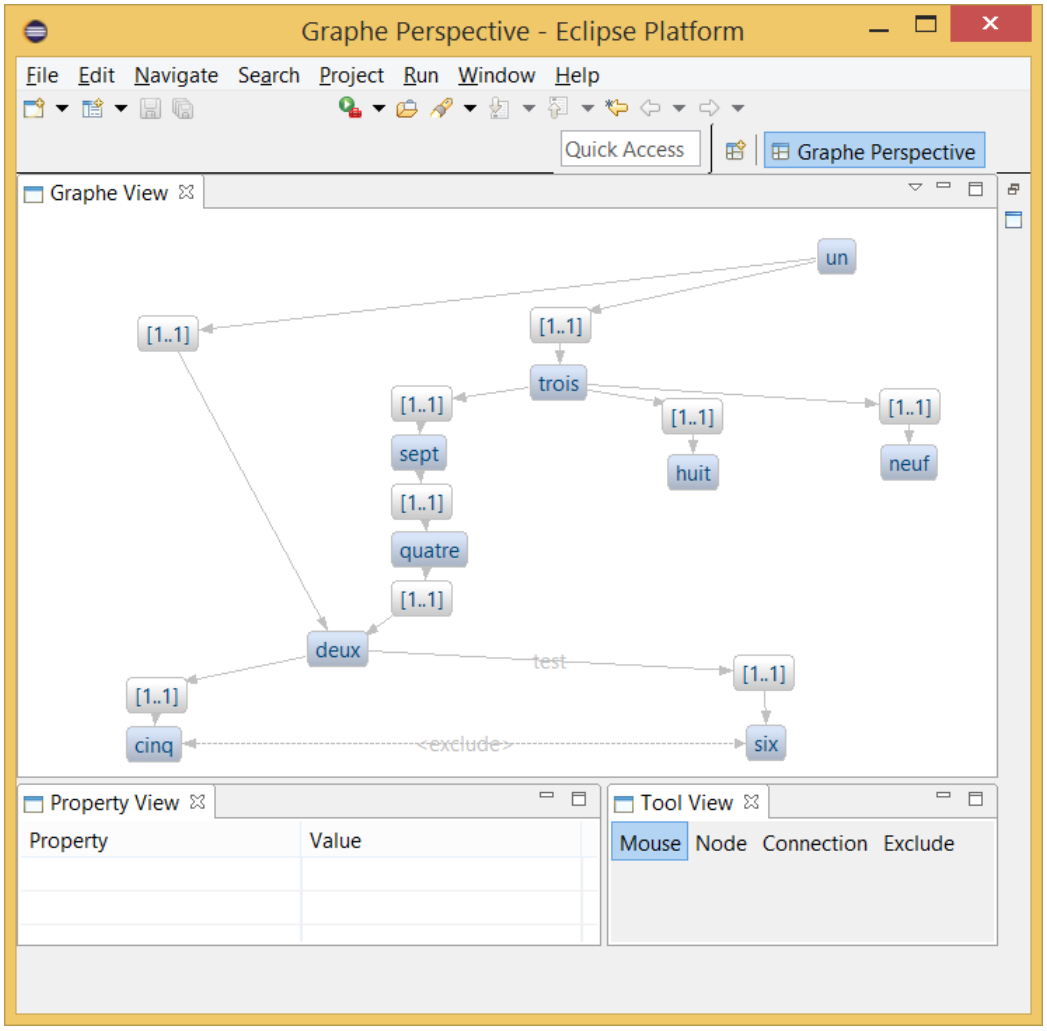


Figure 6.3: Capture d'écran du prototype

- Ajouter plusieurs noeuds en une seule fois
- Ajouter du texte sur un arc
- Ajouter une contrainte d'exclusion

6.4 Critiques du prototype

Eclipse est un produit complet mais complexe à prendre en mains surtout quand on essaie de l'étendre avec un plugin. Les fonctionnalités de base sont expliquées dans des tutoriaux proposés par des utilisateurs mais celles avancées ne sont pas couvertes ou très partiellement. La documentation d'Eclipse est très parcellaire et souvent mal conçue, surtout pour l'ajout de plugins. De plus, lier les fonctionnalités du plugins à celles d'Eclipse, comme le copier/coller ou les barres d'outils, est un processus très voire trop complexe.

Le framework Zest a le même problème qu'Eclipse. Peu d'explications sont fournies par les créateurs, ne parlons pas de la documentation du code qui est mal élaborée. Gérer les contraintes n'a été possible qu'en générant des noeuds de types contraintes, ce qui rend la compréhension du dessin plus difficile.

6.5 Conclusions

Dans ce chapitre, nous avons constaté que les fonctionnalités essentielles doivent être parmi les premières à être implémentées. D'autres fonctionnalités peuvent être ajoutées pour faciliter l'utilisation du logiciel comme l'ajout de plusieurs noeuds en une fois.

L'implémentation d'un prototype a été une tâche de longue durée et le manque de documentation un frein. Le choix d'utiliser Eclipse et un framework existant est une arme à double tranchant. Ils nout apportent des facilités mais d'autres parties que l'on voulait réaliser deviennent difficiles voire impossibles comme les commandes avec Eclipse ou la création intuitive de contraintes avec Zest.

Chapitre 7

Conclusion

Ce mémoire a commencé par une introduction présentant :

- le cadre, le nouveau langage TVL développé pour construire des features models sur base de la variabilité,
- les objectifs, le language TVL étant purement textuel, il est nécessaire de trouver un moyen de rendre un schéma représentant les concepts du code et leurs interactions,
- et le plan des différents chapitres.

Le premier chapitre a défini les concepts de la variabilité, des lignes de produits logiciels, les artefacts et des features models avec les liens qui les unissent.
Il a également présenté le langage TVL qui permet de concevoir ces features models. Il compare enfin les différences entre modèle textuel et modèle graphique pour conclure que ce sont deux vues complémentaires d'un même problème.

Dans le deuxième chapitre, nous avons abordé le concept mathématique de graphe. De sa représentation mathématique, un rendu est généré grâce à un algorithme. Celui-ci est choisi par le programme qui automatise le dessin et dépend les valeurs des critères sélectionnés par l'utilisateur afin d'obtenir une représentation claire et compréhensible. Ces critères sont les données à afficher, le support cible, la sémantique et le style de dessin.

Au cours du troisième chapitre, des tests sur des outils de variabilité utilisant la technologie WYSIWYG ont été réalisés. De ceux-ci, des résultats puis enseignements ont été tirés pour mettre en avant certaines fonctionnalités.

Le quatrième chapitre synthétise les états de l'art des deux chapitres précédents. Il propose une liste de fonctionnalités essentielles ainsi que certains manquements qui apporteraient une plus-value pour les outils dédiés à la variabilité.

Le dernier chapitre propose les spécifications d'un nouvel outil pour construire des

features models. Il comprend également un test effectué lors de la création d'un prototype et des difficultés rencontrées lors de son implémentation.

Proposer un outil visuel de génération de code TVL à partir de représentation de graphes est un excellent moyen de rendre accessible ce langage à des personnes qui n'y ont actuellement pas accès.
C'est de la vulgarisation de concepts évolués, tout comme les éditeurs de texte tel que Word ou Windows l'ont fait en leur temps.

Cet outil viendra enrichir l'écosystème qui devrait se créér autour de TVL en augmentant le nombre de personnes qui pourront être amenées à travailler avec ce langage. Il proposera une alternative au code brut et ne deviendra incontournable que si ses fonctionnalités s'enrichissent mutuellement et permettent d'accroitre la productivité des utilisateurs.

Comme améliorations, il serait intéressant d'augmenter la recherche d'outils en ne se limitant pas à ceux qui sont dédiés à la variabilité. Des logiciels tels que Yed ou Astah sont de bons exemples pour démontrer ce qu'il est possible de réaliser avec les graphes.
Il existe aussi d'autres algorithmes d'agencement de graphes, bien qu'ils soient souvent plus confidentiels. Des recherches plus poussées les concernant nous permettraient d'améliorer le rendu général des graphes en diminuant la complexité des calculs ou en apportant de nouvelles manières de visualiser un graphe.

Les futurs travaux se tourneront vers la recherche d'une bibliothèque logicielle permettant de construire des graphes avec suffisamment de documentation pour faciliter son apprentissage. Si aucune n'est découverte, il faudra soit prendre la moins catastrophique, soit en créer une de toutes pièces bien que ce type de chantier dépasse grandement le cadre de ce mémoire vu l'ampleur de la tâche. Il serait également intéressant de regarder du coté des nouvelles bibliothèques écrites en Javascript pour voir s'il n'est pas possible de créer un outil purement web.

Pour terminer cette conclusion et ce mémoire, et si c'était à recommencer, il faudrait limiter les travaux à la visualisation du code TVL sans s'attaquer à un outil WYSIWYG permettant de générer ce code. Cela simplifierait grandement les recherches et la création du logiciel. Une phase ultérieure se chargerait de créer l'outil de génération de code.

Bibliography

[Bra15] Aaron Bramson. Parameterized tree graph in netlogo, aout 2015. http://complexityblog.com/blog/index.php?itemid=70.

[Con15] Constant314. Telegrapher's equations, aout 2015. https://en.wikipedia.org/wiki/Telegrapher's_equations.

[Del15] David Delaunay. Graphe et ordonnancement de projet, aout 2015. http://mp.cpgedupuydelome.fr/document.php?doc=Article-Ordonnancementdeprojet.txt.

[Ebe15] Colin Eberhardt. Plotting circular relationship graphs with silverlight, aout 2015. http://www.codeproject.com/Articles/342715/Plotting-Circular-Relationship-Graphs-with-Silverl.

[ei15] Text encoding initiative. Graphs, networks, and trees, aout 2015. http://www.tei-c.org/release/doc/tei-p5-doc/fr/html/GD.html.

[Epi15] Epita. Les graphes > parcours profondeur > détails, aout 2015. http://algo-td.infoprepa.epita.fr/algo/spe/graphes_prof_appli.php.

[FB15] Paul C. Clements Felix Bachmann. Variability in software product lines, aout 2015. http://resources.sei.cmu.edu/asset_files/TechnicalReport/2005_005_001_14600.pdf.

[GP15] UK Gail Preston, University of Oxford. Multi-locus sequence analysis maximum likelihood tree of the actinobacterium phylum., aout 2015. http://www.frontiersin.org/files/Articles/103878/fpls-05-00406-HTML/image_m/fpls-05-00406-g004.jpg.

[Gra15] Cody Gray. Treeview with columns, aout 2015. http://stackoverflow.com/questions/4912873/treeview-with-columns.

[HE15] Seok-Hee Hong and Peter Eades. Drawing oneconnected planar graphs symmetrically, aout 2015. http://sydney.edu.au/engineering/it/~shhong/2dgraph.htm.

[HH15] Joshua Wing Kei Ho and Seok-Hee Hong. 2.5d drawings of clustered graphs, aout 2015. http://sydney.edu.au/engineering/it/~shhong/25dcluster.htm.

[Hun15] Michael Hunger. Why choose a graph database, aout 2015. http://radar.oreilly.com/2013/07/why-choose-a-graph-database.html/.

[Hya15] Hyacinth. Arc diagram, aout 2015. https://en.wikipedia.org/wiki/Arc_diagram.

[IDcsc15] Gordana Milosavljevi c Igor Dejanovi c, Maja Tumbas and Branko Peri si c. Comparison of textual and visual notations of dommlite domain-specific language, aout 2015. http://ceur-ws.org/Vol-639/131-dejanovic.pdf.

[IR15] Emil Eifrem Ian Robinson, Dr. Jim Webber. Graph databases, aout 2015. http://semanticommunity.info/Data_Science/Graph_Databases.

[JHS15] Peter Rodgers John Howse and Gem Stapleton. Automated diagram drawing, aout 2015. http://www.eulerdiagrams.com/tutorial/AutomatedDiagramDrawing.html.

[JR15] Texcel Research Jonathan Robie. What is the document object model?, aout 2015. http://www.w3.org/TR/DOM-Level-1/images/table.gif.

[Kor15] Yehuda Koren. Drawing graphs by eigenvectors: Theory and practice, aout 2015. http://www.research.att.com/export/sites/att_labs/groups/infovis/res/legacy_papers/DBLP-journals-camwa-Koren05.pdf.

[Mü15] Didier Müller. Lexique sur les graphes, aout 2015. http://www.nymphomath.ch/graphes-ancien/lexique/.

[Pan15] Pierre Pansu. Le découpage des graphes, aout 2015. http://images.math.cnrs.fr/Le-decoupage-des-graphes.html.

[Pol15a] Polytechnique. Graphes, aout 2015. https://www.enseignement.polytechnique.fr/informatique/ARCHIVES/IF/poly/main003.html.

[Pol15b] Polytechnique. Matrices d'adjacence, aout 2015. https://www.enseignement.polytechnique.fr/profs/informatique/Jean-Jacques.Levy/poly/main5/node4.html.

[Rap15] Christophe Rapine. Arbres et forêts, aout 2015. http://idmme06.inpg.fr/\~rapinec/Graphe/Arbre/default.html.

[RL15] Lionel Tabourier Renaud Lambiotte. Cours sur la théorie des graphes, aout 2015. http://webcampus.unamur.be/claroline/backends/download.php?url=L3N5bGxhYnVzL2dyYXBoZXNfMjAxMy0yMDE0LnBkZg%3D%3D&cidReset=true&cidReq=IHDCB321.

[rob15] roblevy. Welcome to datashine!, aout 2015. http://blogs.casa.ucl.ac.uk/category/visualisation/.

[Rus15] Adrian Rusu. Tree drawing algorithms, aout 2015. http://cs.brown.edu/\~rt/gdhandbook/chapters/trees.pdf.

[Sac15] Joe Sack. The case of the cardinality estimate red herring, aout 2015. http://sqlperformance.com/2013/04/t-sql-queries/the-case-of-the-cardinality-estimate-red-herring.

[Seg15] Segura09. E-shopfm, aout 2015. https://commons.wikimedia.org/wiki/File:E-shopFM.jpg.

[Teh15] Tehnick. Layered graph drawing, aout 2015. http://www.wikiwand.com/en/Layered_graph_drawingl.

[TNTUT04] Md Saidur Rahman (Bangladesh University of Engineering Takao Nishizeki (Tohoku University, Japan) and Bangladesh) Technology. *Planar Graph Drawing*. Wspc, 2004.

[Uni15] Carnegie Mellon University. Layered graph drawing, aout 2015. http://www.sei.cmu.edu/productlines/frame_report/what.is.a.PL.htm.

[UR15] T Dwyer K Marriott M Wybrow U Rüegg, S Kieffer. Stress-minimizing orthogonal layout of data flow diagrams with ports, aout 2015. http://marvl.infotech.monash.edu/\~dwyer/.

[Ver15] Académie Versailles. Graphe orienté, aout 2015. https://euler.ac-versailles.fr/baseeuler/lexique/notion.jsp?id=185.

[Wik15a] Wikipédia. Feature model, aout 2015. https://en.wikipedia.org/wiki/Feature_model.

[Wik15b] Wikipédia. Graphe planaire, aout 2015. https://fr.wikipedia.org/wiki/Graphe_planaire.

[Wik15c] Wikipédia. Isabelle de naples, aout 2015. https://fr.wikipedia.org/wiki/Isabelle_de_Naples.

[Wik15d] Wikipédia. Matrice des degrés, aout 2015. https://fr.wikipedia.org/wiki/Matrice_des_degr%C3%A9s.

[Wik15e] Wikipédia. Matrice laplacienne, aout 2015. https://fr.wikipedia.org/wiki/Matrice_laplacienne.

[Wik15f] Wikipédia. Théorie des graphes - lexique, aout 2015. https://fr.wikipedia.org/wiki/Lexique_de_la_thÃľorie_des_graphes\#A.

[Ywo15a] Yworks. Compact orthogonal layout - chapter 5. automatic graph layout, aout 2015. http://docs.yworks.com/yfiles/doc/developers-guide/compact_orthogonal_layouter.html.

[Ywo15b] Yworks. Visual representation of graph elements - chapter 2. displaying and editing graphs, aout 2015. http://docs.yworks.com/yfilesflex/doc/dguide/styles.html.

[Zhu15] Leonid Zhukov. Exploratory graph visualization, aout 2015. https://www.sci.utah.edu/alumni-highlight/426-leonid-zhukov.html.

Appendix A

Annexe 1

A.1 Captain Feature

Scénario 1: Installation

Télécharger le logiciel sur http://sourceforge.net/projects/captainfeature/?source=navbar
.
Extraire le fichier compressé.
Lancer l'interface en double-cliquant sur le fichier run.bat à la racine du dossier extrait.

Durée: 10 minutes

Scénario 2: Création d'un projet

Ce scénario a été réalisé sans rencontrer de problème.
Il n'est pas possible de sélectionner une arète (ou arc car ils sont d'office orienté). Donc impossible d'également la nommer.
Pour renommer un noeud, les opérations sont assez longues. Il faut:

- Sélectionner le noeud dans le treeview de gauche
- Cliquer droit pour faire apparaitre le menu contextuel
- Cliquer sur 'Properties'
- Modifier le nom dans la fenêtre qui vient d'apparaitre

Durée: 15 minutes

Scénario 3: Reproduire le schéma ci-dessous

Le programme ne supporte ni les contraintes, ni les cardinalités. Seul le graphe vierge a pu être reproduit.
Durée: 10 minutes

Scénario 4: Ressources utilisées

Le programme occupe 86 Mb en mémoire.

Scénario 5: Fonctionnalités externes aux processus métiers

Vérifier si le programme offre:

- *une fonctionnalité faire-défaire pour plusieurs opérations contiguës:* Non
- *une fonctionnalité de zoom:* Non
- *une fonctionnalité de recherche sur le schéma:* Non
- *une fonctionnalité d'export du schéma sous forme d'image:* Oui (jpeg)
- *une fonctionnalité de sauvegarde du schéma:* Oui
- *une fonctionnalité d'ouverture d'un schéma sauvegardé interne au programme:* Non
- *une fonctionnalité d'ouverture d'un schéma sauvegardé externe au programme:* Non
- *une fonctionnalité de couper/copier/coller:* Non

Autres questions

1. *Le programme est-il portable sur d'autres plateformes (linux,...)?* La version 1.0 ne supporte que windows
2. *Les fichiers sauvegardés sont-ils utilisables pour d'autres programmes?* Oui, ce sont des fichiers xml
3. *Le programme est-il encore supporté par une équipe? Bugfixing? Nouveaux développements ?* La dernière version fournie date du 22/03/2013. Le support est parcelaire. Aucun nouveau développement n'a été réclamé, aucun bug récent ($<$ 1 an) n'est ouvert
4. *Depuis combien de temps le programme existe-t-il?* 10 ans (enregistré 05/05/2003)
5. *Le programme propose-t-il des extensions par plugin?* Non
6. *Les fonctionnalités du programme comportent-elles encore des bugs?* Oui, la fenêtre de propriétés ne s'ouvre pas sur le bon onglet. Ce n'est pas bloquant mais embêtant
7. *Est-ce qu'une opération non appropriée/autorisée fait fermer le programme?* Aucun problème de ce genre n'a été rencontré

8. *Le travail d'une session est-il récupérable après une fermeture non prévue du programme?* Je ne pense pas
9. *L'apprentissage est-il aisé?* L'apprentissage de base est assez aisé. Il existe une aide pour des fonctionnalités plus avancées
10. *L'interface homme-machine est-elle facilement compréhensible?* Oui, même si la fenêtre des propriétés est embêtante à aller chercher et est bloquante pour les autres opérations
11. *Est-elle attractive?* De mon point de vue, pas vraiment.
12. *Existe-il une version de production?* Oui, la dernière version est la 1.0
13. *Le programme propose-t-il des fonctions de décorations du schéma (couleur, type de noeuds, type de traits)?* Non
14. *Quelles sont les vues proposées par le programme (graphe, treeview,...)?* Graphe pour les modèles, treeview pour lister le contenu
15. *Est-il possible de rajouter un attribut/propriété dans le modèle?* Tout ce qui peut être spécifié c'est le type de la feature
16. *L'ajout des contraintes est-il possible?* Non

A.2 Feature Mapper

Scénario 1: Installation

Pour l'installer:

- Installer Eclipse version 'Eclipse IDE for Java Developers'
- Ouvrir Eclipse et aller à 'Help > Install New Software...'
- Ajouter le site suivant : http://featuremapper.org/update/
- Sélectionner les fonctionnalités FeatureMapper
- Continuer avec 'Install...'

Extensions installées:

- FeatureMapper Core
- FeatureMapper User Guide

J'ai rencontré des problèmes du à une tentative d'installation sur une trop récente version d'Eclipse.
La version utilisée était la 4.3 et une erreur apparaissait à la fin de l'installation du plugin. La version 3.7 ne pose pas de problème à l'installation.

Durée: 40 minutes

Scénario 2: Création d'un projet

L'affichage en treeview n'aide pas vraiment à construire un graphe rapidement.
Il n'est pas possible de nommer un arc. Il n'y a pas d'arêtes vu que le graphe est complètement orienté.

Pour renommer un arc, il faut passer par les propriétés puis changer la propriété "name". C'est long. Pas de double-clic ni autre raccourci.

Durée 20 minutes

Scénario 3: Reproduire le schéma ci-dessous

Ce scénario a été long à réaliser car la navigation entre les différentes parties n'est pas optimale. En effet, aller sans cesse de la partie graphe à la partie propriétés est une perte de temps.

De plus, on se demande à quoi servent toutes les cardinalités disponibles entre celles des features et des groupes.

L'éditeur de contraintes n'est pas intuitif. On se demande si les contraintes ajoutées seront correctement interprétées.

Durée : 40 minutes

Scénario 4: Ressources utilisées

Pas d'impact sur la consommation mémoire.

Scénario 5: Fonctionnalités externes aux processus métiers

Vérifier si le programme offre:

- *une fonctionnalité faire-défaire pour plusieurs opérations contiguës:* Oui
- *une fonctionnalité de zoom:* Non
- *une fonctionnalité de recherche sur le schéma:* Non
- *une fonctionnalité d'export du schéma sous forme d'image:* Non
- *une fonctionnalité de sauvegarde du schéma:* Oui mais attention à l'encodage et aux noms des noeuds. Il se peut qu'à la réouverture du projet, il n'est plus possible de générer le graphe.
- *une fonctionnalité d'ouverture d'un schéma sauvegardé interne au programme:* Oui

- *une fonctionnalité d'ouverture d'un schéma sauvegardé externe au programme:* Oui
- *une fonctionnalité de couper/copier/coller:* Oui

Autres questions

1. *Le programme propose-t-il des extensions par plugin?* Le programme est une extension d'Eclispe
2. *Le programme est-il portable sur d'autres plateformes (linux,...)?* Selon eclipse, Mac OS X, Windows, Linux
3. *Les fichiers sauvegardés sont-ils utilisables pour d'autres programmes?* Ce sont des fichiers xml, donc oui. Seule l'extension change
4. *Le programme est-il encore supporté par une équipe? Bugfixing? Nouveaux développements?* La dernière mise à jour date du 04/09/2013 et ne comprenait que le support du dernier SDK d'eclipse. Pas de bugtracker accessible, pas de nouvelles features prévues.
5. *Depuis combien de temps le programme existe-t-il?* Depuis le 22/01/2008
6. *Les fonctionnalités du programme comportent-elles encore des bugs?* Oui, surtout pour la réouverture des fichiers qui contiennent des erreurs. Ils en deviennent rapidement indébugables.
7. *Est-ce qu'une opération non appropriée/autorisée fait fermer le programme?* Aucun opération qui a demandé une réouverture du programme n'a été rencontrée lors des tests.
8. *Le travail d'une session est-il récupérable après une fermeture non prévue du programme?* Cela dépend d'eclipse
9. *L'apprentissage est-il aisé?* Non, créer un projet est déjà le parcours du combattant sans tutoriel complet. On se demande pourquoi certaines possibilités sont proposées (cardinalité partout). On navigue souvent à l'aveugle
10. *L'interface homme-machine est-elle facilement compréhensible?* Si l'on comprend que tout se passe dans la fenêtre des propriétés et dans la vue oui. Mais elle n'est pas pensée pour une utilisation à long terme.
11. *Est-elle attractive?* C'est l'interface d'Eclipse
12. *Existe-il une version de production?* Je dirais non, vu que la dernière version est la 0.8.9. Pas de version 1.0
13. *Le programme propose-t-il des fonctions de décorations du schéma (couleur, type de noeuds, type de traits)?* Pas de décorations disponibles

14. *Quelles sont les vues proposées par le programme (graphe, treeview,...)?* Uniquement le treeview

15. *Est-il possible de rajouter un attribut/propriété dans le modèle?* Il est possible de rajouter des attributs dans le modèle

16. *L'ajout des contraintes est-il possible?* Il est possible de rajouter des contraintes.

A.3 Feature Modeling

Scénario 1: Installation

Téléchargement du plugin et ajout de celui-ci dans le folder des plugins d'Eclipse.

Durée: 10 minutes

Scénario 2: Création d'un projet

Pas d'arcs visibles.

Durée: 10 minutes

Scénario 3: Reproduire le schéma ci-dessous

Le scénario s'est déroulé sans problème.

Durée: 20 minutes

Scénario 4: Ressources utilisées

Pas d'impact sur la mémoire consommée par Eclipse.

Scénario 5: Fonctionnalités externes aux processus métiers

Vérifier si le programme offre:

- *une fonctionnalité faire-défaire pour plusieurs opérations contiguës:* Oui
- *une fonctionnalité de zoom:* Oui
- *une fonctionnalité de recherche sur le schéma:* Oui par eclipse
- *une fonctionnalité d'export du schéma sous forme d'image:* Non
- *une fonctionnalité de sauvegarde du schéma:* Oui
- *une fonctionnalité d'ouverture d'un schéma sauvegardé interne au programme:* Oui par eclipse

- *une fonctionnalité d'ouverture d'un schéma sauvegardé externe au programme:* Oui par eclipse
- *une fonctionnalité de couper/copier/coller:* Non

Autres questions

1. *Le programme propose-t-il des extensions par plugin?* Oui par eclipse
2. *Le programme est-il portable sur d'autres plateformes (linux,...)?* Suivant la portabilité d'eclipse
3. *Les fichiers sauvegardés sont-ils utilisables pour d'autres programmes?* Oui, ils sont sous format xml
4. *Le programme est-il encore supporté par une équipe? Bugfixing? Nouveaux développements?* Le programme est considéré comme achevé, pas de développement ni de bugfixing
5. *Depuis combien de temps le programme existe-t-il?* 2004
6. *Les fonctionnalités du programme comportent-elles encore des bugs?* Il y a une liste de problèmes connus
7. *Est-ce qu'une opération non appropriée/autorisée fait fermer le programme?* Pas à ma connaissance
8. *Le travail d'une session est-il récupérable après une fermeture non prévue du programme?* Non
9. *L'apprentissage est-il aisé?* Difficile de comprendre certaines choses
10. *L'interface homme-machine est-elle facilement compréhensible?* Oui, tout est gérable depuis le graphe
11. *Est-elle attractive?* Peut mieux faire
12. *Existe-il une version de production?* Oui, la 0.6.6
13. *Le programme propose-t-il des fonctions de décorations du schéma (couleur, type de noeuds, type de traits)?* Non
14. *Quelles sont les vues proposées par le programme (graphe, treeview,...)?* Treeview uniquement
15. *Est-il possible de rajouter un attribut/propriété dans le modèle?* Pas à ma connaissance
16. *L'ajout des contraintes est-il possible?* Non

A.4 Feature IDE

Le plufin est disponible depuis les extensions d'Eclipse sur les versions depuis la 3.4 (Ganymede) à 4.3 (Kepler).
Le test est effectué sur la version Kepler
Page de présentation: http://www.fosd.de/featureide/
GitHub: https://github.com/tthuem/FeatureIDE
Marketplace Eclipse: http://marketplace.eclipse.org/content/featureide#.Ut18KrTjLct

Scénario 1: Installation

Pour l'installer:

- Installer Eclipse version 'Eclipse IDE for Java Developers'
- Ouvrir Eclipse et aller à 'Help > Install New Software...'
- Ajouter le site suivant : http://wwwiti.cs.uni-magdeburg.de/iti_db/research/featureide/deploy/
- Sélectionner les fonctionnalités FeatureIDE, Feature Modeling et les extensions Features IDE requises.
- Continuer avec 'Install...'

Extensions installées:

- Colligens
- Feature Modeling
- FeatureIDE
- FeatureIDE example projects

L'installation s'est déroulée sans problème.
Durée: 10 minutes

Scénario 2: Création d'un projet

La prise en main se déroule sans problème. Il y a un tutoriel pour nous guider.
Il est possible de créer des noeuds.
Les liens sont créés automatiquement, entre deux noeuds selon une hiérarchie en arborescence.
Il n'est pas possible de nommer une arête, mais elles sont orientées suivant la hiérarchie.
A noter que les noms des noeuds sont uniques dans le graphe.

Durée: 10 minutes

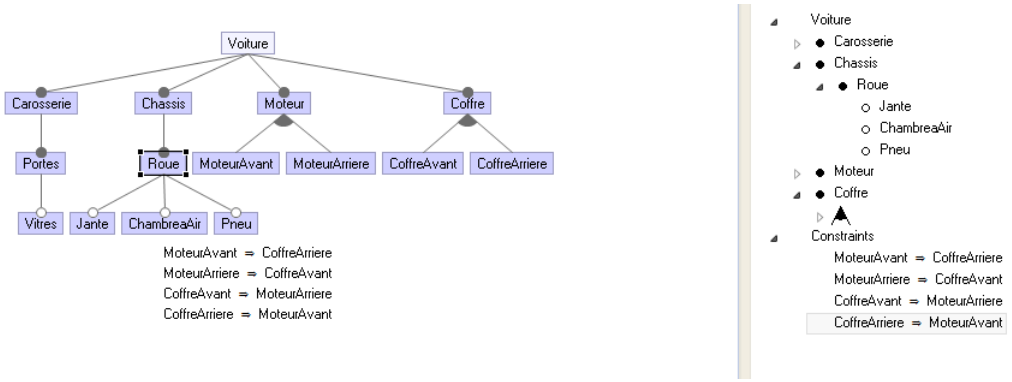


Figure A.1: Graphe généré par Feature IDE

Scénario 3: Reproduire le schéma ci-dessous

La réalisation du scénario a donné le schéma suivant référencé à la figure A.1.
Il n'a pas été possible d'assigner 4 roues.
Des contraintes sont ajoutables facilement via un éditeur externe au schéma.

Scénario 4: Ressources utilisées

L'impact sur la mémoire n'est que de 2MB de RAM. C'est Eclipse qui en utilise le plus (+/- 200MB)

Scénario 5: Fonctionnalités externes aux processus métiers

Le programme offre les fonctionnalités de faire-défaire sur plusieurs opérations contiguës par les combinaisons de touche CTRL+Z et CTRL+Y ainsi que par un sous-menu dans la barre des menus.
Le programme offre une fonction de zoom accessible dans dans la barre d'outils.
La recherche est possible dans le graphe par le module de recherche d'Eclipse. Elle s'effectue dans le fichier xml.
L'export du schéma est possible par l'option 'Save As' et le sauve sous la forme d'un fichier png, bmp ou jpeg.
Le fichier peut également être exporté sous forme d'un fichier xml ou d'un fichier GUIDSL Grammar (*.m)
On peut importer un fichier xml grâce à l'interface d'Eclipse. Si le format n'est pas celui attendu par le module, le contenu brut est affiché au lieu de la représentation sous forme de graphe. Une croix rouge est également ajouté sur l'icone du fichier dans l'explorateur de solutions.
La ou les lignes incorrectes sont également signalées.
L'export est possible par l'interface d'Eclipse.
Les fonctions de copier/couper/coller ne sont pas disponibles.

Autres questions

1. *Le programme propose-t-il des extensions par plugin?* Le programme est déjà lui-même un plugin d'Eclipse. Il peut lui aussi supporter d'autres plugins.

2. *Le programme est-il portable sur d'autres plateformes (linux,...)?* Eclipse est portable sur les plateformes suivantes:

   - linux
   - Windows
   - MAC OS X

3. *Les fichiers sauvegardés sont-ils utilisables pour d'autres programmes?* Les fichiers sont sauvegardés sous format XML. Ils sont transformables et utilisables par d'autres programmes

4. *Le programme est-il encore supporté par une équipe? Bugfixing? Nouveaux développements?* La version courante est la 2.6.7, disponible depuis octobre 2013
   Il n'y a pas d'annonce pour une future version, ni de nouvelles fonctionnalités mais il y a de l'activité sur le dépot GitHub (https://github.com/tthuem/FeatureIDE). De plus 32 demandes d'ajout de fonctionnalités sont ouvertes dans le bugtracker. https://github.com/tthuem/FeatureIDE/issues?labels=enhancement&page=1&state=open
   Il y a un bugtracker disponible pour signaler les bugs. Il est disponible à cette adresse: https://github.com/tthuem/FeatureIDE/issues

5. *Depuis combien de temps le programme existe-t-il?* Le plugin existe depuis 1997

6. *Les fonctionnalités du programme comportent-elles encore des bugs?* Actuellement il y a 12 bugs ouverts

7. *Est-ce qu'une opération non appropriée/autorisée fait fermer le programme?* L'application est relativement stable. Je n'ai pas rencontré de problème durant mes tests.

8. *Le travail d'une session est-il récupérable après une fermeture non prévue du programme?* Cela dépend d'eclipse

9. *L'apprentissage est-il aisé?* L'apprentissge est aisé car il existe un tutoriel pour démarrer.

10. *L'interface homme-machine est-elle facilement compréhensible?* Il faut un petit moment d'adaptation dû à l'environnement d'eclipse et des différentes vues qui sont ouvertes. Il faut également savoir que le menu contextuel contient la plupart des outils nécessaires au rajout de composants dans l'arbre.

11. *Est-elle attractive?* L'interface est suffisament épurée pour être attractive. Ca n'est pas fait par des designers mais c'est fonctionnel.

12. *Existe-il une version de production?* La version de production est la 2.6.7

13. *Le programme propose-t-il des fonctions de décorations du schéma (couleur, type de noeuds, type de traits)?* Le layout de l'arbre peut être modifié. Aucun changement de couleurs, de types de traits ou de noeuds n'est disponible.

14. *Quelles sont les vues proposées par le programme (graphe, treeview,...)?* Il y a une vue

    - En arbre (le design peut-être modifié)
    - En treeview
    - Des contraintes du graphe

15. *Est-il possible de rajouter un attribut/propriété dans le modèle?* Les noeuds ne peuvent pas accueillir d'attributs.

16. *L'ajout des contraintes est-il possible?* Il est possible d'ajouter des contraintes entre les noeuds par un gestionnaire externe au graphe

A.5 Hypersense

Scénario 1: Installation

L'installation s'est déroulée sans problème mais impossible de trouver ou d'activer une licence valide.
Il est donc impossible de tester le programme
Durée: N/A

Scénario 2: Création d'un projet

Non réalisé. Raison: voir scénario 1

Scénario 3: Reproduire le schéma ci-dessous

Non réalisé. Raison: voir scénario 1

Scénario 4: Ressources utilisées

Non réalisé. Raison: voir scénario 1

Scénario 5: Fonctionnalités externes aux processus métiers

Non réalisé. Raison: voir scénario 1

Autres questions

1. *Le programme propose-t-il des extensions par plugin?* C'est un plugin Eclispe, donc oui
2. *Le programme est-il portable sur d'autres plateformes (linux,...)?* Non
3. *Les fichiers sauvegardés sont-ils utilisables pour d'autres programmes?* Aucune idée, impossible de lancer le programme
4. *Le programme est-il encore supporté par une équipe? Bugfixing? Nouveaux développements?* Pas de communication sur ce point-là
5. *Depuis combien de temps le programme existe-t-il?* Au moins 2012
6. *Les fonctionnalités du programme comportent-elles encore des bugs?* Aucune idée, impossible de lancer le programme
7. *Est-ce qu'une opération non appropriée/autorisée fait fermer le programme?* Aucune idée, impossible de lancer le programme
8. *Le travail d'une session est-il récupérable après une fermeture non prévue du programme?* Aucune idée, impossible de lancer le programme
9. *L'apprentissage est-il aisé?* Aucune idée, impossible de lancer le programme
10. *L'interface homme-machine est-elle facilement compréhensible?* Aucune idée, impossible de lancer le programme
11. *Est-elle attractive?* Aucune idée, impossible de lancer le programme
12. *Existe-il une version de production?* Aucune idée
13. *Le programme propose-t-il des fonctions de décorations du schéma (couleur, type de noeuds, type de traits)?* Aucune idée, impossible de lancer le programme
14. *Quelles sont les vues proposées par le programme (graphe, treeview,...)?* Aucune idée, impossible de lancer le programme
15. *Est-il possible de rajouter un attribut/propriété dans le modèle?* Aucune idée, impossible de lancer le programme
16. *L'ajout des contraintes est-il possible?* Aucune idée, impossible de lancer le programme

A.6 MetaEdit

Scénario 1: Installation

Téléchargement et installation sans problème

Durée: 20 minutes

Scénario 2: Création d'un projet

Difficulté de compréhension du logiciel.
Il est trop complet.

Sinon il n'est pas possible de nommer les arcs.

Durée: 10 minutes

Scénario 3: Reproduire le schéma ci-dessous

Le schéma n'a posé aucun problème à créer mais il n'est pas possible de rajouter des contraintes

Durée: 20 minutes

Scénario 4: Ressources utilisées

Mémoire utilisée : 36 Mb

Scénario 5: Fonctionnalités externes aux processus métiers

Vérifier si le programme offre:

- *une fonctionnalité faire-défaire pour plusieurs opérations contiguës:* oui
- *une fonctionnalité de zoom:* Oui
- *une fonctionnalité de recherche sur le schéma:* Non
- *une fonctionnalité d'export du schéma sous forme d'image:* Oui, sous format bitmap, gif, png
- *une fonctionnalité de sauvegarde du schéma:* Automatique
- *une fonctionnalité d'ouverture d'un schéma sauvegardé interne au programme:* Oui
- *une fonctionnalité d'ouverture d'un schéma sauvegardé externe au programme:* Import possible
- *une fonctionnalité de couper/copier/coller:* Oui

Autres questions

1. *Le programme propose-t-il des extensions par plugin?* Non
2. *Le programme est-il portable sur d'autres plateformes (linux,...)?* Mac OS X, linux, windows

3. *Les fichiers sauvegardés sont-ils utilisables pour d'autres programmes?* Je ne pense pas

4. *Le programme est-il encore supporté par une équipe? Bugfixing? Nouveaux développements?* Oui, c'est un programme commercial.

5. *Depuis combien de temps le programme existe-t-il?* 1991

6. *Les fonctionnalités du programme comportent-elles encore des bugs?* Pas de problèmes rencontrés

7. *Est-ce qu'une opération non appropriée/autorisée fait fermer le programme?* Aucune idée

8. *Le travail d'une session est-il récupérable après une fermeture non prévue du programme?* Je ne pense pas

9. *L'apprentissage est-il aisé?* Non, on est perdu

10. *L'interface homme-machine est-elle facilement compréhensible?* Non, trop d'informations dans les écrans du début

11. *Est-elle attractive?* Ca peut aller

12. *Existe-il une version de production?* Oui

13. *Le programme propose-t-il des fonctions de décorations du schéma (couleur, type de noeuds, type de traits)?* Non

14. *Quelles sont les vues proposées par le programme (graphe, treeview,...)?* Graphe et treeview

15. *Est-il possible de rajouter un attribut/propriété dans le modèle?* Pas à l'origine mais il est possible d'en rajouter

16. *L'ajout des contraintes est-il possible?* Pas le feature model de base

A.7 PureVariants

Scénario 1: Installation

Il faut

1. aller sur le site internet http://www.pure-systems.com/Variant_Management.49+M54a708de802.0

2. Télécharger la version community (prendre la version complète)

3. Installer le programme

4. Lancer eclipse lié au répertoire .../pure-systems/pv_Community Edition_3.2/eclipse

Notes:

- Pure variants ne fonctionne que sur un système d'exploitation 32 bits.
- Le programme nécessite d'avoir java, version minimum 1.5, configuration 32 bits.

Durée: 30 minutes

Scénario 2: Création d'un projet

Le scénario n'a pas posé de problème.
Remarques:

- les arêtes sont d'office des arcs
- Pas de nommage possible des arcs
- Pas moyens d'inverser les arcs

Durée: 10 minutes

Scénario 3: Reproduire le schéma ci-dessous

Le scénario n'a pas posé de problème.
Les contraintes sont à exprimer en prolog ou pvscl.
Les contraintes n'ont pas été ajoutées.
Pas de cardinalités possibles sur les éléments mandatory.

Les types de noeuds (mandatory, optional...) sont sous forme d'icône.

Durée: 10 minutes

Scénario 4: Ressources utilisées

A part eclipse, pas de charge supplémentaire significative à l'ouverture d'un logiciel.

Scénario 5: Fonctionnalités externes aux processus métiers

Vérifier si le programme offre:

- *une fonctionnalité faire-défaire pour plusieurs opérations contiguës:* oui
- *une fonctionnalité de zoom:* Non
- *une fonctionnalité de recherche sur le schéma:* Oui par les fonctions d'eclipse
- *une fonctionnalité d'export du schéma sous forme d'image:* Non. Mais, dans la version payante, des exports en csv, html, xml et définis par l'utilisateur existent

- *une fonctionnalité de sauvegarde du schéma:* Oui
- *une fonctionnalité d'ouverture d'un schéma sauvegardé interne au programme:* Oui par les fonctionnalités d'Eclipse
- *une fonctionnalité d'ouverture d'un schéma sauvegardé externe au programme:* Oui par les fonctionnalités d'Eclipse
- *une fonctionnalité de couper/copier/coller:* Oui mais ça copie tout l'arborescence qui suit

Autres questions

1. *Le programme propose-t-il des extensions par plugin?* Oui, grâce à eclipse
2. *Le programme est-il portable sur d'autres plateformes (linux,...)?* oui par eclipse (mac os x, linux, windows)
3. *Les fichiers sauvegardés sont-ils utilisables pour d'autres programmes?* oui, ce sont des fichiers xml avec une extension spécifique
4. *Le programme est-il encore supporté par une équipe? Bugfixing? Nouveaux développements?* C'est un programme commercial, donc oui (enfin, j'espère). Pas de bugtracker ni de liste de nouveaux développements disponibles.
5. *Depuis combien de temps le programme existe-t-il?* Le spin-off éditant le logiciel existe depuis 2001
6. *Les fonctionnalités du programme comportent-elles encore des bugs?* La seule limitation trouvée est l'impossibilité d'ouvrir deux modèles en même temps
7. *Est-ce qu'une opération non appropriée/autorisée fait fermer le programme?* Pas rencontré de bug de ce genre durant les tests
8. *Le travail d'une session est-il récupérable après une fermeture non prévue du programme?* Tout dépend d'eclipse
9. *L'apprentissage est-il aisé?* La prise en main est rapide
10. *L'interface homme-machine est-elle facilement compréhensible?* L'interface est assez bien pensée, utilisable rapidement
11. *Est-elle attractive?* C'est l'interface d'eclipse
12. *Existe-il une version de production?* Oui la 3.2.7
13. *Le programme propose-t-il des fonctions de décorations du schéma (couleur, type de noeuds, type de traits)?* Non
14. *Quelles sont les vues proposées par le programme (graphe, treeview,...)?* graphe, treeview, table, contraintes

15. *Est-il possible de rajouter un attribut/propriété dans le modèle?* Oui, même directement dans le graphe

16. *L'ajout des contraintes est-il possible?* Oui par une interface externe

A.8 SPLOT

Scénario 1: Installation

Le programme est une application web, pas besoin d'installation, ni d'inscription.

Durée: 5 minutes

Scénario 2: Création d'un projet

Le scénario s'est déroulé sans problème.
Toutes les arêtes sont des arcs à orientation unique. On ne peut les nommer.

Durée: 5 minutes

Scénario 3: Reproduire le schéma ci-dessous

Le scénario s'est déroulé sans problème.

Il n'est pas possible de mettre des valeurs de maximums et des minimums spécifiques.

Durée: 10 minutes

Scénario 4: Ressources utilisées

N/A, c'est une application web.

Scénario 5: Fonctionnalités externes aux processus métiers

Vérifier si le programme offre:

- *une fonctionnalité faire-défaire pour plusieurs opérations contiguës:* oui, sauf sur les contraintes
- *une fonctionnalité de zoom:* Oui grâce au navigateur
- *une fonctionnalité de recherche sur le schéma:* Oui grâce au navigateur
- *une fonctionnalité d'export du schéma sous forme d'image:* print screen
- *une fonctionnalité de sauvegarde du schéma:* Oui dans le repository du serveur

- *une fonctionnalité d'ouverture d'un schéma sauvegardé interne au programme:* récupération d'un schéma existant
- *une fonctionnalité d'ouverture d'un schéma sauvegardé externe au programme:* non
- *une fonctionnalité de couper/copier/coller:* Non, sauf sur les contraintes (duplication)

Autres questions

1. *Le programme propose-t-il des extensions par plugin?* Non
2. *Le programme est-il portable sur d'autres plateformes (linux,...)?* N/A
3. *Les fichiers sauvegardés sont-ils utilisables pour d'autres programmes?* Oui, c'est exporté au format xml
4. *Le programme est-il encore supporté par une équipe? Bugfixing? Nouveaux développements?* Non
5. *Depuis combien de temps le programme existe-t-il?* 2009
6. *Les fonctionnalités du programme comportent-elles encore des bugs?* Aucun bug rencontré
7. *Est-ce qu'une opération non appropriée/autorisée fait fermer le programme?* Non
8. *Le travail d'une session est-il récupérable après une fermeture non prévue du programme?* Non
9. *L'apprentissage est-il aisé?* Oui
10. *L'interface homme-machine est-elle facilement compréhensible?* Un peu dur de trouver où l'on rajoute un noeud
11. *Est-elle attractive?* Oui
12. *Existe-il une version de production?* Oui
13. *Le programme propose-t-il des fonctions de décorations du schéma (couleur, type de noeuds, type de traits)?* Non
14. *Quelles sont les vues proposées par le programme (graphe, treeview,...)?* Treeview uniquement
15. *Est-il possible de rajouter un attribut/propriété dans le modèle?* Non
16. *L'ajout des contraintes est-il possible?* Oui, un éditeur simple est accessible

A.9 VMC

Scénario 1: Installation

Il n'y a pas d'installation pour pour ce programme car il n'est accessible qu'en ligne.
Il suffit de se connecter au site : `http://fmtlab.isti.cnr.it/vmc/V5.5/vmc.html`.

Durée: 1 minute

Scénario 2: Création d'un projet

On ne peut que créer des modèles en ligne.
Ils sont décrits avant un rendu graphique.

Il n'est pas possible de nommer les noeuds, seulement les arêtes.
Le rendu après avoir édité le contenu du texte est assez dérangeant surtout qu'il faut cliquer sur plusieurs liens pour y arriver.

Durée: 10 minutes

Scénario 3: Reproduire le schéma ci-dessous

Reproduire le schéma demandé est impossible étant donné que c'est plutôt un diagramme d'état que l'on dessine.

Durée: minutes

Scénario 4: Ressources utilisées

Cette question n'a pas d'application ici.

Scénario 5: Fonctionnalités externes aux processus métiers

Vérifier si le programme offre:

- *une fonctionnalité faire-défaire pour plusieurs opérations contiguës:* oui sur l'édition du contenu
- *une fonctionnalité de zoom:* oui sur le diagramme généré
- *une fonctionnalité de recherche sur le schéma:* sur le contenu par l'intermédiaire du navigateur
- *une fonctionnalité d'export du schéma sous forme d'image:* enregister l'image générée depuis le navigateur
- *une fonctionnalité de sauvegarde du schéma:* enregister le contenu

- *une fonctionnalité d'ouverture d'un schéma sauvegardé interne au programme:* non
- *une fonctionnalité d'ouverture d'un schéma sauvegardé externe au programme:* non
- *une fonctionnalité de couper/copier/coller:* sur le contenu

Autres questions

1. *Le programme propose-t-il des extensions par plugin?* Non
2. *Le programme est-il portable sur d'autres plateformes (linux,...)?* Tant que la plateforme possède un navigateur compatible, oui
3. *Les fichiers sauvegardés sont-ils utilisables pour d'autres programmes?* Non, le format est trop spécifique
4. *Le programme est-il encore supporté par une équipe? Bugfixing? Nouveaux développements?* Aucune indication sur le site internet
5. *Depuis combien de temps le programme existe-t-il?* Pas d'indication
6. *Les fonctionnalités du programme comportent-elles encore des bugs?* Non
7. *Est-ce qu'une opération non appropriée/autorisée fait fermer le programme?* Non
8. *Le travail d'une session est-il récupérable après une fermeture non prévue du programme?* Non
9. *L'apprentissage est-il aisé?* Non, car il faut connaitre le langage pour savoir générer des diagrammes
10. *L'interface homme-machine est-elle facilement compréhensible?* Oui
11. *Est-elle attractive?* Non, c'est un design d'un autre âge
12. *Existe-il une version de production?* Oui, la version 5.5 qui est accessible depuis le site internet
13. *Le programme propose-t-il des fonctions de décorations du schéma (couleur, type de noeuds, type de traits)?* Non, aucun contrôle n'est possible sur le graphe
14. *Quelles sont les vues proposées par le programme (graphe, treeview,...)?* Code + image générée pour le graphe
15. *Est-il possible de rajouter un attribut/propriété dans le modèle?* Pas d'attribut ou de propriété sur les noeuds
16. *L'ajout des contraintes est-il possible?* Oui

A.10 XFeature

Scénario 1: Installation

Après avoir essayé de rajouter le plugin depuis le gestionnaire d'extensions, j'ai réussi à installer le plugin en téléchargeant le jar et le copiant dans le répertoire plugin d'Eclipse.

C'est loin d'être facile d'accès.
Au vue du temps passé sur l'installation, les tests sur ce programme ont été abandonnées.

Durée: 3 heures

Scénario 2: Création d'un projet

Non réalisé. Raison: voir scénario 1

Scénario 3: Reproduire le schéma ci-dessous

Non réalisé. Raison: voir scénario 1

Scénario 4: Ressources utilisées

Non réalisé. Raison: voir scénario 1

Scénario 5: Fonctionnalités externes aux processus métiers

Non réalisé. Raison: voir scénario 1

Autres questions

1. *Le programme propose-t-il des extensions par plugin?* Oui, c'est un plugin Eclipse
2. *Le programme est-il portable sur d'autres plateformes (linux,...)?* Oui, partout où Eclipse est supporté
3. *Les fichiers sauvegardés sont-ils utilisables pour d'autres programmes?* Aucune idée, devant l'absence de tests
4. *Le programme est-il encore supporté par une équipe? Bugfixing? Nouveaux développements?* Il existait un bug tracker, il ne répond plus. Le projet n'a plus l'air d'être suivi et maintenu.
5. *Depuis combien de temps le programme existe-t-il?* 2004

6. *Les fonctionnalités du programme comportent-elles encore des bugs?* Aucune idée, devant l'absence de tests

7. *Est-ce qu'une opération non appropriée/autorisée fait fermer le programme?* Aucune idée, devant l'absence de tests

8. *Le travail d'une session est-il récupérable après une fermeture non prévue du programme?* Aucune idée, devant l'absence de tests

9. *L'apprentissage est-il aisé?* Aucune idée, devant l'absence de tests

10. *L'interface homme-machine est-elle facilement compréhensible?* Aucune idée, devant l'absence de tests

11. *Est-elle attractive?* Aucune idée, devant l'absence de tests

12. *Existe-il une version de production?* Aucune idée, devant l'absence de tests

13. *Le programme propose-t-il des fonctions de décorations du schéma (couleur, type de noeuds, type de traits)?* Aucune idée, devant l'absence de tests

14. *Quelles sont les vues proposées par le programme (graphe, treeview,...)?* Aucune idée, devant l'absence de tests

15. *Est-il possible de rajouter un attribut/propriété dans le modèle?* Aucune idée, devant l'absence de tests

16. *L'ajout des contraintes est-il possible?* Aucune idée, devant l'absence de tests

Appendix B

Annexe 2

B.1 Global package

Activator.java

```
package com.tvlViewer;

import org.eclipse.ui.IViewReference;
import org.eclipse.ui.PlatformUI;
import org.eclipse.ui.plugin.AbstractUIPlugin;
import org.osgi.framework.BundleContext;

import com.tvlViewer.Interface.IGraphView;
import com.tvlViewer.Interface.IPropertiesView;
import com.tvlViewer.Interface.IToolView;

/**
 * The activator class controls the plug-in life cycle
 */
public class Activator extends AbstractUIPlugin {

	// The plug-in ID
	public static final String PLUGIN_ID = "com.tvlViewer"; //$NON-
	   NLS-1$

	// The shared instance
	private static Activator plugin;

	/**
	 * The constructor
	 */
	public Activator() {
	}
```

```
/*
 * (non-Javadoc)
 * @see org.eclipse.ui.plugin.AbstractUIPlugin#start(org.osgi.
    framework.BundleContext)
 */
public void start(BundleContext context) throws Exception {
       super.start(context);
       plugin = this;
}

/*
 * (non-Javadoc)
 * @see org.eclipse.ui.plugin.AbstractUIPlugin#stop(org.osgi.
    framework.BundleContext)
 */
public void stop(BundleContext context) throws Exception {
       plugin = null;
       super.stop(context);
}

public IGraphView getGrapheView()
{
       return (IGraphView)GetView(PluginConstants.GrapheViewId)
          ;
}

public IPropertiesView getPropertiesView()
{
       return (IPropertiesView)GetView(PluginConstants.
          PropertiesViewId);
}

private Object GetView(String viewId)
{
       IViewReference[] views = PlatformUI.getWorkbench().
          getActiveWorkbenchWindow().getActivePage().
          getViewReferences();
       for(int i = 0; i < views.length; i++)
       {
                     IViewReference ref = views[i];
                     if(ref.getId().equals(viewId))
                     {
                            return ref.getView(false);
                     }
```

```
                }
                return null;
        }

        public IToolView getToolView()
        {
                return (IToolView)GetView(PluginConstants.ToolViewId);
        }


        /**
         * Returns the shared instance
         *
         * @return the shared instance
         */
        public static Activator getDefault() {
                return plugin;
        }

}
```

Application.java

```
package com.tvlViewer;

import org.eclipse.equinox.app.IApplication;
import org.eclipse.equinox.app.IApplicationContext;
import org.eclipse.swt.widgets.Display;
import org.eclipse.ui.IWorkbench;
import org.eclipse.ui.PlatformUI;

/**
 * This class controls all aspects of the application's execution
 */
public class Application implements IApplication {

        public Object start(IApplicationContext context) {
                Display display = PlatformUI.createDisplay();
                try {
                        int returnCode = PlatformUI.
                           createAndRunWorkbench(display, new
                           ApplicationWorkbenchAdvisor());
                        if (returnCode == PlatformUI.RETURN_RESTART) {
                                return IApplication.EXIT_RESTART;
                        }
                        return IApplication.EXIT_OK;
```

```
                } finally {
                        display.dispose();
                }
        }

        public void stop() {
                if (!PlatformUI.isWorkbenchRunning())
                        return;
                final IWorkbench workbench = PlatformUI.getWorkbench();
                final Display display = workbench.getDisplay();
                display.syncExec(new Runnable() {
                        public void run() {
                                if (!display.isDisposed())
                                        workbench.close();
                        }
                });
        }
}
```

ApplicationActionBarAdvisor.java

```
package com.tvlViewer;

import org.eclipse.ui.actions.ActionFactory.IWorkbenchAction;
import org.eclipse.ui.application.ActionBarAdvisor;
import org.eclipse.ui.application.IActionBarConfigurer;

public class ApplicationActionBarAdvisor extends ActionBarAdvisor {

        private IWorkbenchAction exitAction;
    private IWorkbenchAction aboutAction;
    private IWorkbenchAction newWindowAction;

        public ApplicationActionBarAdvisor(IActionBarConfigurer
            configurer) {
                super(configurer);
                // TODO Auto-generated constructor stub
        }

}
```

ApplicationWorkbenchAdvisor.java

```
package com.tvlViewer;

import org.eclipse.ui.application.IWorkbenchWindowConfigurer;
import org.eclipse.ui.application.WorkbenchAdvisor;
```

```
import org.eclipse.ui.application.WorkbenchWindowAdvisor;

public class ApplicationWorkbenchAdvisor extends WorkbenchAdvisor {

       private static final String PERSPECTIVE_ID ="com.tvlViewer.
          Factory.GraphePerspectiveFactory";

       public WorkbenchWindowAdvisor createWorkbenchWindowAdvisor(
                      IWorkbenchWindowConfigurer configurer) {
              return new ApplicationWorkbenchWindowAdvisor(configurer)
                  ;
       }

       public String getInitialWindowPerspectiveId() {
              return PERSPECTIVE_ID;
       }

}
```

ApplicationWorkbenchWindowAdvisor.java

```
package com.tvlViewer;

import org.eclipse.swt.graphics.Point;
import org.eclipse.ui.application.ActionBarAdvisor;
import org.eclipse.ui.application.IActionBarConfigurer;
import org.eclipse.ui.application.IWorkbenchWindowConfigurer;
import org.eclipse.ui.application.WorkbenchWindowAdvisor;

public class ApplicationWorkbenchWindowAdvisor extends
   WorkbenchWindowAdvisor {

       public ApplicationWorkbenchWindowAdvisor(
          IWorkbenchWindowConfigurer configurer) {
              super(configurer);
       }

       public ActionBarAdvisor createActionBarAdvisor(
                      IActionBarConfigurer configurer) {
              return new ApplicationActionBarAdvisor(configurer);
       }

       //preWindowOpen method configure the initial window size and
          positions.
       public void preWindowOpen() {
```

```
                IWorkbenchWindowConfigurer configurer =
                   getWindowConfigurer();
                configurer.setInitialSize(new Point(400, 300));
                configurer.setShowCoolBar(false);
                configurer.setShowStatusLine(false);
                configurer.setTitle("RCP Application");
        }
}
```

InternalGraphics.java

```
package com.tvlViewer;

import org.eclipse.draw2d.Graphics;

public enum InternalGraphics
{
        LineCustom("Line custom",Graphics.LINE_CUSTOM),
        LineDash("Line dash",Graphics.LINE_DASH),
        LineDashDot("Line dash dot",Graphics.LINE_DASHDOT),
        LineDashDotDot("Line dash dot dot",Graphics.LINE_DASHDOTDOT),
        LineDot("Line dot",Graphics.LINE_DOT),
        LineSolid("Line solid",Graphics.LINE_SOLID);

        private String name = "";
        private int value = -1;

        InternalGraphics(String name, int value)
        {
                this.name = name;
                this.value = value;
        }

        public String getName()
        {
                return name;
        }

        public int getValue()
        {
                return value;
        }
}
```

JoinType.java

```
package com.tvlViewer;
```

```
public enum JoinType {

	OR("Or",0),
	AND("And", 1),
	XOR("Xor",2);

	private String name = "";
	private int value = -1;

	JoinType(String name, int value)
	{
		this.name = name;
		this.value = value;
	}

	public String getName()
	{
		return name;
	}

	public int getValue()
	{
		return value;
	}
}
```

ModelMarshal.java

```
package com.tvlViewer;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Collections;

import org.eclipse.emf.common.util.URI;
import org.eclipse.emf.ecore.resource.Resource;
import org.eclipse.emf.ecore.resource.ResourceSet;
import org.eclipse.emf.ecore.util.EcoreUtil;

import com.google.inject.Injector;

import be.ac.fundp.info.TVLStandaloneSetupGenerated;
import be.ac.fundp.info.tVL.Model;
```

```
public class ModelMarshal {

	/**
	 * Parses the TVL string <code>s</code> and returns the
	    corresponding instance of the TVL <code>Model</code>
	 *
	 * @throws IOException If error in the underlying file system
	 * @throws NullPointerException If parameters are <code>null</
	    code>
	 * @param s The <code>String</code> specifying the TVL model
	 * @return The instance of the TVL <code>Model</code> specified
	     in <code>s</code>
	 */
	public static Model load(String s) throws IOException {

		// Prepare temporary TVL file
		File f;
		FileWriter fw;
		f = File.createTempFile("temp_model", ".tvl");
		fw = new FileWriter(f);
		fw.write(s);
		fw.close();

		return ModelMarshal.load(f);
	}

	/**
	 * Parses the TVL file <code>f</code> and returns the
	    corresponding instance of the TVL <code>Model</code>
	 *
	 * @throws IOException If error in the underlying file system
	 * @throws NullPointerException If parameters are <code>null</
	    code>
	 * @param f The <code>File</code> specifying the TVL model
	 * @return The instance of the TVL <code>Model</code> specified
	     in <code>f</code>
	 */
	public static Model load(File f) throws IOException {

		// Get the URI of the model file
		URI fileURI = URI.createFileURI(f.getPath());

		return ModelMarshal.load(fileURI);
	}
```

```
/**
 * Parses the TVL file at <code>uri</code> and returns the
    corresponding instance of the TVL <code>Model</code>
 *
 * @throws IOException If error in the underlying file system
 * @throws NullPointerException If parameters are <code>null</
    code>
 * @param uri The <code>URI</code> where the TVL model can be
    found
 * @return The instance of the TVL <code>Model</code> specified
     at <code>uri</code>
 */
public static Model load(URI uri) throws IOException {
       // Create the resource set
       Injector injector = new TVLStandaloneSetupGenerated()
                     .createInjectorAndDoEMFRegistration();
       ResourceSet resourceSet = injector.getInstance(
          ResourceSet.class);

       // Load the actual file from URI
       Resource res = resourceSet.getResource(uri, true);
       EcoreUtil.resolveAll(res);

       // In Xtext a Resource always has just one root element
       if (res.getContents().size()>0)
              return (Model) res.getContents().get(0);
       else
              throw new IOException("Could not load TVL model
                 .");
}

/**
 * Saves the model <code>m</code> in the file <code>filePath</
    code>
 *
 * @param m The instance of <code>Model</code> to be saved
 * @param filePath The pathname of the file in which the model
    will be saved
 * @throws IOException If error in the underlying file system
 * @throws NullPointerException If parameters are <code>null</
    code>
 * @see java.io.File#File(java.lang.String)
 */
public static void save(Model m, String filePath) throws
   IOException {
```

```
        ModelMarshal.save(m, new File(filePath));
    }


    /**
     * Saves the model <code>m</code> in the file <code>f</code>
     *
     * @param m The instance of <code>Model</code> to be saved
     * @param f The <code>File</code> in which the model will be
        saved
     * @throws IOException If error in the underlying file system
     * @throws NullPointerException If parameters are <code>null</
        code>
     */
    public static void save(Model m, File f) throws IOException {
        ModelMarshal.save(m, URI.createFileURI(f.getPath()));
    }

    /**
     * Saves the model <code>m</code> to the URI <code>uri</code>
     *
     * @param m The instance of <code>Model</code> to be saved
     * @param uri The <code>URI</code> at which the model will be
        saved
     * @throws IOException If error in the underlying file system
     * @throws NullPointerException If parameters are <code>null</
        code>
     */
    public static void save(Model m, URI uri) throws IOException {
        m.eResource().setURI(uri);
        // the map parameter contains options. usually left
            empty
        m.eResource().save(Collections.EMPTY_MAP);
    }

    public static void saveXMI(Model m) throws IOException {
        URI xmiuri = m.eResource().getURI().trimFileExtension().
            appendFileExtension("xmi");
        Resource xmiResource = m.eResource().getResourceSet().
            createResource(xmiuri);
        xmiResource.getContents().add(m);
        xmiResource.save(Collections.EMPTY_MAP);
    }

    /*public static void main (String[] args) throws IOException {
```

```
final String usage =
               "Usage: java ModelMarshal [command] file.
                  tvl\n" +
               " with command being one of:\n" +
               " -p simply parse the input tvl file\n" +
               " -w parse and write back the input tvl
                  file (overwriting the original file)\n
                  " +
               " -x parse the input tvl file and write
                  the corresponding xmi file\n";

boolean parse = false, write = false, xmiwrite = false;
String filename = null;

if(args.length == 0) {
       System.out.println(usage);
       System.exit(1);
} else {
       for (String arg : args) {
              switch(arg) {
                     case "-p":
                            parse = true;
                            break;
                     case "-w":
                            write = true;
                            break;
                     case "-x":
                            xmiwrite = true;
                            break;
                     default:
                            if (filename == null)
                               filename = arg;
                            else {
                                   exitWithError("Wrong
                                       arguments\n\n"+
                                      usage);
                            }
                            break;
              }
       }

       if (!(parse ^ write ^ xmiwrite) || filename ==
          null) {
              exitWithError("Wrong arguments\n\n"+usage
                 );
```

```
                }
            }

            Model m = null;

            File tvlfile = new File(filename);

            if(parse) {
                m = load(tvlfile);
                System.out.println("File '"+m.eResource().getURI
                    ()+"' parsed");
            } else if (write) {
                m = load(tvlfile);
                System.out.println("File '"+m.eResource().getURI
                    ()+"' parsed");
                save(m, filename);
                System.out.println("File '"+m.eResource().getURI
                    ()+"' saved");
            } else if (xmiwrite) {
                m = load(tvlfile);
                System.out.println("File '"+m.eResource().getURI
                    ()+"' parsed");
                saveXMI(m);
                System.out.println("File '"+m.eResource().getURI
                    ()+"' saved");
            } else {
                exitWithError("Wrong arguments\n\n"+usage);
            }
        }*/

        private static void exitWithError(String msg) {
            System.out.println(msg);
            System.exit(1);
        }
}
```

PluginConstants.java

```
package com.tvlViewer;

public final class PluginConstants {

        public static String GrapheViewId = "com.tvlViewer.grapheViewId
            ";
        public static String ToolViewId = "com.tvlViewer.toolViewId";
```

```
        public static String PropertiesViewId = "com.tvlViewer.
           propertiesViewId";

        public static final int CONNECTION_CIRCLE = 1 << 6;
}
```

ToolMode.java

```
package com.tvlViewer;


public enum ToolMode
{
        MOUSE, NODE, CONNECTION, EXCLUDE

}
```

B.2 View Package

GrapheView.java

```
package com.tvlViewer.View;

import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.ui.IActionBars;
import org.eclipse.ui.part.ViewPart;
import org.eclipse.zest.core.viewers.AbstractZoomableViewer;
import org.eclipse.zest.core.viewers.GraphViewer;
import org.eclipse.zest.core.viewers.IZoomableWorkbenchPart;
import org.eclipse.zest.core.viewers.ZoomContributionViewItem;
import org.eclipse.zest.core.widgets.Graph;
import org.eclipse.zest.core.widgets.GraphItem;
import org.eclipse.zest.core.widgets.GraphNode;
import org.eclipse.zest.core.widgets.ZestStyles;
import org.eclipse.zest.layouts.LayoutStyles;
import org.eclipse.zest.layouts.algorithms.TreeLayoutAlgorithm;

import com.tvlViewer.Activator;
import com.tvlViewer.ToolMode;
import com.tvlViewer.Component.JoinGraphNode;
import com.tvlViewer.Factory.GrapheConnectionFactory;
import com.tvlViewer.Factory.GrapheNodeFactory;
import com.tvlViewer.Interface.IGraphView;
import com.tvlViewer.Interface.IToolView;
```

```
import com.tvlViewer.Listener.GraphKeyListener;
import com.tvlViewer.Listener.GraphMouseListener;
import com.tvlViewer.Listener.GraphSelectionListener;

public class GrapheView extends ViewPart implements IGraphView,
   IZoomableWorkbenchPart {

       private GraphViewer graphViewer;
       private GraphNode startNode;

       @Override
       public void createPartControl(Composite parent) {
              this.graphViewer = new GraphViewer(parent, SWT.NONE);

              final Graph graph = this.graphViewer.getGraphControl();

              final GraphNode node1 = new GraphNode(graph, ZestStyles.
                 NONE, "un");
              final GraphNode node2 = new GraphNode(graph, ZestStyles.
                 NONE, "deux");
              final GraphNode node3 = new GraphNode(graph, ZestStyles.
                 NONE, "trois");
              final GraphNode node4 = new GraphNode(graph, ZestStyles.
                 NONE, "quatre");
              final GraphNode node5 = new GraphNode(graph, ZestStyles.
                 NONE, "cinq");
              final GraphNode node6 = new GraphNode(graph, ZestStyles.
                 NONE, "six");
              final GraphNode node7 = new GraphNode(graph, ZestStyles.
                 NONE, "sept");
              final GraphNode node8 = new GraphNode(graph, ZestStyles.
                 NONE, "huit");
              final GraphNode node9 = new GraphNode(graph, ZestStyles.
                 NONE, "neuf");

              this.CreateNode(node1, node2);
              this.CreateNode(node1, node3);
              this.CreateNode(node7, node4);
              this.CreateNode(node2, node5);
              this.CreateNode(node2, node6);
              this.CreateNode(node3, node7);
              this.CreateNode(node3, node8);
              this.CreateNode(node3, node9);
              this.CreateNode(node4, node2);
```

```
        graph.addMouseListener(new GraphMouseListener(graph));
        graph.addKeyListener(new GraphKeyListener(graphViewer));
        graph.addSelectionListener(new GraphSelectionListener())
            ;

    graph.setLayoutAlgorithm(new TreeLayoutAlgorithm(
       LayoutStyles.NO_LAYOUT_NODE_RESIZING), true);
    graph.applyLayout();

        ZoomContributionViewItem zoomItem = new
           ZoomContributionViewItem(this);
        IActionBars bars = getViewSite().getActionBars();
    bars.getMenuManager().add(zoomItem);

}

@Override
public void showNode(int nodeId) {
       // TODO Auto-generated method stub

}

@Override
public void CreateNode(int x, int y) {
       GrapheNodeFactory.addBasicNode(this.graphViewer.
          getGraphControl(), x, y);
}

@Override
public void CreateConnection(GraphNode node)
{
       //Object obj = graph.getFigureAt(x,y);
       if(node != null)
       {
              if(this.startNode == null)
              {
                     this.startNode = node;
              }
              else
              {
                     CreateNode(this.startNode, node);
                     this.startNode = null;

                     IToolView toolView = Activator.getDefault
                        ().getToolView();
```

```
                        if(toolView != null && toolView.
                           GetSelectedMode() == ToolMode.
                           CONNECTION)
                        {
                                toolView.setMode(ToolMode.MOUSE);
                        }
                }
        }
}

public void CreateNode(GraphNode startNode, GraphNode endNode)
{
        GraphNode join = null;
        if(startNode instanceof JoinGraphNode)
        {
                join = startNode;
                startNode = ((JoinGraphNode)startNode).
                   getParentNode();
        }
        else
        {
                join = GrapheNodeFactory.addJoinNode(this.
                   graphViewer.getGraphControl(), startNode.
                   getLocation().x, startNode.getLocation().y +
                   30, "<1 - 1>", startNode);
                GrapheConnectionFactory.addParentConnection(this
                   .graphViewer.getGraphControl(), startNode,
                   join);
        }
        GrapheConnectionFactory.addJoinConnection(this.
           graphViewer.getGraphControl(), join, endNode);
}

@Override
public void setFocus() {
        Graph graph = this.graphViewer.getGraphControl();
        graph.setFocus();

}

@Override
public AbstractZoomableViewer getZoomableViewer() {
        return this.graphViewer;
}
```

```
        @Override
        public void removeSelection(GraphItem item) {
                if(item.equals(this.startNode))
                {
                        this.startNode = null;
                }
        }

        @Override
        public void CreateExclusion(GraphNode node) {
                if(node instanceof JoinGraphNode)
                        return;

                if(startNode != null && startNode instanceof
                   JoinGraphNode)
                        return;

                if(this.startNode == null)
                {
                        this.startNode = node;
                }
                else
                {
                        GrapheConnectionFactory.addExclusionConnection(
                           this.graphViewer.getGraphControl(), this.
                           startNode, node);

                        this.startNode = null;

                        IToolView toolView = Activator.getDefault().
                           getToolView();
                        if(toolView != null && toolView.GetSelectedMode
                           () == ToolMode.EXCLUDE)
                        {
                                toolView.setMode(ToolMode.MOUSE);
                        }
                }

        }
}
```

PropertiesView.java

```
package com.tvlViewer.View;

import org.eclipse.jface.viewers.TableViewer;
```

```
import org.eclipse.jface.viewers.TableViewerColumn;
import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Table;
import org.eclipse.ui.part.ViewPart;

import com.tvlViewer.Component.ExcludeGraphConnection;
import com.tvlViewer.Interface.IPropertiesView;
import com.tvlViewer.Provider.GrapheContentProvider;
import com.tvlViewer.Provider.GrapheLabelProvider;
import com.tvlViewer.editing.PropertyEditingSupport;

public class PropertiesView extends ViewPart implements
   IPropertiesView {

       private TableViewer viewer;
       private GrapheLabelProvider provider;
       private PropertyEditingSupport editingSupport;

       @Override
       public void createPartControl(Composite parent) {
              viewer = new TableViewer(parent, SWT.FULL_SELECTION |
                 SWT.H_SCROLL);
              Table table = viewer.getTable();
              this.editingSupport = new PropertyEditingSupport(viewer)
                 ;
              this.createColumns(viewer);
              table.setHeaderVisible(true);
              table.setLinesVisible(true);
           viewer.setContentProvider(new GrapheContentProvider());
           this.provider = new GrapheLabelProvider();
           viewer.setLabelProvider(provider);
       }

       @Override
       public void setFocus() {
              viewer.getTable().setFocus();

       }

       @Override
       public void updateProperties(Object obj) {
              if(obj instanceof ExcludeGraphConnection)
                     obj = null;
```

```
                this.editingSupport.setSource(obj);
                this.provider.setSource(obj);
                viewer.setInput(obj);
        }

        private void createColumns(TableViewer table)
        {
                TableViewerColumn column = new TableViewerColumn(table,
                   SWT.NONE);
                column.getColumn().setText("Property");
            column.getColumn().setWidth(200);
            //column.setResizable(true);
            //column.setMoveable(true);

                column = new TableViewerColumn(table, SWT.NONE);
                column.getColumn().setText("Value");
            column.getColumn().setWidth(200);
            column.setEditingSupport(this.editingSupport);
            //column.setEditingSupport()
            //column.setResizable(true);
            //column.setMoveable(true);
        }
}
```

ToolView.java

```
package com.tvlViewer.View;

import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.ToolBar;
import org.eclipse.swt.widgets.ToolItem;
import org.eclipse.ui.part.ViewPart;

import com.tvlViewer.ToolMode;
import com.tvlViewer.Interface.IToolView;

public class ToolView extends ViewPart implements IToolView {

        //TableViewer viewer;
        ToolItem btnMouse;
        ToolItem btnNode;
        ToolItem btnConnection;
        ToolItem btnExclude;
        private boolean isShiftKey;
```

```
@Override
public void createPartControl(Composite parent) {
	//CoolBar coolBar = new CoolBar(parent, SWT.BORDER);
	final ToolBar toolBar = new ToolBar(parent, SWT.FLAT);

	btnMouse = new ToolItem(toolBar, SWT.RADIO);
	btnMouse.setText("Mouse");
	btnMouse.setSelection(true);

	btnNode = new ToolItem(toolBar, SWT.RADIO);
	btnNode.setText("Node");

	btnConnection = new ToolItem(toolBar, SWT.RADIO);
	btnConnection.setText("Connection");

	btnExclude = new ToolItem(toolBar, SWT.RADIO);
	btnExclude.setText("Exclude");
}

@Override
public void setFocus() {
	// TODO Auto-generated method stub

}

public ToolMode GetSelectedMode()
{
	if(btnNode.getSelection())
		return ToolMode.NODE;
	else if (btnConnection.getSelection())
		return ToolMode.CONNECTION;
	else if (btnExclude.getSelection())
		return ToolMode.EXCLUDE;
	return ToolMode.MOUSE;
}

@Override
public void setMode(ToolMode mode) {
	if(!this.isShiftKey)
	{
		btnNode.setSelection(mode.equals(ToolMode.NODE))
		    ;
		btnConnection.setSelection(mode.equals(ToolMode.
		    CONNECTION));
```

```
                        btnMouse.setSelection(mode.equals(ToolMode.MOUSE
                            ));
                        btnExclude.setSelection(mode.equals(ToolMode.
                            EXCLUDE));
                }
        }

        @Override
        public void setShiftKey(boolean value) {
                this.isShiftKey = value;
        }
}
```

B.3 User Action Package

ToolAction.java

```
package com.tvlViewer.userAction;

import org.eclipse.jface.action.Action;
import org.eclipse.ui.IWorkbenchWindow;

public class ToolAction extends Action {
        ToolAction(String text, IWorkbenchWindow window)
        {
                super(text);
                setId(ICommandIds.CMD_OPEN_MESSAGE);
        }
}
```

B.4 Provider Package

ToolViewLabelProvider.java

```
package com.tvlViewer.Provider;

import org.eclipse.jface.viewers.ITableLabelProvider;
import org.eclipse.jface.viewers.LabelProvider;
import org.eclipse.swt.graphics.Image;
import org.eclipse.ui.views.properties.IPropertyDescriptor;

public class ToolViewLabelProvider extends LabelProvider implements
        ITableLabelProvider {
```

```
        @Override
        public Image getColumnImage(Object element, int columnIndex) {
                // TODO Auto-generated method stub
                return null;
        }

        @Override
        public String getColumnText(Object element, int columnIndex) {
                return element.toString();
        }
}
```

ToolViewContentProvider.java

```
package com.tvlViewer.Provider;

import java.util.ArrayList;

import org.eclipse.jface.viewers.IStructuredContentProvider;
import org.eclipse.jface.viewers.Viewer;

import com.tvlViewer.ToolMode;

public class ToolViewContentProvider implements
    IStructuredContentProvider {

        @Override
        public void dispose() {
                // TODO Auto-generated method stub

        }

        @Override
        public void inputChanged(Viewer viewer, Object oldInput, Object
            newInput) {
                // TODO Auto-generated method stub

        }

        @Override
        public Object[] getElements(Object inputElement) {
                return ((ArrayList)inputElement).toArray(new ToolMode
                    [0]);
        }

}
```

GrapheContentProvider.java

```
package com.tvlViewer.Provider;

import org.eclipse.jface.viewers.IStructuredContentProvider;
import org.eclipse.jface.viewers.Viewer;
import org.eclipse.ui.views.properties.IPropertySource;

import com.tvlViewer.Factory.GrapheNodeAdapterFactory;

public class GrapheContentProvider implements
   IStructuredContentProvider {

       @Override
       public void dispose() {
              // TODO Auto-generated method stub

       }

       @Override
       public void inputChanged(Viewer viewer, Object oldInput, Object
           newInput) {
       }

       @Override
       public Object[] getElements(Object inputElement) {
              GrapheNodeAdapterFactory factory = new
                 GrapheNodeAdapterFactory();
              return ((IPropertySource) factory.getAdapter(
                 inputElement, IPropertySource.class)).
                 getPropertyDescriptors();
       }

}
```

GrapheLabelProvider.java

```
package com.tvlViewer.Provider;

import org.eclipse.jface.viewers.ITableLabelProvider;
import org.eclipse.jface.viewers.LabelProvider;
import org.eclipse.swt.graphics.Image;
import org.eclipse.ui.views.properties.IPropertyDescriptor;
import org.eclipse.ui.views.properties.IPropertySource;
import org.eclipse.zest.core.widgets.GraphNode;

import com.tvlViewer.Adapter.PropertyValueAdapter;
```

```
import com.tvlViewer.Factory.GrapheNodeAdapterFactory;

public class GrapheLabelProvider extends LabelProvider implements
		ITableLabelProvider {

	private IPropertySource source;

	@Override
	public Image getColumnImage(Object element, int columnIndex) {
		// TODO Auto-generated method stub
		return null;
	}

	@Override
	public String getColumnText(Object element, int columnIndex) {
		if(!(element instanceof IPropertyDescriptor))
		{
			return "";
		}

		IPropertyDescriptor desc = (IPropertyDescriptor)element;
		switch(columnIndex)
		{
		case 0 :
			return desc.getDisplayName();
		case 1:
		{
			if(source != null)
			{
				return PropertyValueAdapter.getValue(
				   source.getPropertyValue(desc.getId()))
				   ;
			}
		}
		default:
			break;
		}
		return "";
	}

	public void setSource(Object source)
	{
		if(source != null && source instanceof IPropertySource)
			this.source = (IPropertySource) source;
		else
```

```
                        this.source = (IPropertySource) new
                           GrapheNodeAdapterFactory().getAdapter(source,
                            IPropertySource.class);
        }
}
```

B.5 Listener Package

GraphSelectionListener.java

```
package com.tvlViewer.Listener;

import org.eclipse.swt.events.SelectionAdapter;
import org.eclipse.swt.events.SelectionEvent;
import org.eclipse.zest.core.widgets.GraphConnection;
import org.eclipse.zest.core.widgets.GraphNode;

import com.tvlViewer.Activator;
import com.tvlViewer.ToolMode;
import com.tvlViewer.Interface.IToolView;

public class GraphSelectionListener extends SelectionAdapter {

        @Override
          public void widgetSelected(SelectionEvent e) {
                if(e.item instanceof GraphNode || e.item instanceof
                   GraphConnection)
                {
                        Activator.getDefault().getPropertiesView().
                           updateProperties(e.item);
                        IToolView toolView = Activator.getDefault().
                           getToolView();
                        if(toolView != null
                                        && e.item instanceof GraphNode)
                        {
                                if( toolView.GetSelectedMode() ==
                                   ToolMode.CONNECTION)
                                        Activator.getDefault().
                                           getGrapheView().
                                           CreateConnection((GraphNode)e.
                                           item);
                                else if (toolView.GetSelectedMode() ==
                                   ToolMode.EXCLUDE)
                                        Activator.getDefault().
```

```
                                        getGrapheView().CreateExclusion
                                        ((GraphNode)e.item);
                        }
                }
                else
                        Activator.getDefault().getPropertiesView().
                           updateProperties(null);
        }
}
```

GraphMouseListener.java

```
package com.tvlViewer.Listener;

import org.eclipse.swt.events.MouseEvent;
import org.eclipse.swt.events.MouseListener;
import org.eclipse.zest.core.widgets.Graph;

import com.tvlViewer.Activator;
import com.tvlViewer.ToolMode;
import com.tvlViewer.Interface.IGraphView;
import com.tvlViewer.Interface.IToolView;


public class GraphMouseListener implements MouseListener {

        private Graph graph;

        public GraphMouseListener(Graph graph)
        {
                this.graph = graph;
        }

        @Override
        public void mouseDoubleClick(MouseEvent e) {
                // TODO Auto-generated method stub

        }

        @Override
        public void mouseDown(MouseEvent e) {
        }

        @Override
        public void mouseUp(MouseEvent e) {
```

```
                IToolView toolView = Activator.getDefault().getToolView
                    ();
                if(toolView != null)
                {
                        ToolMode mode = toolView.GetSelectedMode();
                        if(mode == ToolMode.NODE)
                        {
                                IGraphView grapheView = Activator.
                                    getDefault().getGrapheView();
                                if(grapheView != null)
                                {
                                        grapheView.CreateNode(e.x, e.y);
                                }
                                toolView.setMode(ToolMode.MOUSE);
                        }
                }

        }

}
```

GraphKeyListener.java

```
package com.tvlViewer.Listener;

import org.eclipse.swt.SWT;
import org.eclipse.swt.events.KeyEvent;
import org.eclipse.swt.events.KeyListener;
import org.eclipse.zest.core.viewers.GraphViewer;
import org.eclipse.zest.core.widgets.GraphItem;

import com.tvlViewer.Activator;
import com.tvlViewer.Component.ParentGraphConnection;
import com.tvlViewer.Interface.IGraphView;
import com.tvlViewer.Interface.IToolView;

public class GraphKeyListener implements KeyListener {

        private GraphViewer _graph;

        public GraphKeyListener(GraphViewer graph)
        {
                this._graph = graph;
        }

        @Override
```

```
public void keyPressed(KeyEvent e) {
        if(e.keyCode == SWT.SHIFT)
        {
                IToolView toolView = Activator.getDefault().
                   getToolView();
                if(toolView != null)
                {
                        toolView.setShiftKey(true);
                }
        }
}

@Override
public void keyReleased(KeyEvent e) {
        if(e.keyCode == SWT.DEL)
        {
                Object[] selections = this._graph.
                   getGraphControl().getSelection().toArray();

                int count = selections.length;
                IGraphView graphView = Activator.getDefault().
                   getGrapheView();
                for(int i= 0; i < count; i++)
                {
                        if(selections[i] instanceof
                           ParentGraphConnection)
                        {
                                if(graphView != null)
                                        graphView.removeSelection
                                            (((ParentGraphConnection
                                            )selections[i]).
                                            getDestination());
                                ((ParentGraphConnection)selections
                                    [i]).getDestination().dispose()
                                    ;
                        }
                        if(graphView != null)
                                graphView.removeSelection((
                                   GraphItem)selections[i]);
                        ((GraphItem)selections[i]).dispose();

                }
        }
        else if(e.keyCode == SWT.SHIFT)
        {
```

```
                        IToolView toolView = Activator.getDefault().
                           getToolView();
                        if(toolView != null)
                        {
                                toolView.setShiftKey(false);
                        }
                }

        }

}
```

B.6 Interface Package

IToolView.java

```
package com.tvlViewer.Interface;

import com.tvlViewer.ToolMode;

public interface IToolView {
        ToolMode GetSelectedMode();
        void setMode(ToolMode mode);
        void setShiftKey(boolean value);
}
```

IPropertiesView.java

```
package com.tvlViewer.Interface;

public interface IPropertiesView {
        void updateProperties(Object obj);
}
```

IGraphView.java

```
package com.tvlViewer.Interface;

import org.eclipse.zest.core.widgets.GraphItem;
import org.eclipse.zest.core.widgets.GraphNode;

public interface IGraphView {
        void showNode(int nodeId);
        void CreateNode(int x, int y);
        void CreateConnection(GraphNode node);
        void removeSelection(GraphItem item);
```

```
        void CreateExclusion(GraphNode item);
}
```

B.7 Input Package

GraphEditorInput.java

```
package com.tvlViewer.Input;

import org.eclipse.jface.resource.ImageDescriptor;
import org.eclipse.ui.IEditorInput;
import org.eclipse.ui.IPersistableElement;

public class GraphEditorInput implements IEditorInput {

        private final int id;

    public GraphEditorInput(int id) {
        this.id = id;
    }

        @Override
        public Object getAdapter(Class adapter) {
                return null;
        }

        @Override
        public boolean exists() {
                return true;
        }

        @Override
        public ImageDescriptor getImageDescriptor() {
                return null;
        }

        @Override
        public String getName() {
                return String.valueOf(id);
        }

        @Override
        public IPersistableElement getPersistable() {
                return null;
```

```
	}

	@Override
	public String getToolTipText() {
		return "New filename";
	}

}
```

B.8 Factory Package

GraphePerspectiveFactory.java

```
package com.tvlViewer.Factory;

import org.eclipse.ui.IPageLayout;
import org.eclipse.ui.IPerspectiveFactory;

import com.tvlViewer.PluginConstants;

public class GraphePerspectiveFactory implements IPerspectiveFactory {

	@Override
	public void createInitialLayout(IPageLayout layout) {
		layout.setEditorAreaVisible(false);
		String editorArea = layout.getEditorArea();

		layout.addView(PluginConstants.GrapheViewId, IPageLayout
		    .TOP, 0.5f, editorArea);
		layout.addView(PluginConstants.PropertiesViewId,
		    IPageLayout.LEFT, 0.5f, editorArea);
		layout.addView(PluginConstants.ToolViewId, IPageLayout.
		    LEFT, 0.5f, editorArea);
	}

}
```

GrapheNodeFactory.java

```
package com.tvlViewer.Factory;

import org.eclipse.swt.SWT;
import org.eclipse.zest.core.widgets.GraphNode;
import org.eclipse.zest.core.widgets.IContainer;
```

```
import com.tvlViewer.Component.JoinGraphNode;

public class GrapheNodeFactory {
	private int nodeCount;

	private GrapheNodeFactory()
	{
		this.nodeCount = 1;
	}

	private static GrapheNodeFactory instance = null;
	private static GrapheNodeFactory getInstance()
	{
		if(instance == null)
			instance = new GrapheNodeFactory();
		return instance;
	}

	public static GraphNode addBasicNode(IContainer parent, int x,
	    int y)
	{
		GraphNode node = new GraphNode(parent, SWT.NONE, "Node"
		    + getInstance().nodeCount++);
		node.setLocation(x, y);
		return node;
	}
	public static GraphNode addJoinNode(IContainer parent, int x,
	    int y, String text, GraphNode parentNode)
	{
		JoinGraphNode node = new JoinGraphNode(parent, SWT.NONE,
		     text);
		node.setLocation(x, y);
		node.setParentNode(parentNode);
		return node;
	}
}
```

GrapheNodeAdapterFactory.java

```
package com.tvlViewer.Factory;

import org.eclipse.core.runtime.IAdapterFactory;
import org.eclipse.ui.views.properties.IPropertySource;
import org.eclipse.zest.core.widgets.GraphConnection;
import org.eclipse.zest.core.widgets.GraphNode;
```

```
import com.tvlViewer.Adapter.GraphConnectionPropertySourceAdapter;
import com.tvlViewer.Adapter.GraphNodePropertySourceAdapter;
import com.tvlViewer.Adapter.JoinGraphConnectionPropertySourceAdapter;
import com.tvlViewer.Adapter.JoinGraphNodePropertySourceAdapter;
import com.tvlViewer.Component.ExcludeGraphConnection;
import com.tvlViewer.Component.JoinGraphConnection;
import com.tvlViewer.Component.JoinGraphNode;

public class GrapheNodeAdapterFactory implements IAdapterFactory {

	private static final Class[] TYPES = { IPropertySource.class};
	@Override
	public Object getAdapter(Object adaptableObject, Class
	   adapterType) {
	    if(adapterType == IPropertySource.class) {
			if(adaptableObject instanceof JoinGraphNode)
			{
				return new
				   JoinGraphNodePropertySourceAdapter((
				   JoinGraphNode)adaptableObject);
			}
			else if(adaptableObject instanceof GraphNode)
			{
				return new GraphNodePropertySourceAdapter
				   ((GraphNode)adaptableObject);
			}
			else if(adaptableObject instanceof
			   JoinGraphConnection)
			{
				return new
				   JoinGraphConnectionPropertySourceAdapter
				   ((JoinGraphConnection)adaptableObject)
				   ;
			}
			else if(adaptableObject instanceof
			   ExcludeGraphConnection)
			{
				return null;
			}
			else if(adaptableObject instanceof
			   GraphConnection)
			{
				return new
				   GraphConnectionPropertySourceAdapter((
				   GraphConnection)adaptableObject);
```

```
                }
            }
            return null;
        }

        @Override
        public Class[] getAdapterList() {
            // TODO Auto-generated method stub
            return TYPES;
        }

}
```

GrapheConnectionFactory.java

```
package com.tvlViewer.Factory;

import org.eclipse.zest.core.widgets.Graph;
import org.eclipse.zest.core.widgets.GraphNode;
import org.eclipse.zest.core.widgets.ZestStyles;

import com.tvlViewer.Component.ExcludeGraphConnection;
import com.tvlViewer.Component.JoinGraphConnection;
import com.tvlViewer.Component.ParentGraphConnection;

public class GrapheConnectionFactory {
        private int connCount;

        private GrapheConnectionFactory()
        {
            this.connCount = 1;
        }

        private static GrapheConnectionFactory instance = null;
        private static GrapheConnectionFactory getInstance()
        {
            if(instance == null)
                instance = new GrapheConnectionFactory();
            return instance;
        }

        public static void addParentConnection(Graph parent, GraphNode
           start, GraphNode end)
        {
            new ParentGraphConnection(parent, ZestStyles.
               CONNECTIONS_DIRECTED, start, end);
```

```
                //conn.setText("Connection" + getInstance().connCount++)
                    ;
        }

        public static void addJoinConnection(Graph parent, GraphNode
            start, GraphNode end)
        {
                new JoinGraphConnection(parent, ZestStyles.
                    CONNECTIONS_DIRECTED, start, end);
                //conn.setText("Connection" + getInstance().connCount++)
                    ;
        }

        public static void addExclusionConnection(Graph parent,
                        GraphNode startNode, GraphNode endNode) {
                new ExcludeGraphConnection(parent, startNode, endNode);

        }
}
```

B.9 editing Package

PropertyEditingSupport.java

```
package com.tvlViewer.editing;

import org.eclipse.jface.viewers.CellEditor;
import org.eclipse.jface.viewers.EditingSupport;
import org.eclipse.jface.viewers.TableViewer;
import org.eclipse.jface.viewers.TextCellEditor;
import org.eclipse.ui.views.properties.IPropertyDescriptor;
import org.eclipse.ui.views.properties.IPropertySource;
import org.eclipse.ui.views.properties.PropertyDescriptor;

import com.tvlViewer.Adapter.PropertyValueAdapter;
import com.tvlViewer.Factory.GrapheNodeAdapterFactory;

public class PropertyEditingSupport extends EditingSupport {
        private TableViewer viewer;
        private IPropertySource source;

        public PropertyEditingSupport(TableViewer viewer) {
                super(viewer);
                this.viewer = viewer;
```

```
        // TODO Auto-generated constructor stub
    }

    @Override
    protected CellEditor getCellEditor(Object element) {
        if(element instanceof PropertyDescriptor)
        {
            PropertyDescriptor prop = (PropertyDescriptor)
               element;
            if(prop.getId().equals("LineStyle"))
                return new LineStyleComboBoxCellEditor(
                   viewer.getTable());
            else if(prop.getId().equals("Type"))
                return new JoinTypeComboBoxCellEditor(
                   viewer.getTable());
            else if(prop.getId().equals("Optional"))
                return new InternalCheckBoxCellEditor(
                   viewer.getTable());
            else if(prop.getId().equals("MaxCardinality"))
                return new CardinalityComboBoxCellEditor(
                   viewer.getTable());
            else if(prop.getId().equals("MinCardinality"))
                return new CardinalityComboBoxCellEditor(
                   viewer.getTable());
            return new TextCellEditor(viewer.getTable());
        }
        return null;
    }

    @Override
    protected boolean canEdit(Object element) {
        // TODO Auto-generated method stub
        return true;
    }

    @Override
    protected Object getValue(Object element) {

        if(element instanceof IPropertyDescriptor)
        {
            IPropertyDescriptor desc = (IPropertyDescriptor)
               element;
            return PropertyValueAdapter.getValue(this.source
               .getPropertyValue(desc.getId()));
        }
```

```
                return "";
        }

        @Override
        protected void setValue(Object element, Object value) {
                this.source.setPropertyValue(element, value);
                this.viewer.refresh(element);
        }

        public void setSource(Object source)
        {
                if(source != null)
                {
                        if(source instanceof IPropertySource)
                                this.source = (IPropertySource) source;
                        else
                                this.source = (IPropertySource) new
                                   GrapheNodeAdapterFactory().getAdapter(
                                   source, IPropertySource.class);
                }
        }

}
```

LineStyleComboBoxCellEditor.java

```
package com.tvlViewer.editing;

import org.eclipse.jface.viewers.ComboBoxCellEditor;
import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Table;

import com.tvlViewer.InternalGraphics;

public class LineStyleComboBoxCellEditor extends ComboBoxCellEditor {
        public LineStyleComboBoxCellEditor(Table table)
        {
                super(table, getStyles(), SWT.READ_ONLY);
        }

        private static String[] getStyles()
        {
                return new String[]
                        {
                                InternalGraphics.LineCustom.getName(),
                                InternalGraphics.LineDash.getName(),
```

```
                        InternalGraphics.LineDashDot.getName(),
                        InternalGraphics.LineDashDotDot.getName()
                            ,
                        InternalGraphics.LineDot.getName(),
                        InternalGraphics.LineSolid.getName()
                };
        }

        protected void doSetValue(Object value)
        {
                for(int i = 0; i<this.getItems().length; i++)
                {
                        if(this.getItems()[i].equals(value))
                        {
                                super.doSetValue(i);
                                break;
                        }
                }
        }

        protected Object doGetValue()
        {
                return this.getItems()[(int)super.doGetValue()];
                /*String name = super.doGetValue().toString();
                InternalZestStyleAdapter adapter = new
                   InternalZestStyleAdapter();
                int val = adapter.ConvertStringToInt(name);
                if(val >-1)
                {
                        return val;
                }
                return "";*/
        }


}
```

JoinTypeComboBoxCellEditor.java

```
package com.tvlViewer.editing;

import org.eclipse.jface.viewers.ComboBoxCellEditor;
import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Table;

import com.tvlViewer.JoinType;
```

```
public class JoinTypeComboBoxCellEditor extends ComboBoxCellEditor {
	public JoinTypeComboBoxCellEditor(Table table)
	{
		super(table, getTypes(), SWT.READ_ONLY);
	}

	private static String[] getTypes()
	{
		return new String[]
			{
				JoinType.OR.getName(),
				JoinType.AND.getName(),
				JoinType.XOR.getName()
			};
	}

	protected void doSetValue(Object value)
	{
		for(int i = 0; i<this.getItems().length; i++)
		{
			if(this.getItems()[i].equals(value))
			{
				super.doSetValue(i);
				break;
			}
		}
	}

	protected Object doGetValue()
	{
		return this.getItems()[(int)super.doGetValue()];
	}


}
```

InternalCheckBoxCellEditor.java

```
package com.tvlViewer.editing;

import org.eclipse.jface.viewers.CheckboxCellEditor;
import org.eclipse.swt.widgets.Table;

public class InternalCheckBoxCellEditor extends CheckboxCellEditor{
```

```
        public InternalCheckBoxCellEditor(Table table) {
                super(table);
        }

        protected void doSetValue(Object value)
        {
                try
                {
                        if(value instanceof Boolean)
                                super.doSetValue(value);
                        else if(value.equals("true"))
                                super.doSetValue(true);
                        else if(value.equals("false"))
                                super.doSetValue(false);
                }
                catch (IllegalArgumentException e)
                {}
        }

        protected Object doGetValue()
        {
                return super.doGetValue();
        }
}
```

CardinalityComboBoxCellEditor.java

```
package com.tvlViewer.editing;

import org.eclipse.jface.viewers.ComboBoxCellEditor;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.KeyEvent;
import org.eclipse.swt.widgets.Table;

public class CardinalityComboBoxCellEditor extends ComboBoxCellEditor
   {
        public CardinalityComboBoxCellEditor(Table table)
        {
                super(table, getVals(), SWT.READ_ONLY);
        }

        private static String[] getVals()
        {
                return new String[]
                        {
                                "0",
```

```
                        "1",
                        "2",
                        "*"
                };
        }

        private String[] getVals(String newValue)
        {
                return new String[]
                        {
                                "0",
                                "1",
                                "2",
                                newValue,
                                "*"
                        };
        }

        protected void doSetValue(Object value)
        {
                Boolean found = false;
                for(int i = 0; i<this.getItems().length; i++)
                {
                        if(this.getItems()[i].equals(value))
                        {
                                super.doSetValue(i);
                                found = true;
                                break;
                        }
                }
                if(!found)
                {
                        String[] vals = getVals(value.toString());
                        this.setItems(vals);
                        super.doSetValue(3);
                }
        }

        protected Object doGetValue()
        {
                return this.getItems()[(int)super.doGetValue()];
        }

        protected void keyReleaseOccured(KeyEvent keyEvent)
        {
```

```
                if(!('0' <= keyEvent.character && keyEvent.character <=
                    '9'))
                    keyEvent.doit = false;
        }
}
```

B.10 Component Package

ParentGraphConnection.java

```
package com.tvlViewer.Component;

import org.eclipse.zest.core.widgets.Graph;
import org.eclipse.zest.core.widgets.GraphConnection;
import org.eclipse.zest.core.widgets.GraphNode;

public class ParentGraphConnection extends GraphConnection{

        public ParentGraphConnection(Graph graphModel, int style,
            GraphNode source,
                        GraphNode destination) {
                super(graphModel, style, source, destination);
        }

}
```

JoinGraphNode.java

```
package com.tvlViewer.Component;

import org.eclipse.draw2d.ColorConstants;
import org.eclipse.zest.core.widgets.GraphNode;
import org.eclipse.zest.core.widgets.IContainer;

import com.tvlViewer.JoinType;

public class JoinGraphNode extends GraphNode {

        private GraphNode parentNode;
        private JoinType joinType;
        private int maxCardinality, minCardinality;


        public JoinGraphNode(IContainer graphModel, int style, String
            text) {
```

```
        super(graphModel, style, text);
        this.setJoinType(JoinType.OR);
        this.maxCardinality = this.minCardinality = 1;

        this.setText();
        // TODO Auto-generated constructor stub
    }

    public GraphNode getParentNode()
    {
        return this.parentNode;
    }

    public void setParentNode(GraphNode node)
    {
        this.parentNode = node;
    }

    public JoinType getJoinType()
    {
        return this.joinType;
    }

    public void setJoinType(JoinType type)
    {
        this.joinType = type;
        this.setBackgroundColor();
    }

    private void setBackgroundColor()
    {
        if(this.joinType == JoinType.XOR)
        {
            this.setBackgroundColor(ColorConstants.orange);
            this.setMinCardinality("0");
            this.setMaxCardinality("1");
        }
        else if(this.joinType == JoinType.AND)
        {
            this.setBackgroundColor(ColorConstants.green);
            this.setMinCardinality("-1");
            this.setMaxCardinality("-1");
        }
        else
        {
```

```
                this.setBackgroundColor(ColorConstants.white);
        }
}

public void setMinCardinality(String s)
{
        if(s.equals("*"))
        {
                this.minCardinality = -1;
                this.setText();
        }
        else
        {
                try
                {
                        int val = Integer.parseInt(s);
                        if(val >= -1)
                                this.minCardinality = val;
                        this.setText();
                }
                catch (NumberFormatException e)
                {}
        }
}

public String getMinCardinality()
{
        String min = this.minCardinality + "";
        if(this.minCardinality == -1)
                min = "*";
        return min;
}

public void setMaxCardinality(String s)
{
        if(s.equals("*"))
        {
                this.maxCardinality = -1;
                this.setText();
        }
        else
        {
                try
                {
                        int val = Integer.parseInt(s);
```

```
                        if(val >= -1)
                                this.maxCardinality = val;
                        this.setText();
                }
                catch (NumberFormatException e)
                {}
        }
    }

    public String getMaxCardinality()
    {
        String max = this.maxCardinality + "";
        if(this.maxCardinality == -1)
                max = "*";
        return max;
    }

    private void setText()
    {
        if(this.joinType == JoinType.AND)
                this.setText("*");
        else
                this.setText("[" + this.getMinCardinality() +
                   ".." + this.getMaxCardinality() + "]");
    }
}
```

JoinGraphConnection.java

```
package com.tvlViewer.Component;

import org.eclipse.zest.core.widgets.Graph;
import org.eclipse.zest.core.widgets.GraphConnection;
import org.eclipse.zest.core.widgets.GraphNode;

public class JoinGraphConnection extends GraphConnection{

    public JoinGraphConnection(Graph graphModel, int style,
       GraphNode source,
                GraphNode destination) {
        super(graphModel, style, source, destination);
    }
}
```

ExcludeGraphConnection.java

```
package com.tvlViewer.Component;
```

```
import org.eclipse.draw2d.Connection;
import org.eclipse.draw2d.Graphics;
import org.eclipse.draw2d.PolygonDecoration;
import org.eclipse.draw2d.PolylineConnection;
import org.eclipse.zest.core.widgets.Graph;
import org.eclipse.zest.core.widgets.GraphConnection;
import org.eclipse.zest.core.widgets.GraphNode;
import org.eclipse.zest.core.widgets.ZestStyles;

public class ExcludeGraphConnection extends GraphConnection{

	public ExcludeGraphConnection(Graph graphModel, GraphNode
	   source,
			GraphNode destination) {
		super(graphModel, ZestStyles.CONNECTIONS_DIRECTED,
		   source, destination);
		this.setLineStyle(Graphics.LINE_DASH);

		Connection figure = this.getConnectionFigure();
		((PolylineConnection)figure).setSourceDecoration(new
		   PolygonDecoration());
		this.setText("<exclude>");
	}
}
```

B.11 Adapter Package

PropertyValueAdapter.java

```
package com.tvlViewer.Adapter;

import org.eclipse.swt.graphics.Color;


public class PropertyValueAdapter {
	public static String getValue(Object obj)
	{
		if( obj == null)
			return "";
		Class cl = obj.getClass();
		if(cl.isEnum())
		{
			Enum e = Enum.valueOf(cl, cl.getName());
```

```
                        return e.toString();
                }
                else if(obj instanceof Color)
                {
                        Color color = (Color)obj;
                        return color.toString();
                }


                return obj.toString();
        }
}
```

JoinTypeAdapter.java

```
package com.tvlViewer.Adapter;

import com.tvlViewer.JoinType;

public class JoinTypeAdapter {

        public String ConvertIntToString(int val)
        {
                for(JoinType g : JoinType.values())
                {
                        if(g.getValue() == val)
                                return g.getName();
                }
                return "";
        }

        public JoinType ConvertStringToType(String name)
        {
                for(JoinType g : JoinType.values())
                {
                        if(g.getName() == name)
                                return g;
                }
                return JoinType.AND;
        }
}
```

JoinGraphNodePropertySourceAdapter.java

```
package com.tvlViewer.Adapter;

import java.util.ArrayList;
```

```
import java.util.List;

import org.eclipse.ui.views.properties.IPropertyDescriptor;
import org.eclipse.ui.views.properties.IPropertySource;
import org.eclipse.ui.views.properties.PropertyDescriptor;

import com.tvlViewer.Component.JoinGraphNode;

public class JoinGraphNodePropertySourceAdapter implements
   IPropertySource {

       private JoinGraphNode node;

       public JoinGraphNodePropertySourceAdapter(JoinGraphNode node)
       {
              this.node = node;
       }

       @Override
       public Object getEditableValue() {
              return this;
       }

       @Override
       public IPropertyDescriptor[] getPropertyDescriptors() {
              List<PropertyDescriptor> descriptors = new ArrayList<
                 PropertyDescriptor>();

              PropertyDescriptor descriptorText = new
                 PropertyDescriptor("Type", "Type");
              descriptorText.setAlwaysIncompatible(true);
              descriptors.add(descriptorText);

              PropertyDescriptor descriptorMaxCardinality = new
                 PropertyDescriptor("MaxCardinality", "Max
                 Cardinality");
              descriptorMaxCardinality.setAlwaysIncompatible(true);
              descriptors.add(descriptorMaxCardinality);

              PropertyDescriptor descriptorMinCardinality = new
                 PropertyDescriptor("MinCardinality", "Min
                 Cardinality");
              descriptorMinCardinality.setAlwaysIncompatible(true);
              descriptors.add(descriptorMinCardinality);
```

```
        return descriptors.toArray(new IPropertyDescriptor[0]);
    }

    @Override
    public Object getPropertyValue(Object id) {

        if(id.toString().equals("Text"))
                return this.node.getText();
        else if(id.toString().equals("Type"))
        {
                return this.node.getJoinType().getName();
        }
        else if(id.toString().equals("MaxCardinality"))
        {
                return this.node.getMaxCardinality();
        }
        else if(id.toString().equals("MinCardinality"))
        {
                return this.node.getMinCardinality();
        }
        return null;
    }

    @Override
    public boolean isPropertySet(Object id) {
        // TODO Auto-generated method stub
        return true;
    }

    @Override
    public void resetPropertyValue(Object id) {
        // TODO Auto-generated method stub

    }

    @Override
    public void setPropertyValue(Object id, Object value) {
        if(value == null || id == null)
                return;

        String prop = "";
        if(id instanceof IPropertyDescriptor)
                prop = ((IPropertyDescriptor)id).getDisplayName
                    ();
        else
```

```
                        prop = id.toString();

                if(prop.equals("Text"))
                {
                        this.node.setText(value.toString());
                }
                else if(prop.equals("Type"))
                {
                        JoinTypeAdapter adapter = new JoinTypeAdapter();
                        this.node.setJoinType(adapter.
                           ConvertStringToType(value.toString()));
                }
                else if(prop.equals("Max Cardinality"))
                {
                        this.node.setMaxCardinality(value.toString());
                }
                else if(prop.equals("Min Cardinality"))
                {
                        this.node.setMinCardinality(value.toString());
                }
        }
}
```

JoinGraphConnectionPropertySourceAdapter.java

```
package com.tvlViewer.Adapter;

import java.util.ArrayList;
import java.util.List;

import org.eclipse.draw2d.ColorConstants;
import org.eclipse.draw2d.Connection;
import org.eclipse.draw2d.PolylineConnection;
import org.eclipse.draw2d.PolylineDecoration;
import org.eclipse.draw2d.geometry.PointList;
import org.eclipse.ui.views.properties.IPropertyDescriptor;
import org.eclipse.ui.views.properties.IPropertySource;
import org.eclipse.ui.views.properties.PropertyDescriptor;
import com.tvlViewer.PluginConstants;
import com.tvlViewer.Component.JoinGraphConnection;

public class JoinGraphConnectionPropertySourceAdapter implements
    IPropertySource {

        private JoinGraphConnection conn;

```

```
public JoinGraphConnectionPropertySourceAdapter(
   JoinGraphConnection conn)
{
       this.conn = conn;
}

@Override
public Object getEditableValue() {
       return this;
}

@Override
public IPropertyDescriptor[] getPropertyDescriptors() {
       List<PropertyDescriptor> descriptors = new ArrayList<
          PropertyDescriptor>();

       PropertyDescriptor optionalText = new PropertyDescriptor
          ("Optional", "Optional");
       optionalText.setAlwaysIncompatible(true);
       descriptors.add(optionalText);

       return descriptors.toArray(new IPropertyDescriptor[0]);
}

@Override
public Object getPropertyValue(Object id) {
       if (id.toString() == "Optional")
              return (Boolean)((this.conn.getConnectionStyle()
                   & PluginConstants.CONNECTION_CIRCLE) ==
                  PluginConstants.CONNECTION_CIRCLE);
       return null; }

@Override
public boolean isPropertySet(Object id) {
       // TODO Auto-generated method stub
       return false;
}

@Override
public void resetPropertyValue(Object id) {
       // TODO Auto-generated method stub

}

@Override
```

```
public void setPropertyValue(Object id, Object value) {
        if(value == null || id == null)
                return;

        String prop = "";
        if(id instanceof IPropertyDescriptor)
                prop = ((IPropertyDescriptor)id).getId().
                   toString();
        else
                prop = id.toString();

        if(prop == "Optional")
        {
                if((boolean)value)
                {
                        PointList pl = this.getCircleList();
                        PolylineDecoration circle = new
                           PolylineDecoration();
                        circle.setBackgroundColor(ColorConstants.
                           black);
                        //circle.setLineWidth(3);
                        circle.setScale(3,3);
                        circle.setTemplate(pl);
                        Connection figure = this.conn.
                           getConnectionFigure();
                        ((PolylineConnection)figure).
                           setTargetDecoration(circle);
                        this.conn.setConnectionStyle(this.conn.
                           getConnectionStyle() | PluginConstants
                           .CONNECTION_CIRCLE);
                }
                else
                {
                        Connection figure = this.conn.
                           getConnectionFigure();
                        ((PolylineConnection)figure).
                           setTargetDecoration(null);
                        this.conn.setConnectionStyle(this.conn.
                           getConnectionStyle() & ~
                           PluginConstants.CONNECTION_CIRCLE);
                }
        }
}

private PointList getCircleList()
```

```
{
        PointList pl = new PointList();
        /*pl.addPoint(-6, 0);
        pl.addPoint(-5, 2);
        pl.addPoint(-4, 3);
        pl.addPoint(-3, 3);
        pl.addPoint(-2, 3);
        pl.addPoint(-1, 2);
        pl.addPoint(-0, 0);
        pl.addPoint(-1, -2);
        pl.addPoint(-2, -3);
        pl.addPoint(-3, -3);
        pl.addPoint(-4, -3);
        pl.addPoint(-5, -2);
        pl.addPoint(-6, 0);*/
        pl.addPoint(-2,0);
        pl.addPoint(-1,1);
        pl.addPoint(-0,0);
        pl.addPoint(-1,-1);
        pl.addPoint(-2,0);

        /*int r = 3;
        int x0 = -3, y0 = 0;
        int x = 0;
        int y = r;
        int d = r - 1;
        while (y>=x)
        {
                pl.addPoint(x0 + x, y0 + y);
                pl.addPoint(x0 + x, y0 - y);
                pl.addPoint(x0 - x, y0 + y);
                pl.addPoint(x0 - x, y0 - y);
                pl.addPoint(x0 + y, y0 + x);
                pl.addPoint(x0 + y, y0 - x);
                pl.addPoint(x0 - y, y0 + x);
                pl.addPoint(x0 - y, y0 - x);
                if (d >= 2*x)
                {
            d = d-2*x-1;
            x = x+1;
                }
                else if(d < 2*(r-y))
                {
        d = d+2*y-1;
        y = y-1;
```

```
                        }
                else
                {
                d = d+2*(y-x-1);
                y = y-1;
                x = x+1;
                }
                }*/
                return pl;
        }
}
```

InternalGraphicsAdapter.java

```
package com.tvlViewer.Adapter;

import com.tvlViewer.InternalGraphics;

public class InternalGraphicsAdapter {

        public String ConvertIntToString(int val)
        {
                for(InternalGraphics g : InternalGraphics.values())
                {
                        if(g.getValue() == val)
                                return g.getName();
                }
                return "";
        }

        public int ConvertStringToInt(String name)
        {
                for(InternalGraphics g : InternalGraphics.values())
                {
                        if(g.getName() == name)
                                return g.getValue();
                }
                return -1;
        }
}
```

GraphNodePropertySourceAdapter.java

```
package com.tvlViewer.Adapter;

import java.util.ArrayList;
import java.util.List;
```

```
import org.eclipse.ui.views.properties.IPropertyDescriptor;
import org.eclipse.ui.views.properties.PropertyDescriptor;
import org.eclipse.ui.views.properties.IPropertySource;
import org.eclipse.zest.core.widgets.GraphNode;

public class GraphNodePropertySourceAdapter implements IPropertySource
    {

       private GraphNode node;

       public GraphNodePropertySourceAdapter(GraphNode node)
       {
              this.node = node;
       }

       @Override
       public Object getEditableValue() {
              return this;
       }

       @Override
       public IPropertyDescriptor[] getPropertyDescriptors() {
              List<PropertyDescriptor> descriptors = new ArrayList<
                 PropertyDescriptor>();
              PropertyDescriptor descriptorText = new
                 PropertyDescriptor("Text", "Text");
              descriptorText.setAlwaysIncompatible(true);
              descriptors.add(descriptorText);

              return descriptors.toArray(new IPropertyDescriptor[0]);
       }

       @Override
       public Object getPropertyValue(Object id) {
              if(id.toString() == "Text")
                     return this.node.getText();
              return null;
       }

       @Override
       public boolean isPropertySet(Object id) {
              // TODO Auto-generated method stub
              return true;
       }
```

```
	@Override
	public void resetPropertyValue(Object id) {
		// TODO Auto-generated method stub

	}

	@Override
	public void setPropertyValue(Object id, Object value) {
		if(value == null || id == null)
			return;

		String prop = "";
		if(id instanceof IPropertyDescriptor)
			prop = ((IPropertyDescriptor)id).getDisplayName
			    ();
		else
			prop = id.toString();

		if(prop == "Text")
		{
			this.node.setText(value.toString());
		}
	}
}
```

GraphConnectionPropertySourceAdapter.java

```
package com.tvlViewer.Adapter;

import java.util.ArrayList;
import java.util.List;

import org.eclipse.ui.views.properties.IPropertyDescriptor;
import org.eclipse.ui.views.properties.IPropertySource;
import org.eclipse.ui.views.properties.PropertyDescriptor;
import org.eclipse.zest.core.widgets.GraphConnection;

public class GraphConnectionPropertySourceAdapter implements
    IPropertySource {

	private GraphConnection conn;

	public GraphConnectionPropertySourceAdapter(GraphConnection
	    conn)
	{
```

```
        this.conn = conn;
    }

    @Override
    public Object getEditableValue() {
        return this;
    }

    @Override
    public IPropertyDescriptor[] getPropertyDescriptors() {
        List<PropertyDescriptor> descriptors = new ArrayList<
            PropertyDescriptor>();

        PropertyDescriptor descriptorText = new
            PropertyDescriptor("Text", "Text");
        descriptorText.setAlwaysIncompatible(true);
        descriptors.add(descriptorText);
        //PropertyDescriptor descriptorLineStyle = new
            PropertyDescriptor("LineStyle", "Line style");
        //descriptorLineStyle.setAlwaysIncompatible(true);
        //descriptors.add(descriptorLineStyle);

        return descriptors.toArray(new IPropertyDescriptor[0]);
    }

    @Override
    public Object getPropertyValue(Object id) {
        if(id.toString() == "Text")
                return this.conn.getText();
        else if(id.toString() == "LineStyle")
        {
                InternalGraphicsAdapter adapter = new
                    InternalGraphicsAdapter();
                return adapter.ConvertIntToString(this.conn.
                    getLineStyle());
        }
        return null; }

    @Override
    public boolean isPropertySet(Object id) {
        // TODO Auto-generated method stub
        return false;
    }

    @Override
```

```
        public void resetPropertyValue(Object id) {
                // TODO Auto-generated method stub

        }

        @Override
        public void setPropertyValue(Object id, Object value) {
                if(value == null || id == null)
                        return;

                String prop = "";
                if(id instanceof IPropertyDescriptor)
                        prop = ((IPropertyDescriptor)id).getId().
                           toString();
                else
                        prop = id.toString();

                if(prop == "Text")
                {
                        this.conn.setText(value.toString());
                }
                else if(prop == "LineStyle")
                {
                        InternalGraphicsAdapter adapter = new
                           InternalGraphicsAdapter();
                        this.conn.setLineStyle(adapter.
                           ConvertStringToInt(value.toString()));
                }
        }
}
```

B.12 Projects Files

plugin.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.4"?>
<plugin>
   <extension
         point="org.eclipse.ui.views">
      <view
            category="com.tvlViewer.category"
            class="com.tvlViewer.View.GrapheView"
            id="com.tvlViewer.grapheViewId"
```

```
         name="Graphe View"
         restorable="true">
   </view>
   <category
         id="com.tvlViewer.category"
         name="Graphe">
   </category>
   <view
         category="com.tvlViewer.category"
         class="com.tvlViewer.View.ToolView"
         id="com.tvlViewer.toolViewId"
         name="Tool View"
         restorable="true">
   </view>
   <view
         category="com.tvlViewer.category"
         class="com.tvlViewer.View.PropertiesView"
         id="com.tvlViewer.propertiesViewId"
         name="Property View"
         restorable="true">
   </view>
</extension>
<extension
      point="org.eclipse.ui.perspectives">
   <perspective
         class="com.tvlViewer.Factory.GraphePerspectiveFactory"
         id="com.tvlViewer.perspective"
         name="Graphe Perspective">
   </perspective>
</extension>
<extension
      point="org.eclipse.core.runtime.adapters">
   <factory
         adaptableType="org.eclipse.zest.core.widgets.GraphNode"
         class="com.tvlViewer.GraphNodeAdapterFactory">
      <adapter
            type="org.eclipse.ui.views.properties.IPropertySource">
      </adapter>
   </factory>
</extension>
     <extension
    point="org.eclipse.core.contenttype.contentTypes">
            <content-type
      file-extensions="tvl"
      id="be.ac.fundp.info.tvl.contenttypeMod"
```

```
        name="TVL File"
        priority="normal">
       </content-type>
       </extension>
 <extension
      id="id1"
      point="org.eclipse.core.runtime.applications">
    <application
         cardinality="singleton-global"
         thread="main"
         visible="true">
    </application>
 </extension>
</plugin>
```

.project

```
<?xml version="1.0" encoding="UTF-8"?>
<projectDescription>
       <name>com.plugin.project</name>
       <comment></comment>
       <projects>
       </projects>
       <buildSpec>
              <buildCommand>
                     <name>org.eclipse.jdt.core.javabuilder</name>
                     <arguments>
                     </arguments>
              </buildCommand>
              <buildCommand>
                     <name>org.eclipse.pde.ManifestBuilder</name>
                     <arguments>
                     </arguments>
              </buildCommand>
              <buildCommand>
                     <name>org.eclipse.pde.SchemaBuilder</name>
                     <arguments>
                     </arguments>
              </buildCommand>
       </buildSpec>
       <natures>
              <nature>org.eclipse.pde.PluginNature</nature>
              <nature>org.eclipse.jdt.core.javanature</nature>
       </natures>
</projectDescription>
```

.classpath

```
<?xml version="1.0" encoding="UTF-8"?>
<classpath>
	<classpathentry exported="true" kind="lib" path="org.eclipse.
	   emf.common_2.8.0.v20130125-0546.jar"/>
	<classpathentry exported="true" kind="lib" path="org.eclipse.
	   emf.ecore_2.8.3.v20130125-0546.jar"/>
	<classpathentry exported="true" kind="lib" path="org.eclipse.
	   emf.ecore.xmi_2.8.1.v20130125-0546.jar"/>
	<classpathentry exported="true" kind="lib" path="com.google.
	   guava_10.0.1.v201203051515.jar"/>
	<classpathentry exported="true" kind="lib" path="org.eclipse.
	   core.resources_3.8.101.v20130717-0806.jar"/>
	<classpathentry exported="true" kind="lib" path="com.google.
	   inject_3.0.0.v201203062045.jar"/>
	<classpathentry exported="true" kind="lib" path="javax.inject_1
	   .0.0.v20091030.jar"/>
	<classpathentry exported="true" kind="lib" path="org.antlr.
	   runtime_3.2.0.v201101311130.jar"/>
	<classpathentry exported="true" kind="lib" path="org.apache.
	   log4j_1.2.15.v201012070815.jar"/>
	<classpathentry exported="true" kind="lib" path="be.ac.fundp.
	   info.tvl.editor_0.2.0.jar"/>
	<classpathentry kind="con" path="org.eclipse.jdt.launching.
	   JRE_CONTAINER/org.eclipse.jdt.internal.debug.ui.launcher.
	   StandardVMType/JavaSE-1.7"/>
	<classpathentry kind="src" path="src"/>
	<classpathentry kind="con" path="org.eclipse.pde.core.
	   requiredPlugins"/>
	<classpathentry kind="output" path="bin"/>
</classpath>
```