

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

Avant-propos

Ce document fait partis d'un groupe de documents permettant de montrer notre approche pour développer une bibliothèque d'objets graphiques UML ¹. UML est un langage permettant de modéliser une application, de la concevoir.

Présentation du groupe

Notre groupe projet est composé de quatre étudiants de deuxième année de DUT Informatique à l'IUT² A de Toulouse, voici la composition de l'équipe :

- Antoine de ROQUEMAUREL
- Mathieu SOUM
- Geoffroy SUBIAS
- Marie-Ly TANG

Nous avons monté ce groupe, car nos compétences sont complémentaires et que nous savons déjà comment chacun travaille.

Antoine de ROQUEMAUREL et Mathieu SOUM sont spécialisés en programmation par objet, Geoffroy SUBIAS est le plus compétent lorsqu'il s'agit de modélisation UML ³ et Marie-Ly TANG s'occupera principalement de l'organisation et de la gestion de projet.

Nos compétences sont différentes mais sont complémentaires pour mener à bien notre projet.

Fonctionnement du document

Ce document est un document expliquant notre approche pour développer une bibliothèque d'objets graphiques UML.

Dans ce dossier, vous pourrez repérer diverses notations, cette introduction a pour but de vous expliquer les notations afin que vous puissiez lire en toute sérénité.

Le glossaire

Un mot dans le glossaire a 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 référer à l'annexe A page 12.

1. Unified Modelling Language
2. Institut Universitaire de Technologie
3. Unified Modelling Language

Les noms de méthode, d'attribut ou de classe

Les mots se référant à un nom présent dans du 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 noms de paquetage

Les noms de paquetage utiliseront une police particulière, afin que l’on puisse les différencier d’une classe ou d’une méthode, vous les trouverez comme suit : `unPaquetage`.

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 exposant, vous trouverez la note correspondante en bas de la page courante, comme ceci⁴.

Les liens hypertextes

Dans le document, nous pouvons faire référence à un lien d’un site web, tous les liens seront donc symbolisés par une petite puce, et une police particulière comme ceci :

▷ `http://monLien.fr/index.html`

4. Ceci est une note de bas de page

Table des matières

1	Spécification des besoins	5
1.1	Objectifs, champ d'application, limites du système	5
1.1.1	Fonctionnalités du système	6
1.1.2	États de sortie du système	6
1.2	Besoins non fonctionnels	6
1.2.1	En efficacité	6
1.2.2	En implémentation	6
1.2.3	En compatibilité	6
1.2.4	En ergonomie	6
1.2.5	En documentation :	7
1.3	Cas d'utilisations	7
2	Conception	10
2.1	Architecture générale du projet	10
2.1.1	Paquetage eltGraphique	11
2.1.2	Paquetage diagramme	11
2.2	Conception détaillée	11
2.2.1	Documentation	11
2.2.2	Relations entre les classes	11
A	Glossaire	12
B	Diagramme de paquetages	13

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'IUT¹ 'A' Toulouse et chercheur à l'IRIT², qui nous a demandé la création d'une bibliothèque d'objets graphiques représentant les différents éléments de modélisation de la norme UML³ 2.0 en suivant un modèle de développement incrémental.

La particularité de cette bibliothèque étant de respecter la norme UML 2.0, cela 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 vus durant les cours de PRL⁴ de M. MILLAN afin d'être utilisable pour toutes les étapes de la conception d'un projet simple.

Suivant la demande du client, nous avons traité les trois diagrammes principaux de la norme UML 2.0 : diagramme de **cas d'utilisation**, de **classes** 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 devons estimer une durée de développement pour fournir une bibliothèque complète pouvant être facilement intégrée dans une application plus importante, tel qu'un éditeur de saisie ou un générateur de code par exemple. De plus, nous devons trouver une potentielle utilité à un outil fragmenté, contrairement à celui utilisé à l'IUT, cet outil permettra de ne créer qu'un seul type de diagramme et non de regrouper tous les éléments de la modélisation UML.

Nous avons choisi de développer 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 (bon-homme).

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

Les fonctionnalités principales attendues par notre client sont les suivantes :

- Dessiner un diagramme de cas d'utilisation
- Dessiner un diagramme de classes
- Dessiner un diagramme de séquence
- Restreindre l'utilisation des éléments graphiques aux seuls diagrammes dans lesquels ils sont utilisés.

1.1.2 États de sortie du système

Une **bibliothèque** est, par définition, un composant indépendant fournissant des méthodes déjà implémentées, facilitant le développement d'une application plus importante. Notre bibliothèque sera donc intégrée dans d'autres logiciels plus complets. Par conséquent le système n'a pas d'état 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 composant et qui fournit en état de sortie la représentation graphique d'un diagramme simple.

1.2 Besoins non fonctionnels

1.2.1 En efficacité

Temps de réponse Le temps de réponse doit être suffisamment rapide pour permettre une utilisation fluide du système.

Consommation de mémoire Le système ne doit pas consommer plus de 200 Mio de mémoire vive.

1.2.2 En implémentation

Propreté du code Le nombre cyclomatique ne doit pas dépasser 15.

1.2.3 En compatibilité

La compatibilité avec des applications utilisant le système en tant que composant Le système doit être facilement utilisable par des applications extérieures qui l'utiliseraient comme composant.

La compatibilité sur des systèmes d'exploitation différents Le système doit être capable de fonctionner sur les systèmes d'exploitations les plus courants (Windows, Mac OS, Linux).

1.2.4 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.

1.2.5 En documentation :

Éléments de documentation requis Le système doit être suffisamment documenté 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-association, 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 un 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, Dépendance fonctionnelle, Spécialisation)

Déclencheur : Ce cas d'utilisation se produit lorsque l'utilisateur utilise une fonctionnalité permettant de dessiner un 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 dessiner que les éléments graphiques utilisables pour le diagramme sélectionné. Évitant 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 : -

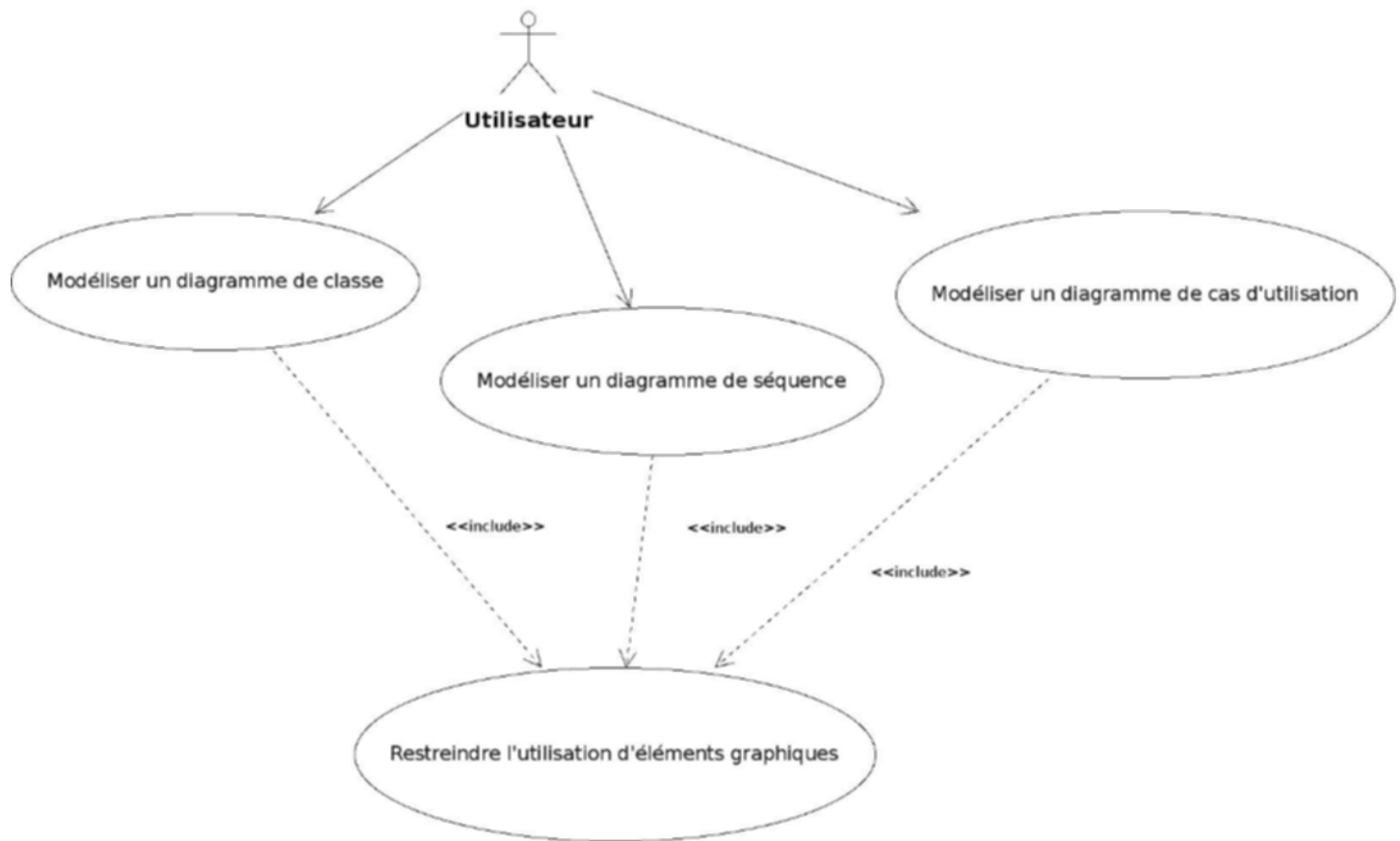
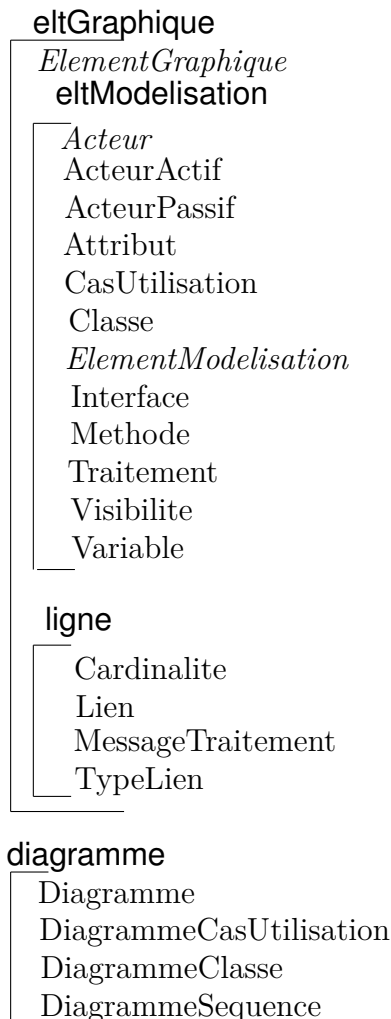


FIGURE 1.1 – Diagramme de cas d'utilisation de libUML

Chapitre 2

Conception

2.1 Architecture générale du projet



L'architecture est la base d'un projet comme le notre, car il sera repris et amélioré par d'autres étudiants. En utilisant la notation UML, nous sommes parvenus à élaborer un "méta modèle" de cette notation mais en isolant uniquement l'aspect graphique, faisant ainsi abstraction des multiples 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**.

Dans cet arbre représentant notre architecture, on peut voir certains noms écrits en **gras**. ou en *italique*. Les noms en **gras** représentent les différents paquetages que nous avons séparés.

Ceux en *italiques* sont des classes abstraites créés 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 séparé les diagrammes des éléments graphiques. En effet, un diagramme sera composé de toute sorte d'éléments graphiques. Puis nous avons découpé ces derniers en deux, isolant ainsi les lignes des éléments de modélisation tels que les classes, les traitements ou les cas d'utilisation.

ElementGraphique et **ElementModelisation** sont des classes abstraites qui regroupent les fonctions communes à toutes les classes 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 voir comment ils fonctionnent plus en détails.

2.1.1 Paquetage `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 respectivement au graphe dans lequel sont stockés les éléments et le diagramme affiché 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 descendantes servant à créer la représentation graphique de l'objet et de l'afficher sur le diagramme.

Paquetage `eltModelisation` Ce paquetage regroupe toutes les classes représentant les différents éléments de modélisation que l'on trouve dans les diagrammes UML de cas d'utilisation, classes et de séquence. *Acteur* est une classe abstraite car les acteurs actifs 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 nouvelles fonctionnalités comme par exemple de la génération de code Java.

Paquetage `ligne` Ce paquetage regroupe peu de classe. *TypeLien* est une classe énumérée servant à recenser tous 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 une saisie facilitée 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 appliquer au nouveau lien le style choisi par l'utilisateur. *MessageTraitement* spécialise *Lien*, permettant de créer un style de lien particulier aux messages entre traitement dans les diagrammes de séquences.

2.1.2 Paquetage `diagramme`

Le paquetage `diagramme` comprend plusieurs types de diagramme prédéfinis qui sont cas d'utilisation, classes et séquence. Ils descendent de la classe *Diagramme*. Chaque type de diagramme implémente deux méthodes `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 réimplémenter 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 différents aux adresses suivantes

- ▷ <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 paquetages est disponible à l'annexe B page 13. Celui-ci vous permettra de repérer les relations entre les classes.

Annexe A

Glossaire

Bibliothèque (p 6) – Un composant indépendant fournissant des méthodes déjà implémentées, facilitant le développement d’une application plus importante.

Démonstrateur (p 6) – Un démonstrateur est un logiciel simple, permettant de montrer les possibilités d’une bibliothèque

Diagramme de cas d’utilisation (p 5) – Représentation graphique permettant de décrire les interactions entre les acteurs et le système.

Diagramme de classes (p 5) – 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 5) – 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 5) – Fonctionnalité du logiciel, ayant un cycle de développement 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 11) – 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 5) – Méthode de développement d’un projet dans laquelle le projet final est divisé en plusieurs fonctionnalités appelées incréments qui sont développés et ajoutées au projet au fur et à mesure.

Paquetage (p 10) – Un paquetage en Java est un regroupement de classes ayant la même thématique.

UML (p 5) – (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 paquetages

