Université Paul Sabatier – Toulouse III
IUT A - Toulouse Rangueil
Projet tuteuré #20

Antoine de Roquemaurel
Mathieu Soum
Geoffroy Subias
Marie-Ly Tang
Groupe B

Pour Monsieur Max Chevalier (Responsable projets)
Monsieur Thierry Millan (Client)
Madame Caroline Kross (Tutrice)

Analyse et conception

Bibliothèque d'objets graphiques UML

Toulouse, le 18 janvier 2012

Table des matières

1 Analyse **3**

2 Conception **4**

2.1 Architecture générale du projet 4

2.1.1 Package eltGraphique 5

Package eltModelisation 5

Package ligne 5

2.1.2 Package diagramme 5

2.1.3 Conception détaillée 5

Chapitre 1

Analyse

Chapitre 2

Conception

2.1 Architecture générale du projet

L'architecture est la base d'une conception telle que nous l'avons choisie. En utilisant la notation UML, nous sommes parvenu à élaborer un "meta-modèle" de cette notation mais en isolant uniquement l'aspect graphique, faisant ainsi abstraction des multipes règles de conception que la norme UML impose. Après discussion avec le client, de nombreuses modifications ont été apportées à l'archetecture d'origine pour aboutir à une solution stable. Cette nouvelle architecture se compose de vingt classes, réparties en quatre paquetages comme suit :

- **eltGraphique**
  - *ElementGraphique*
  - **eltModelisation**
    - *Acteur*
    - ActeurActif
    - ActeurPassif
    - Attribut
    - CasUtilisation
    - Classe
    - *ElementModelisation*
    - Interface
    - Methode
    - Traitement
    - Visibilite
    - Variable
  - **ligne**
    - Cardinalite
    - Lien
    - MessageTraitement
    - TypeLien
- **diagramme**
  - Diagramme
  - DiagrammeCasUtilisation
  - DiagrammeClasse
  - DiagrammeSequence

Dans cet arbre représentant notre architecture, on peut voir certains noms **en gras** ou *en italique*. Les noms **en gras** représentent les différents paquetages que nous avons séparés. Ceux *en italiques* sont des classes abstraites crées afin de factoriser le code dans l'optique de réaliser une programmation objet optimale et de suivre les objectifs de propreté du code imposés par le client.

Dans un premier temps, nous avons séparer les diagrammes des éléments graphiques. En effet, un diagramme sera composé de toute sorte d'éléments graphiques. Puis nous avons découper ces derniers en deux, isolant ainsi les lignes des elements de modélisation tels que les classes, les traitements ou les cas d'utilisation.

ElementGraphique et *ElementModelisation* sont des classes abstraites car elle regroupe les fonctions communes à toues les lasses de leur paquetage respectifs sans pour autant en fournir une implémentation de chacune d'elles – comme par exemple la méthode qui crée la représentation graphique d'un élément ou celle qui permet de le supprimer d'un diagramme.

Prenons maintenant chaque paquetage séparément

2.1.1 Package eltGraphique

Le paquetage eltGraphique regroupe toutes les classes qui représentent des éléments graphiques. Il regroupe la classe *ElementGraphique* et deux paquetages **eltModelisation** et **ligne**. Cette classe possède deux attributs `graph` et `diagramme`, correspondant repectivement au graphe dans lequel sont stockés les éléments et le diagramme afficher à l'écran. Elle comprend également (en plus d'un constructeur initialisant les attributs) les méthodes `supprimer` permettant de supprimer un élément du graphe et du diagramme, ainsi que `creer`, méthode abstraite réimplémentée dans les classes descendentes servant à creer la représentation graphique de l'objet et l'afficher sur le diagramme.

Package eltModelisation Ce paquetage regroupe toutes les classes représentant les différents éléments de modélisation que l'on trouver dans les diagrammes UML de cas d'utilisation, classe et de séquence. *Acteur* est une classe abstraite car les acteur actif et passifs ont beaucoup de caractéristiques identiques mais n'ont pas la même représentation sur un diagramme. Les classes Visibilite, Methode et Attribut permettent au client de créer facilement une interface de saisie de ces éléments, facilitant l'usage du logiciel final. De plus, l'ajout de ces méthodes et attributs dans des classes, des acteurs des traitements ou des interfaces pourra faciliter l'ajout futur de nouvelle fonctionnalités comme apr exemple de la génération de code Java.

Package ligne Ce paquetage regroupe peu de classe. TypeLien est une classe énumérée servant à recenser toutes les types graphiques de liens existant dans la notation UML. Cardinalite quant à elle, permet comme Methode ou Attribut, d'aider le client à permettre un saisie facilité des cardinalités d'un lien dans un diagramme de classe par exemple. La classe Lien comprend la méthode `creer` qui va configurer tous les styles de liens et applique au nouveau lien le style choisi par l'utilisateur. MessageTraitement spécialise Lien, permettant de créer un style de lien particulier aux massages entre traitement dans les diagrammes de séquences.

2.1.2 Package diagramme

La paquetage **diagramme** comprend plusieurs type de diagramme prédéfinis qui sont cas d'utilisation, classe et séquence. Ils descendent de la classe Diagramme. Chaque type de diagramme implémente deux méthode `eltAutorise` et `lienAutorise` qui permettent respectivement d'autoriser ou interdire un type d'élément particulier et un type de lien entre deux types d'éléments particuliers. La classe Diagramme est générique. Par défaut elle autorise tous les éléments et tous les liens. Il est donc possible au client de réimplémnter ses propres méthodes pour créer ses propres règles.

2.1.3 Conception détaillée

La description détaillée des méthodes et des attributs de chaque classe est disponible sous forme papier dans le dossier spécifié à la documentation ou à l'adresse
▷ `http://documentation.joohoo.fr/libUML`