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 22 janvier 2012

Table des matières

Fonctionnement du document

Ce document est un document est un document expliquant notre approche pour développer une bibliothèque d'objets graphiques UML.
Dans ce dossier, vous pourrez repérer diverse notations, cette introduction à pour but de vous expliquer les notations afin que vous puissiez lire en toute sérénité.

Le glossaire

Un mot dans le glossaire à une police particulière, vous pourrez savoir qu'un mot est dans le glossaire lorsque vous repérerez un mot avec la police suivante : leMotDansLeGlossiare. Si vous voyez cette police, vous pouvez donc vous référez à l'annexe A page 11.

Les noms de méthode, d'attribut ou de classe

Les mots se référant à un nom présent dans le code ont une police particulière, une police type "machine à écrire", si vous voyez la police suivante, c'est que c'est un nom de méthode, d'attribut ou de classe : `uneFonction`.

Les notes de bas de page

Nous utilisons régulièrement des notes de bas de pages, pour donner un acronyme pour expliquer plus en détail une notion, ces notes de bas de pages sont un numéro en expostant, vous trouverez la note correspondante en bas de la page courante, comme ceci [1].

Les liens hypertext

Dans le document, nous pouvons faire référence à un lien d'un site web, tous les liens seront donc symbolisés par une petite puce, comme ceci :
▷ `http://monLien.fr/index.html`

1. Ceci est une note de bas de page

Chapitre 1

Spécification des besoins

1.1 Objectifs, champ d'application, limites du système

La demande de ce système émane de Monsieur Thierry MILLAN, enseignant à l'IUT [1] 'A' Toulouse et chercheur à l'IRIT [2], qui nous a demandé la création d'une bibliothèque d'objets graphiques représentant les différents éléments de modélisation de la norme UML [3] 2.0 en suivant un modèle de développement incrémental.

La particularité de cette bibliothèque etant de respecter la norme UML 2.0 ce qui impose des restrictions d'utilisation sur chacun des éléments graphiques utilisés. Cette bibliothèque d'objets graphiques doit comprendre l'ensemble des éléments graphiques composant les principaux diagrammes vue durant les cours de PRL [4] de M. MILLAN afin d'être utilisable pour toutes les étapes de la conception de projets respectant la norme UML 2.0.

Suivant la demande du client, nous avons traité les trois diagrammes principaux de la norme UML 2.0 : diagramme de cas d'utilisation , classe et de séquence . La problématique du projet était à l'origine une étude de faisabilité sur la réalisation de cette bibliothèque. À l'issue de notre projet, nous devrons estimé une durée de développement pour fournir une bibliothèque complète pouvant être facilement intégré dans une application plus importante, telle qu'un éditeur de saisie ou un générateur de code par exemple. De plus, nous devrons trouver une potentielle utilité à un outils fragmenté, contrairement à celui utilisé à l'IUT, où un outil ne permettra de créer qu'un seul type de diagramme et non regrouper tous les éléments de la modélisation UML.

Nous avons choisi de developper notre projet en utilisant la bibliothèque `JGraphX`, bibliothèque graphique contenant une multitude d'éléments permettant par exemple de dessiner facilement des graphes ou des organigrammes. De plus, elle intègre les principaux pictogrammes que l'on retrouve dans les différents diagrammes UML 2.0 tels que les cas d'utilisation (ellipse) ou les acteurs (bonhomme).

1. Institut Universitaire de Technologie
2. Institut de Recherche Informatique de Toulouse
3. Unified Modeling Language
4. Production de logiciel

1.1.1 Fonctionnalités du système

- Permettre de dessiner un diagramme de cas d'utilisatoin
- Permettre de dessiner un diagramme de classe
- Permettre de dessiner un diagramme de séquence
- Permettre de restreindre l'utilisation des éléments graphiques aux seuls diagrammes dans lesquels ils sont utilisés.

1.1.2 Etats de sortie du système

Une bibliothèque est, par définition, un composant indépendant fournissant des méthodes déjà implémentées, facilitant le développement de d'une application plus importante. Notre bibliothèque sera donc intégrée dans d'autres logiciels plus complets. Par conséquent le système n'a pas d'états de sortie à proprement parler.
Néanmoins nous avons développé un démonstrateur parallèlement au projet dans le but de valider l'ensemble des fonctionnalités de notre bibliothèque et d'avoir une idée d'un potentiel rendu final. Ce dernier consiste en une interface graphique qui intègre notre compsant et qui fourni en état de sortie la représentation graphique d'un diagramme simple.

1.2 Besoins non-fonctionnels

En efficacité : Temps de réponse – Le temps de réponse doit être suffisament rapide pour permettre une utilisation fluide du système. Consomation de mémoire – Le système ne doit pas consommer plus de 200 Mo de mémoire vive.

En implémentation : Propreté du code – Le nombre cyclomatique ne doit pas dépasser 15.

En interopérabilité : La capture des erreurs d'entrée-sortie – comment traiter les échecs d'interface électroniques, etc. Le traitement des mauvaises données – import de données, marquer-et-continuer ou arrêt la politique d'importation, etc. Intégrité des données – intégrité référentielle dans tables de base de données et interfaces

En compatibilité : La compatibilité avec des applications utilisant le système en tant que composant – Le système doit être facilement utilisable par des applications extérieurs qui l'utiliserait comme composant. La compatibilité sur des systèmes d'exploitation différents. Le système doit être capable de fonctionner sur les systèmes d'exploitations les plus courant (Windows, Mac OS, Linux).

En ergonomie : Les standards d'ergonomie – la densité d'éléments sur les écrans, la disposition et le flux, les couleurs, l'Interface Utilisateur et les raccourcis clavier doivent correspondre aux normes d'ergonomie. Internationalisation / besoins de localisation - Le système doit pouvoir être utilisé par des utilisateurs de nationalité autre que française.

En documentation : Eléments de documentation requis – Le système doit être suffisament documenter pour être compréhensible rapidement.

1.3 Cas d'utilisations

Cas d'utilisation : Permettre de modéliser un diagramme de classe.
Rôle : Ce cas d'utilisation regroupe toutes les fonctionnalités permettant de dessiner un diagramme de classe. (C'est à dire les fonctionnalités permettant de dessiner les objets graphiques : classe, association, classe-assosiation, agrégation, composition, héritage)
Déclencheur : Ce cas d'utilisation se produit lorsque l'utilisateur utilise une fonctionnalité permettant de dessiner un élément graphique utilisé dans un diagramme de classe.
Entrées : -
Sorties : -
Préconditions : L'utilisateur devra spécifier que son diagramme est un diagramme de classe.
Postconditions : -
Anomalies : -

Cas d'utilisation : Permettre de modéliser un diagramme de séquence.
Rôle : Ce cas d'utilisation regroupe toutes les fonctionnalités permettant de dessiner un diagramme de séquence. (C'est à dire les fonctionnalités permettant de dessiner les objets graphiques : Acteur actif, Acteur passif, Traitement, Message synchrone, message asynchrone et ligne de vie)
Déclencheur : Ce cas d'utilisation se produit lorsque l'utilisateur utilise une fonctionnalité permettant de dessiner une élément graphique utilisé dans un diagramme de séquence.
Entrées : -
Sorties : -
Préconditions : L'utilisateur devra spécifier que son diagramme est un diagramme de séquence.
Postconditions : -
Anomalies : -

Cas d'utilisation : Permettre de modéliser un diagramme de cas d'utilisation.
Rôle : Ce cas d'utilisation regroupe toutes les fonctionnalités permettant de dessiner un diagramme de cas d'utilisation. (C'est à dire les fonctionnalités permettant de dessiner les objets graphiques : Acteur Actif, Acteur passif, Association, Dependance fonctionnelle, Spécialisation)
Déclencheur : Ce cas d'utilisation se produit lorsque l'utilisateur utilise une fonctionnalité permettant de dessiner une élément graphique utilisé dans un diagramme de cas d'utilisation.
Entrées : -
Sorties : -
Préconditions : L'utilisateur devra spécifier que son diagramme est un diagramme de cas d'utilisation.
Postconditions : -
Anomalies : -

Cas d'utilisation : Permettre de restreindre l'utilisation des éléments graphiques aux seuls diagrammes dans lesquels ils sont utilisés.

Rôle : Ce cas d'utilisation regroupe toutes les fonctionnalités permettant de restreindre l'utilisation des éléments graphiques aux seuls diagrammes dans lesquels ils sont utilisés. Permettant ainsi à un utilisateur de la bibliothèque de sélectionner quel diagramme il veut dessiner et de n'avoir la possibilité de dessinner que les éléments graphiques utilisables pour le diagramme selectionné. Evitant ainsi la surcharge de possibilités et donc une utilisation plus simple de la bibliothèque.)

Déclencheur : Ce cas d'utilisation se produit lorsque l'utilisateur choisit un type de diagramme précis à dessiner.

Entrées : -

Sorties : -

Préconditions : -

Postconditions : -

Anomalies : -

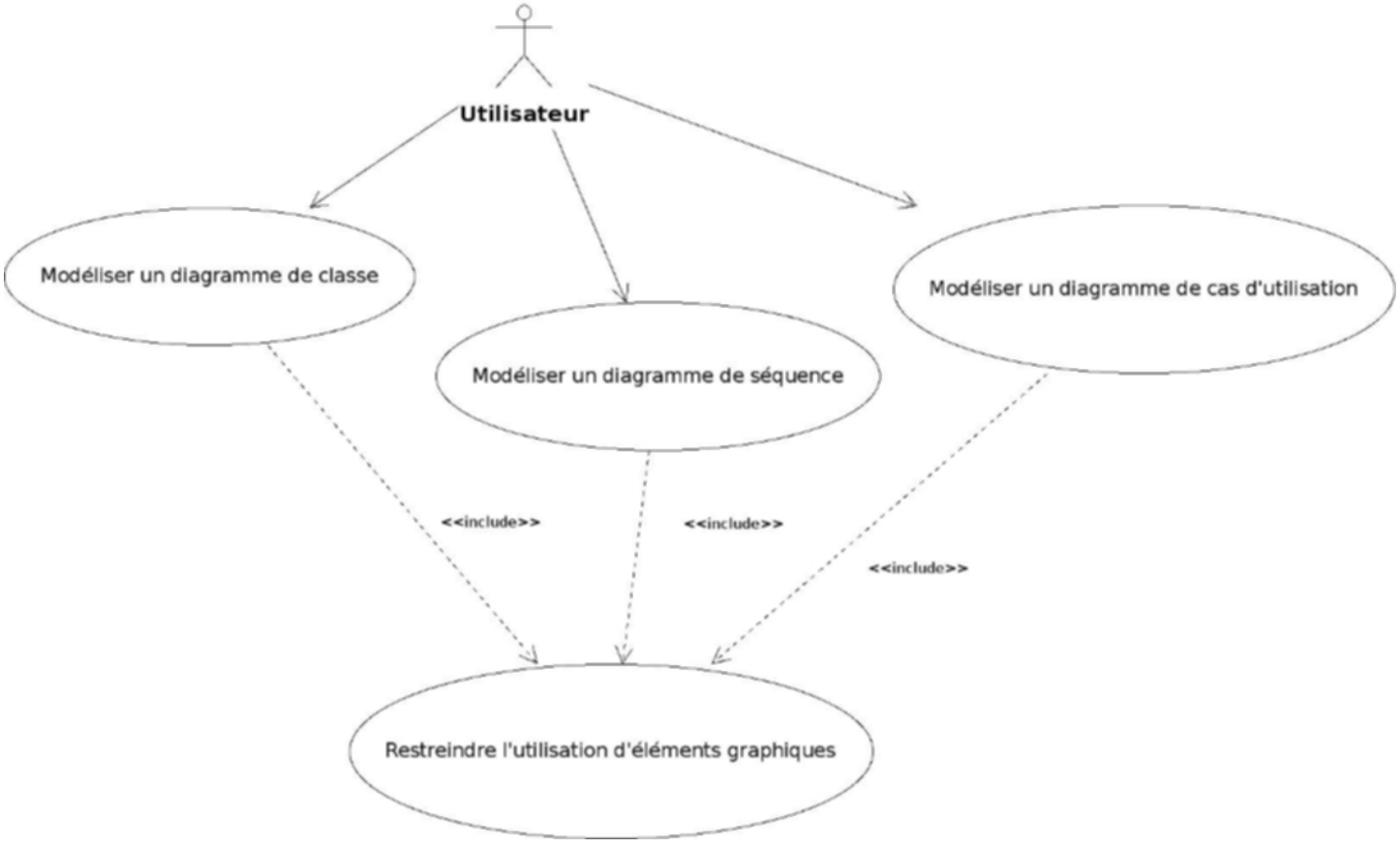

FIGURE 1.1 – Diagramme de cas d'utilisation

Chapitre 2

Conception

2.1 Architecture générale du projet

- **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

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'architecture d'origine pour aboutir à une solution stable. Cette nouvelle architecture se compose de vingt classes, réparties en quatre paquetages comme suit :

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 pour vois comment ils fonctionnent plus en détails.

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 par 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.

2.2 Conception détaillée

2.2.1 Documentation

La description détaillée des méthodes et des attributs de chaque classe est disponible sous forme numérique avec la recette ainsi qu'en trois exemplaires à l'adresse
▷ `http://documentation.joohoo.fr/libUML/public` (Documentation publique et protégée)
▷ `http://documentation.joohoo.fr/libUML/private` (Documentation publique, protégées et privée)
▷ `http://documentation.joohoo.fr/libUML/junitTests` (Tests unitaires)

2.2.2 Relations entre les classes

Le diagramme de paquetage est disponible à l'annexe B page 12. Celui-ci vous permettra de repérer les relations entre les classes.

Annexe A

Glossaire

Bibliothèque (p 5) – un composant indépendant fournissant des méthodes déjà implémentées, facilitant le développement de d'une application plus importante.

Démonstrateur (p 5) – Un démonstrateur est un logiciel simple, permettant de montrer les possibilités d'une bibliothèque

Diagramme de cas d'utilisation (p 4) – Représentation graphique permettant de décrire les intéractions entre les acteurs et le système.

Diagramme de classe (p 4) – Schéma utilisé en génie logiciel pour représenter les classes et les interfaces des systèmes ainsi que les différentes relations entre celles-ci.

Diagramme de séquence (p 4) – Représentation graphique des interactions entre les acteurs et le système selon un ordre chronologique. Ce diagramme est inclus dans la partie dynamique d'UML.

Incrément (p 4) – Fonctionnalité du logiciel, ayant un cycle de developpement lui étant propre (Analyse, Développement, Tests). Cette fonctionnalité doit être opérationnelle pour que l'incrément soit terminé. Il doit améliorer le logiciel par rapport à l'incrément précédent, et ne doit pas altérer les fonctionnalités précédentes.

Java (p 9) – Langage de programmation orienté objet moderne, il compile le programme pour ensuite l'exécuter sur une machine Java, ainsi le programme une fois compilé peut être exécuté sur différentes plateformes (Windows, Linux, Mac OS X, ...).

Modèle de développement incrémental (p 4) – Méthode de développement d'un projet dans laquel le projet final est divisé en plusieurs fonctionnalités appellé incrément qui sont développées et ajouté au projet au fur et à mesure.

Paquetage (p 8) – Un paquetage en Java est un regroupement de classes ayant la même thématique.

UML (p 4) – (Unified Modeling Language) Langage de modélisation graphique à base de pictogramme. Il est apparu dans le monde du génie logiciel dans le cadre de la conception orientée objet. Ce langage est composé de différents diagrammes, allant du développement à la simple analyse des besoins.

Annexe B

Diagramme de paquetage

Bibliothèque UML