Université de Montréal

IFT3051 : projet défini et encadré par un professeur associé à un laboratoire de recherche universitaire.

Sujet : Détection automatique des patrons de conception.

Rapport d'étude.

Réalisé par :
- Harmak Mohammed Amine
- El Badaoui Laila

Responsable :
Yann-Gaël Guéhéneuc, Ph.D.
Professeur adjoint
Laboratoire de génie logiciel
Département d'informatique et de recherche opérationnelle.

INTRODUCTION : ..... **- 2 -**

OBJECTIF DU PROJET : ..... **- 3 -**

LES PREMIÈRES ÉTAPES : ..... **- 4 -**

L'EXPÉRIENCE ECLIPSE: ..... **- 10 -**

CVS (CONCURRENT VERSIONS SYSTEM): ..... **- 10 -**
ENVIRONNEMENT DE DÉVELOPPEMENT : ..... **- 11 -**
ECLIPSEUML : ..... **- 12 -**
AVANTAGES ET FONCTIONNALITÉS : ..... **- 13 -**

ANALYSE DES PROGRAMMES ET RÉSULTATS TROUVÉS : ..... **- 14 -**

ANALYSE DE JUZZLE V0.5 : ..... **- 14 -**
ANALYSE DE HOLUBSQL V1.0 : ..... **- 15 -**
ANALYSE DE JTANS V1.0 : ..... **- 16 -**
ANALYSE DE RISK V1.0.7.5 : ..... **- 17 -**
ANALYSE DE JSETTLERS V1.0.5 : ..... **- 18 -**
ANALYSE DE GANTT PROJECT V1.10.2 : ..... **- 19 -**

ÉVOLUTION DE LA STRATÉGIE DE RECHERCHE : ..... **- 21 -**

UNE MÉTHODE PROGRESSIVE : ..... **- 21 -**
DIVISER LE TRAVAIL EN TROIS ÉTAPES : ..... **- 22 -**
PREMIÈRE ÉTAPE : LES PATRONS CRÉATIONNELS ..... - 22 -
Les singletons : découverte dans le plaisir ..... - 25 -
Factory méthode : contrôle de la création de classes ..... - 25 -
Prototype : Des clones de classes ..... - 26 -
DEUXIÈME ÉTAPE : LES PATRONS STRUCTURELS ..... - 27 -
Composite : utilisation des hiérarchies ..... - 27 -
TROISIÈME ÉTAPE : LES PATRONS COMPORTEMENTAUX ..... - 27 -
Iterator : modèle classique ..... - 27 -

CONCLUSION : ..... **- 28 -**

Introduction :

La programmation orientée objet : POO a fait son apparition dans les années 1960. Après, elle a été utilisée dans les travaux d'intelligence artificielle dans les années 1970-1980.

Le but ultime de la POO contrairement aux programmations procédurale et fonctionnelle est de permettre aux programmeurs de comprendre les programmes faits par d'autres, ajouter des fonctions sans tout réécrire, faciliter la maintenance des programmes et le plus important est de réutiliser des programmes ou des méthodes déjà testées sans être obligés à réécrire tout le code.

C'est de la où nait le concept des patrons de conception, un patron comme son nom l'indique est une structure qu'on réutilise à chaque fois qu'on en a besoin.
Dans la programmation orientée objet, l'utilisation des patrons de conception permet de résoudre les problèmes récurrents de celle-ci.

> « Chaque patron décrit un problème qui se manifeste constamment dans notre environnement, et donc décrit le cœur de la solution à ce problème, d'une façon telle que l'on puisse réutiliser cette solution des millions de fois, sans jamais le faire deux fois de la même manière »
>
> Christopher Alexander -1977.

Parmi les champs d'utilisation des patrons de conception, il y a la qualité des programmes qui permet de mieux développer et donc de ne pas faire beaucoup de maintenance ce qui permet de gagner en temps et en argent. Ceci dit, l'utilisation abusive des ces patrons aura l'effet contraire sur la qualité.

Objectif du projet :

Le projet consiste à se familiariser avec les patrons de conception ainsi que la plateforme de travail Eclipse, trouver une méthode de recherche efficace qui permettra de cibler la recherche des patrons, ainsi que la possibilité d'utiliser un outil de rétro conception qui permet de revenir en arrière c'est-à-dire, à partir du code source, le logiciel de rétro conception permet de recréer les digrammes de classes et ainsi le travail de recherche est plus facile à faire.

On aura à faire l'analyse de six logiciels de grandeurs différentes et de types différents pour trouver les patrons de conception utilisés dans ces logiciels.

Voici un tableau récapitulatif des ces logiciels :

Nom du programme	Description	Nombre de packages	Nombre de classes
Gantt Project v1.10.2	Un logiciel de gestion de projet	27	297
JSettlers v1.0.5	Une version web du jeu les settlers de Catan écrit en java	10	148
Risk v1.0.7.5	Une plateforme classique de RISK (Jeu de guerre)	7	37
JTans v1.0	Un ancien jeu de casse tête chinois	5	36
HolubSQL v1.0	Un interpreteur SQL embarqué	6	30
Juzzle v0.5	Un projet libre dont le but est de fournir un framework et une API permettant de construire des outils de simulation intégrée pour n'importe quel sujet technologique	2	12

Fig.1 : Complexité des programmes

Les premières étapes :

Pour commencer le travail sur le projet, la première chose à faire était de se familiariser avec les patrons de conception, le livre de Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides intitulé : Design Patterns {Elements of Reusable Object-Oriented Software (Addison-Wesley, 1st edition, 1994. isbn: 0-201-63361-2) qui a été d'une grande utilité puisqu'il était la principale et la plus complète source d'information, on peut ajouter à cela des recherches sur internet pour clarifier quelques notions sinon, Internet a été plus utilisé pour faire des recherches concernant Eclipse, Java et pour l'utilisation du logiciel de retro-conception.

Le livre nous a permis de mieux comprendre les patrons de conception, leurs structures et leurs utilisations, au début, le nombre de références au livre était plus important mais au fur et à mesure que le travail avançait, c'est devenu de moins en moins fréquent de s'y référer puisqu' avec le temps, la recherche devient de plus en plus facile.

Il existe plusieurs patrons de conceptions (Gof et GRASP), les plus répandus sont ceux du Gof (Gang of four) qui sont les concepteurs du livre cité en référence, mais l'origine des patrons est surtout tirée des travaux de Christopher Alexander dans les années 1970.

La description d'un patron de conception suit un certain formalisme :

- Nom
- Description du problème à résoudre
- Description de la solution : les éléments de la solution, avec leurs relations.
- Conséquences : résultats issus de la solution.

Voici un résumé des patrons vus dans le livre avec une brève définition :

1. Patrons créationnels :

- **Abstarct factory** : permet à une interface de créer des objets sans leurs classes concrètes.
  Fréquence d'utilisation **:** 

  très forte

- **Builder** : sépare la construction d'un objet complexe de ses représentations comme ça, le même processus de création permet de créer différentes représentations.
  Fréquence d'utilisation **:** 

  moyen bas

- **Factory method** : Définie une interface qui crée un objet mais permet aux classes qui héritent de cette interface de choisir laquelle instancier.
  Fréquence d'utilisation **:** 

  très forte

- **Prototype** : spécifie le type d'objet à créer en utilisant une instance prototypique et créer de nouveaux objets en copiant ce prototype.
  Fréquence d'utilisation **:** 

  moyen

- **Singleton** : assure qu'une classe a une seule instance et permet un accès global à cette classe.
  Fréquence d'utilisation **:** 

  moyen fort

2. Patrons Structurels :

- **Adapter** : Convertit une interface d'une classe à une autre interface que le client attend.
  Fréquence d'utilisation **:** 

  moyen fort

- **Bridge** : découple une abstraction de son implémentation de manière que les deux peuvent varier indépendamment.
  Fréquence d'utilisation **:** 

  moyen

- **Composite** : compos des objets a des arbres de structure pour représenter partie entier hiérarchies, ils permes aux clients de traiter les objets individuels et les compositions des objets d'une manière uniforme.
  Fréquence d'utilisation **:** 

  moyen fort

- **Decorator** : Attache dynamiquement des responsabilités additionnelles à un objet.
  Fréquence d'utilisation **:** 

  moyen

- **Facade** : donne une interface unifiée à un ensemble d'interfaces dans un sous-système, définie une interface d'un niveau élevé qui fait que le sous-système devient facile d'utilisation.
  Fréquence d'utilisation **:** 

  très forte

- **Flyweight** : utilise le partage pour supporter un grand nombre d'objet 'fine-grained' efficacement.
  Fréquence d'utilisation **:** 

  bas

- **Proxy** : Fournit un remplaçant ou un paramètre fictif à un autre objet pour contrôler son accès
  Fréquence d'utilisation **:** 

  moyen fort

3. Patrons comportementaux :

- **Chain of responsability** : évite de coupler l'expéditeur d'une requête avec son receveur en donnant la chance à plus d'un objet de traiter cette requête. Enchaîne les objets receveurs et passe la requête à travers la chaîne jusqu'à ce qu'un objet la traite.
  Fréquence d'utilisation **:** 

  moyen bas

- **Command** : encapsule une requête en un objet, comme ça il permet de paramétrer les clients avec différentes requêtes, queues ou log requêtes et supporter les opérations inaccessibles ou infaisables.
  Fréquence d'utilisation **:** 

  moyen fort

- **Interpreter** : définit pour un langage donné une représentation pour sa grammaire avec un interpréteur qui utilise cette représentation pour interpréter des phrases du langage.
  Fréquence d'utilisation **:** 

  bas

- **Iterator** : fournit une manière pour accéder séquentiellement aux éléments d'un objet agrégé sans exposer sa représentation intérieure.
  Fréquence d'utilisation **:** 

  très forte

- **Mediator** : définit un objet qui encapsule l'interaction d'un ensemble d'objets entre eux. Il permet la perte de couplage en empêchant les objets de s'interférer les uns les autres explicitement, et il permet à l'utilisateur de varier leurs interactions indépendamment.
  Fréquence d'utilisation **:** 

  moyen bas

- **Memento** : sans violer l'encapsulation, capture et extériorise un état interne d'un objet de telle façon que cet objet peut être restauré à cet état plus tard.
- Fréquence d'utilisation **:** 

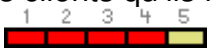
  bas

- **Observer** : définie une dépendance « un à plusieurs » entre les objets de façon à ce que si un objet change d'état, tous les objets dépendants de lui sont mis à jour automatiquement.
  Fréquence d'utilisation **:** 

  très forte

- **State** : permet à un objet de changer son comportement quand son état interne change.
  Fréquence d'utilisation **:** 

  moyen

- **Strategy** : définie une famille d'algorithmes, encapsule chacun d'eux, et les rendre interchangeables. Il laisse l'algorithme varier indépendamment des clients qu'ils l'utilisent.
  Fréquence d'utilisation **:** 

  moyen fort

- **Template method** : définie le squelette d'un algorithme dans une opération, différant quelques étapes aux classes qui héritent. Il permet à ces classes de redéfinir certaines étapes d'un algorithme sans changer la structure de ce dernier.
  Fréquence d'utilisation **:** 

  moyen

- **Visitor** : représente une opération qui sera exécutée dans un élément d'une structure d'objet. Il permet de définir une nouvelle opération sans changer les classes des éléments sur lesquels elles opèrent.
  Fréquence d'utilisation **:** 

  bas

(Source : *http://www.dofactory.com/Patterns.aspx#list*)

Notre travail consistait à trouver la façon la moins contraignante et la plus intéressante pour nous plonger dans le code source. Certaines difficultés sont apparues au niveau de la manipulation de l'outil de travail Éclipse. Il nous a fallu faire une recherche approfondie sur le net pour pouvoir afficher l'arbre de dérivation ainsi que la hiérarchie des classes composant les différents package des programmes.

En s'inspirant de ces hiérarchies des classes, nous avons essayé de déceler des schémas de patrons de conception simples tels que singleton ou prototype, sans grand succès il faut l'avouer.

C'est ainsi que nous nous sommes fixé comme premier objectif de bien comprendre l'essence même des patrons de conception. À mesure que l'on avançait, on découvrait de plus en plus de subtilités dans l'utilisation des patrons de conceptions. Le fait de comprendre pourquoi ce savoir a été rassemblé faisait que notre enthousiasme devenait plus grand.

On a aussi utilisé d'autres méthodes de recherche trouvées par une équipe qui a déjà travaillé sur ce type de projet.

Parmi ces recherches on retrouve **la recherche par héritage** qui consiste à trouver les classes abstraites et les interfaces en premier lieu et ensuite essayer de tourner autour de ces classes pour identifier un patron.

- **La recherche par vue globale** qui consiste à utiliser les outils de rétro conception pour trouver le diagramme de classe et ensuite essayer de trouver une structure similaire à celle d'un patron de conception.

- **La recherche naïve** consiste à rechercher parmi les classes du programme s'il y en a quelques unes qui portent un nom significatif d'un patron de conception et essayer de retrouver toute la structure du patron.

- **La recherche par relation des patrons** consiste à suive le schéma des relations des patrons de conception tiré du livre, cette méthode nécessite déjà l'existence d'un patron de conception pour pouvoir rechercher ceux qui lui sont liés.

De ce fait, on a utilisé **la recherche opportuniste** qui réunie tous ces types de recherches surtout qu'elle réunie les avantages de celles-ci et non pas les inconvenants.

L'expérience Eclipse:

Fig.2 : Logo Eclipse

Avant de commencer le projet, aucun de nous ne connaissais l'outil de développement Eclipse. Il a fallu vite se réajuster pour maîtriser les différentes fonctionnalités offertes.

CVS (Concurrent Versions System):

Initialement développé pour Unix, CVS (Concurrent Versions System) est un outil libre de gestion de versions. Eclipse propose une perspective pour utiliser CVS dans un projet.

La perspective «Exploration du référentiel CVS» permet de gérer les échanges et le contenu des projets stockés sous CVS.

Tous les codes sources ainsi que les différentes versions des logiciels utilisés dans notre cas se trouvent sur le compte CVS de l'équipe **ptitdej** au DIRO. L'accès au référentiel est toutefois restreint par un mot de passe.

L'installation de **CVSnt** sur nos machines est un ajout de valeur dans notre base de connaissances. Un tutorial très simple est disponible à l'adresse suivante :

« http://perso.wanadoo.fr/jm.doudoux/java/dejae/chap012.htm. »

Environnement de développement :

Parmi les fonctionnalités d'Eclipse qui ont contribuées à l'avancement de ce travail figure l'outil de visionnement de classes (Class viewer) ainsi que l'outil de recherche de patterns très avancé d'Eclipse.

Le **class viewer** permet d'avoir une hiérarchie des packages ainsi que la liste de toutes les classes, interfaces et classes abstraites du projet.

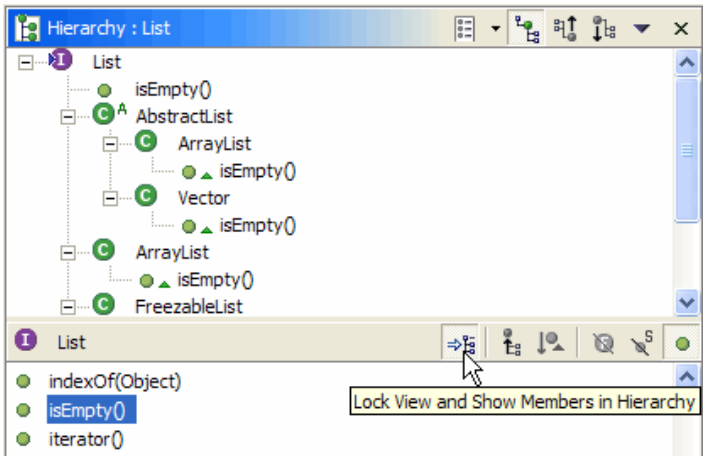


Fig.3 Tips : Pour obtenir la hiérarchie d'un projet ou d'un package Click droit sur le projet >> view hierarchy.

L'outil de recherche d'Eclipse a été également très sollicité. Sa simplicité et ses fonctionnalités avancées ont beaucoup facilité sa prise en main.

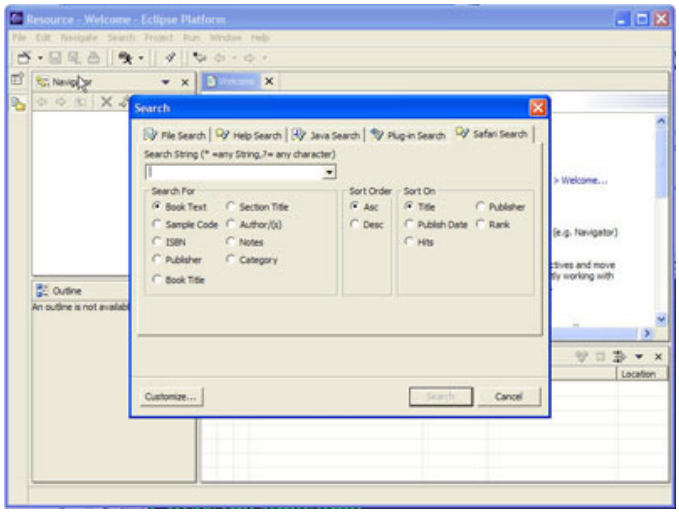


Fig.4 Tips : raccourci pour fenêtre de recherche (ctrl + H)

EclipseUML :

Fig.5: Logo Omondo

Omondo est l'outil de reverse engineering le plus puissant au monde. C'est un plugin qui peut être greffé à Eclipse.

Installation :

- Télécharger la version Eclipse SDK 3.1.2 (ou supérieur), Windows (103 MB)
- Extraire les fichiers dans un répertoire.
- Lancer Eclipse puis mettre à jour les plugins.
- Télécharger le fichier « EclipseUML 2.1.0.*.jar » sur www.omondo.com (Attention la version studio est en période d'essai de 15 jours)
- Lancer le fichier .jar a l'aide de la commande « java –jar EclipseUML.*.jar »
- Après le lancement de l'applet, choisir le chemin de destination d'Eclipse.
- Pour utiliser Omondo, choisir un projet, puis click droit.
- Ensuite choisir l'onglet UML puis l'option Reverse engeneering.

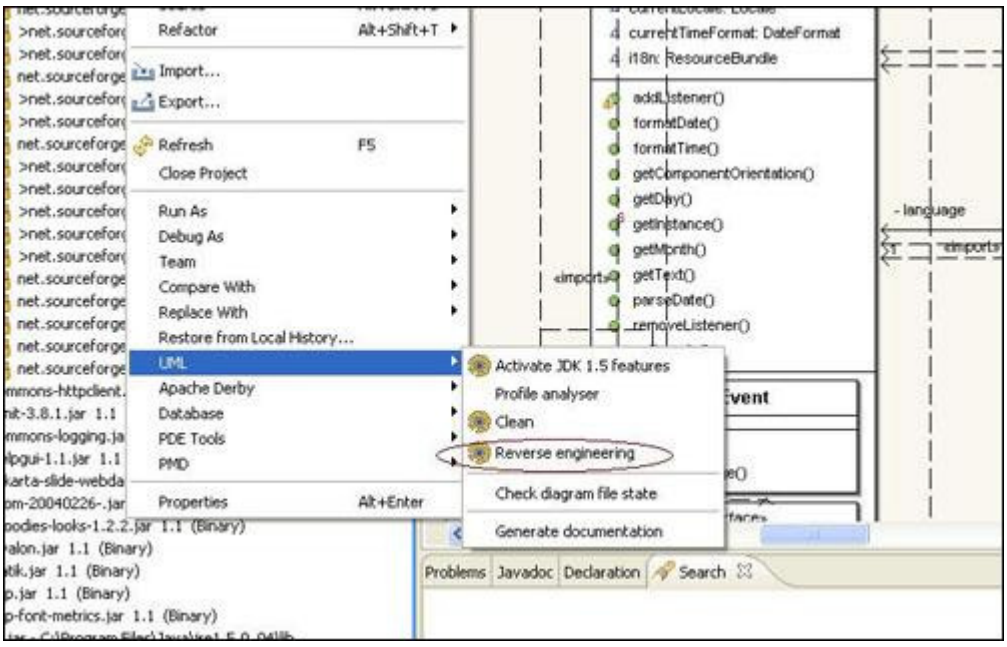


Fig.6 - Tips : Une fois la fenêtre de retro conception apparu, choisir « dependencies » et « inheritence » pour obtenir les relations de dépendances et d'héritage.

Avantages et fonctionnalités :

	EclipseUML Free Edition 1.2.1 Traditional UML Solutions	EclipseUML Enterprise Edition
Package dependency reverse		
Classe diagram reverse		
Attribute reverse		
Association reverse		
Inheritance reverse		
Dependency reverse		
Jar file reverse		
Sequence diagram reverse	-nd	
Project reverse	-nd	
Getter/setter recognition	-nd	
Cardinality recognition	-nd	
Element type in a collection	-nd	
Inverse association resolution	-nd	
Qualified association	-nd	
UML model reverse engineering	-nd	

Fig.7 - Source : http://www.tutorial-omondo.com/reverse/index.html

La rétro conception est un outil qui permet d'obtenir une vue graphique de l'ensemble du projet. Cette vue des classes sert à détecter rapidement des structures des patrons de conception complexes surtout si elles impliquent plusieurs relations d'héritages et d'associations.

Analyse des programmes et résultats trouvés :

Pour se familiariser avec la recherche, le choix a été de commencer avec le programme de petite taille (la taille du programme est calculée suivant le nombre de classes qu'il contient), la raison est que l'analyse devrait être plus facile avec le logiciel de retro conception.

Donc, chaque programme et suivant sa constitution a nécessité une analyse différente et ce en variant les méthodes de recherche et si le logiciel de rétro conception a fourni ou non des informations sur les diagrammes de classes.

Analyse de Juzzle v0.5 :

Fig.8: Logo Juzzle

Juzzle v0.5 est un projet libre dont le but est de fournir un Framework et une API permettant de construire des outils de simulation intégrée pour n'importe quel sujet technologique, il contient deux packages et douze classes et une seule classe Interface.

Puisque presque la majorité des structures des patrons de conception reposent sur l'utilisation des classes abstraites et les interfaces et sur la notion d'héritage des classes, les chances de trouver un patron de conception semblaient très minces.

Donc, la seule classe qui pourrait nous indiquer l'existence d'un patron de conception et les classes qui héritaient de celle-ci n'avaient pas vraiment la structure d'un patron de conception et après plusieurs tentatives d'utilisation des autres méthodes de recherche, la conclusion était claire, ce programme ne contenais aucun patron de conception.

Analyse de HolubSQL v1.0 :

Fig.9: Logo HolubSQL

HolubSQL v1.0 est un interpréteur SQL embarqué, c'est-à-dire que HolubSQL v1.0 est un ensemble de classes qu'on peut incorporer à un programme (par exemple Java), il n y a pas de serveur a part et toutes les tables de la base de données sont sauvegardées en mémoire, il contient 6 packages et 30 classes.

Les patrons de conception trouvés sont en nombre de quatre (quatre Adapter, un Composite, un Iterator et un Flyweight).

Les patrons de conception Adapter ont tous étés trouvés dans un package appelé Adapter, donc le plus difficile était de trouver les autres classes qui interagissaient avec chaque patron et de vérifier la structure de ce dernier.

Pour le patron de conception Composite, la démarche était de prendre la classe abstraite, de chercher toutes les classes qui héritent de celle-ci et en cherchant dans les structures des patrons qui ressemblaient à cette structure, la suite était d'identifier chaque composante de la structure du patron Composite.

Le patron Flyweight a été identifié grâce à l'existence du Composite puisqu'il y a une relation entre ces deux patrons (recherche par relation des patrons), il fallait après spécifier les composantes de la structure du patron, les noms des classes a aidé dans l'identification du patron.

La recherche naïve a permit de trouver le patron de conception Iterator puisqu'il y avait une classe qui portait un nom significatif qui permettait de penser à ce patron et la suite comme pour tous les patrons était de trouver les autres composantes de la structure du patron.

Analyse de JTans v1.0 :

Fig.10: Logo Jtans

JTans v1.0 est une version java d'un ancien jeu de casse-tête chinois, l'objectif du jeu est de construire un dessin à partir de 7 formes géométriques (5 tringles, un rectangle et un parallélogramme), il est constitué de 5 packages et 36 classes.

Les patrons de conception trouvés sont un Observer et un Composite. Le patron Composite a été trouvé grâce l'utilisation du logiciel de retro conception qui permettait de voir les digrammes de classes et ainsi voir l'héritage de ces classes qui poussent à considérer l'existence d'un patron de conception et après essayer de trouver la quelle des structures des patrons peut concorder avec celle trouvée.

Le patron de conception Observer a été trouvé de la même manière.

Analyse de Risk v1.0.7.5 :

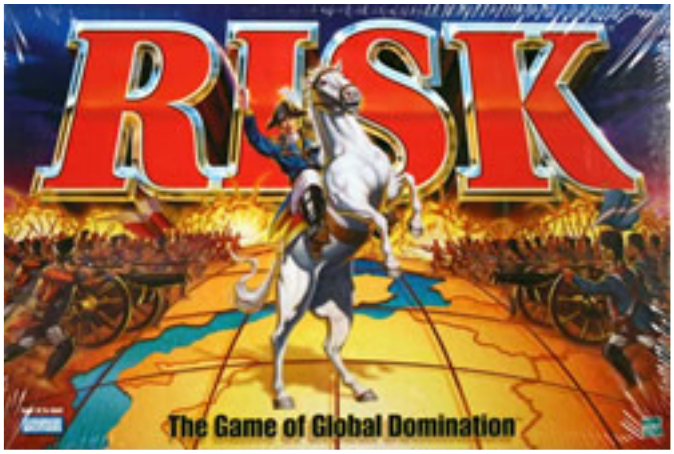


Fig.10: Risk Game

Risk v1.0.7.5 est un jeu de stratégie dont le but est de conquérir des territoires (il y a 42 territoires à gagner sur un planisphère), il contient 7 packages et 37 classes.

Parmi ces classes, il y avait une qui portait un nom significatif au patron de conception Adapter et après recherche de la structure du patron de conception, il a été facile de concorder cette structure à celle trouvée dans le programme. Adapter était le seul patron de conception trouvé dans ce programme.

Analyse de JSettlers v1.0.5 :

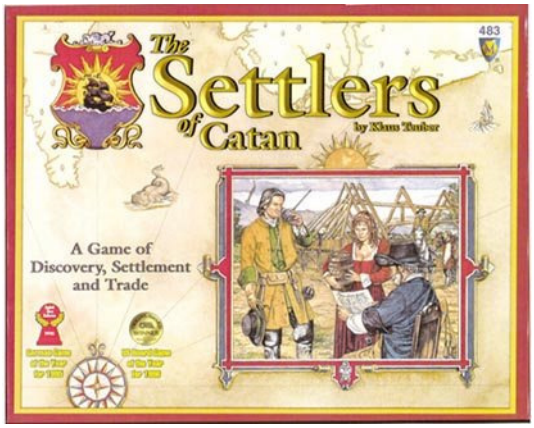


Fig.11 : Settlers game

JSettlers v1.0.5 ou le jeu des colons de Catane qui est une version web du jeu, ce jeu consiste à créer une colonie et essayer de la faire prospérer pour gagner, il est constitué de 10 packages et de 148 classes.

Dans ce programme, deux patrons de conception ont été identifiés, un patron Observer et un patron Builder.

Le patron de conception Observer a été trouvé en cherchant dans les classes pour trouver les interfaces et les classes abstraites, dans ce cas, les interfaces utilisées sont celles de java.awt et ensuite il fallait trouver les autres composantes de la structure du patron de conception en identifiant toutes les classes qui rentrent dans cette structure.

Pour le patron Builder, c'est en suivant la recherche par héritage d'une classe abstraite, ca a aboutit à trouver une structure qui ressemble à la structure du patron de conception Builder.

Analyse de Gantt Project v1.10.2 :

Fig.12 : Logo Gantt project

Tant que le projet ne dépasse pas 100 classes, le travail avec la rétro conception reste très efficace et permet de gagner du temps en apportant un plus à la recherche.

Mais dès que le projet devient un peu plus volumineux, la performance chute dramatiquement et l'on remarque que la méthode de rétro conception ne marche plus. L'exemple le plus flagrant fut très certainement le cas du logiciel Gantt Project v1.10.2. Ce dernier dispose en effet de plus de 27 packages pour un total de prés de 300 classes environ.

Le test de retro conception a donné dans un premier temps le résultat suivant :

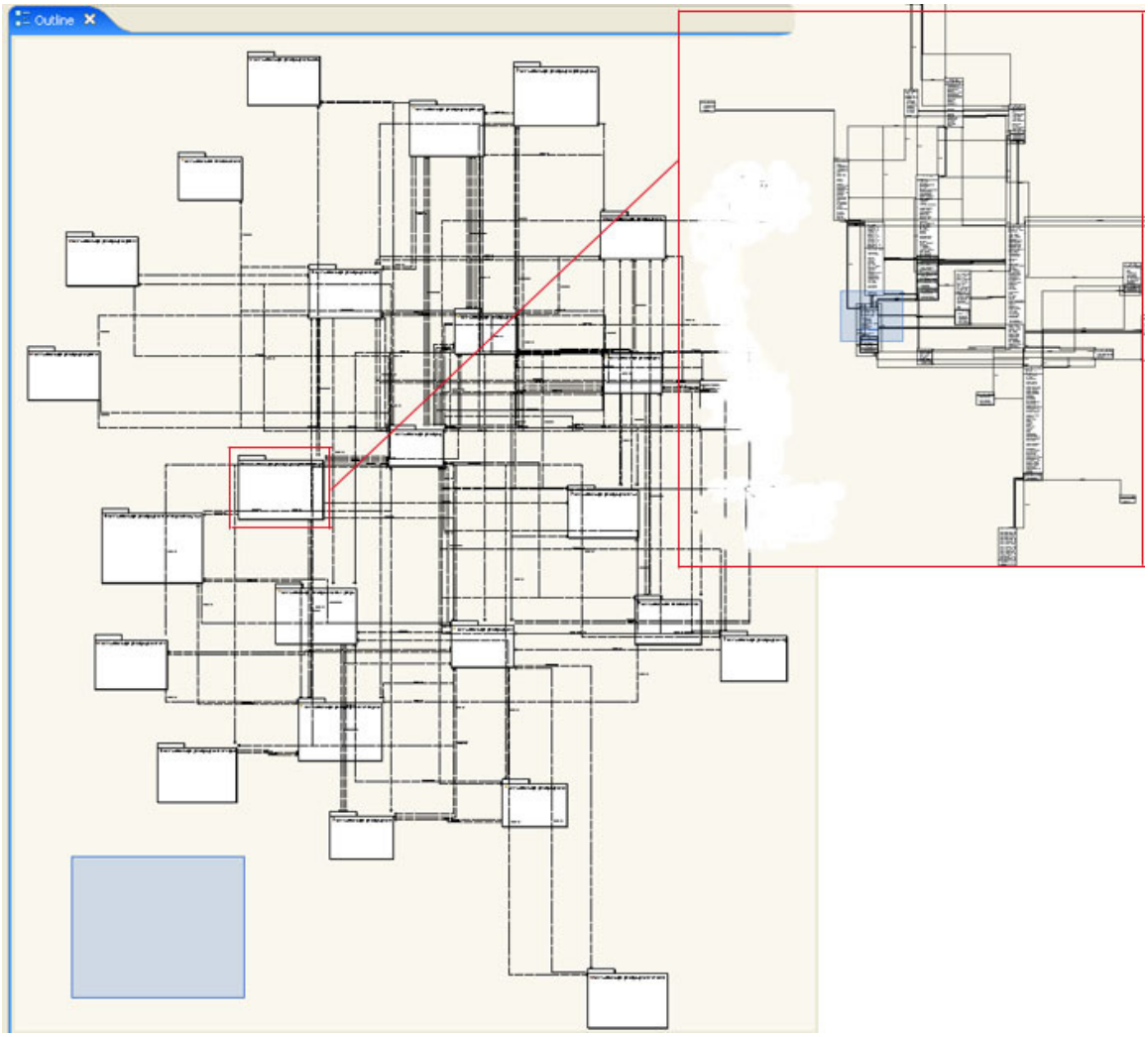


Fig.14 : UMLview du projet Gantt donné par Omondo

Il apparait évident que suivre les relations de dépendances à travers cet outil pour ce programme en particulier serais une tache ardue.

C'est pour cela qu'en cours de route la stratégie de recherche s'est modifié pour se restreindre à une navigation plus sure et moins contraignante à travers le class viewer.

Évolution de la stratégie de recherche :

Gantt Project v1.10.2.

Devant la taille imposante de ce logiciel de management, nous avons du adapter notre méthode de recherche global vers une nouvelle méthode plus progressive.

En se fiant à la logique des patrons de conception, nous avons divisé notre stratégie en trois étapes qui correspondent en fait aux trois familles de la classification du GoF.

Une méthode progressive :

Pour ce qui est des cinq autres programmes, notre démarche se voulait plus générale.

On ne sentait pas réellement la nécessité de comprendre le fonctionnement et l'utilité du programme en tant que tel, mais on se basait plutôt sur la présence ou non de certains patterns dans le code en s'aidant de la représentation graphique des classes obtenu par rétro conception.

De plus, aucun ordre ne prévalait réellement durant la recherche car on pouvait parcourir le code à la recherche d'une piste pour un patron quelconque.

Vu que le nombre de classes abstraites et d'interfaces était limité, il était peu probable que l'on puisse passer au travers d'un patron sans le détecter puisqu'on disposait d'une vue d'ensemble claire de tout le logiciel.

Là où le problème devenait plus corsé, c'est lorsque cette vision globale des dépendances entre classe devenais plus complexes et plus difficile pour un esprit humain à dresser. Le fait de relever les relations d'héritages devenait plus un jeu de méli mélo impossible qu'il fallait résoudre.

Devant l'impasse, nous nous sommes orientés vers une autre démarche, celle de se mettre dans l'esprit des développeurs et de recréer les conditions de développement initiales.

Il était impératif pour ce genre de programme de bien comprendre l'architecture et l'organisation des classes pour comprendre les problèmes auxquels les développeurs ont été confrontés durant la phase de développement.

Il est clair que le logiciel est le fruit de la collaboration de différents programmeurs et de différentes équipes.

Le fait de recréer les conditions de développement initiales à deux personnes… **c'est du sport !!**

Mais le résultat vaux bien le risque à prendre.

Puisque comprendre les problèmes de développement donne un sérieux indice sur l'endroit où les développeurs ont utilisé des patrons de conception pour les résoudre.

Diviser le travail en trois étapes :

Première étape : Les patrons créationnels

La première étape consistait en fait à comprendre la manière dont le programme se lance.

Quels sont les parties chargées durant le lancement ?

Quels sont les classes qui interviennent durant l'affichage et durant l'initialisation du programme?

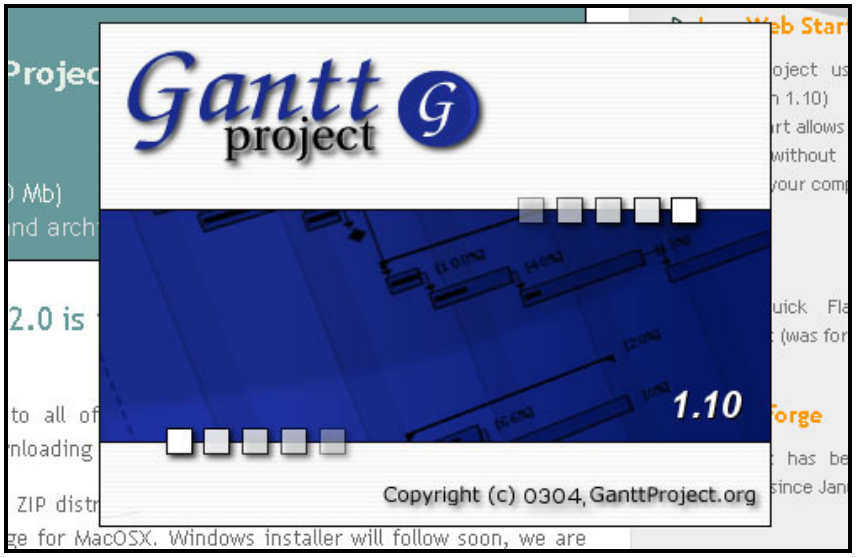


Fig.15 – Lancement du programme : chargement des options et des ressources

Toute la partie de chargement des éléments graphiques se trouve dans le package : Net.sourceforge.ganttproject

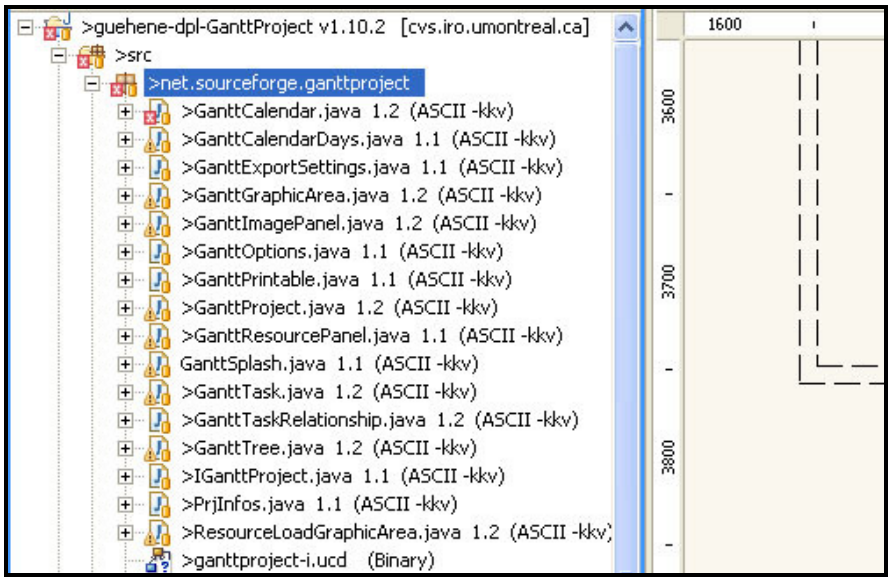


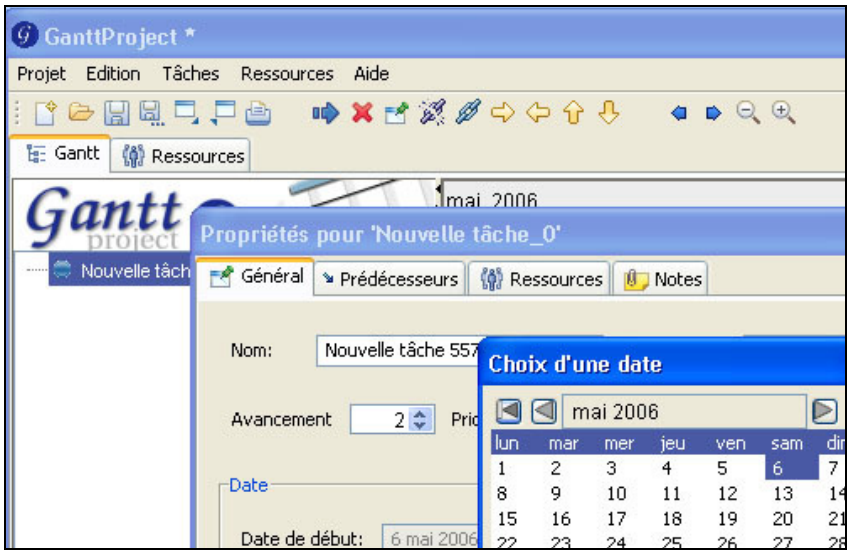
Fig.16 – Ensemble des classes responsables de l'initialisation regroupé dans un package

Le but est de se familiariser avec le fonctionnement des classes.

Cette étape permet de localiser les patrons de conceptions de type créationnels.

Ce type de patrons définit la manière de faire l'instanciation et la configuration des classes et des objets.

Une fois le chargement du programme fini, plusieurs options sont disponibles dans le menu. Il s'agit en fait de faire la correspondance entre les options du programme et les packages ou à l'inverse définir pour chaque package son rôle dans le programme.



Corrélation entre les fenêtres et les packages.
Choix d'une date → net.sourceforge.ganttproject.time.gregorian …
Propriétés pour '' → net.sourceforge.ganttproject.gui.taskproperties …*
GanttProject → net.sourceforge.ganttproject …

En focalisant sur les constructeurs des objets et sur les méthodes appelées à l'intérieur des constructeurs, on arrive à détecter certains motifs de création.

Les singletons : découverte dans le plaisir

Les singletons ont été les premiers patrons détectés durant la première phase.

Le fait de trouver des singletons était généralement le signe que la récolte allait être fructueuse !

La détection des singletons est plutôt simple. C'est un bon exercice pour les débutants d'autant plus que certaines variantes de la structure prédéfinie du GoF présente un défi supplémentaire.

Plusieurs formes du singleton ont été trouvées, certaines possèdent même le choix de deux constructeurs, l'un d'eux étant privé.

Mais tous présentent comme point commun la capacité d'instancier une seul et unique copie de la classe et de la retourner par la méthode GetInstance ().

Factory méthode : contrôle de la création de classes

Trois patrons de conceptions ont été détectés dans le projet.

Deux d'entre eux sont en relation avec les librairies java et le troisième sert à contrôler l'instanciation d'une classe Facade.

En fait l'interface Factory contrôle la création de plusieurs Facades.
C'est la combinaison d'un factory method et du patron Facade.

Prototype : Des clones de classes

Pour détecter ce patron, il est nécessaire de s'attarder sur les classes qui font appel à une méthode clone (). Cette méthode copie les informations de l'instance et retourne un clone de l'objet.

Une fois ce schéma trouvé, on vérifie si dans l'intention du développeur il était question de copier ces objets à partir d'une instance dans le but de rendre le système indépendant de la façon de créer l'objet prototype.

En effet, dans la classe GanttCalendar et GanttTask, on retrouve cette structure spécifique. En fait on comprend le désir des développeurs de vouloir créer une hiérarchie de taches à accomplir dans un projet donné et d'assigner pour ces taches des calendriers.

La création d'une nouvelle tache fait appel à la méthode clone car justement l'application ne se soucie pas de connaitre les détails de création d'une tache. Il en est de même pour les calendriers. L'utilisation du patron prototype permet d'avoir une hiérarchie moins redondante.

Deuxième étape : Les patrons structurels

La deuxième étape débute lorsqu'on commence à comprendre à peu prés a quoi sert tous les packages du projet.

Dés lors il devient possible de suivre la logique derrière la structure des classes et des relations d'héritage.

Comprendre le squelette des packages permet de détecter plus facilement le motif de conception de type comportementaux.

Composite : utilisation des hiérarchies

La structure qu'on a détectée est celle sous forme d'arbre, faisant intervenir des feuilles et des composants. Le patron de conception composite a été clairement défini puisqu'il faisait appel à des librairies Java structurées.

Troisième étape : Les patrons comportementaux

La troisième étape se base sur les comportements des classes et des objets pendant leur collaboration.

Iterator : modèle classique

Il fallait dans ce cas détecter les grandes organisations d'objets regroupés sous forme de structure de données. Souvent en collaboration avec ces structures se trouvent des classes iterateur (quelque fois prédéfini dans Java)

Conclusion :

Au début, le plus difficile était de se familiariser avec les patrons de conception et les outils de développement.

Une fois les notions de bases assimilées, le travail est devenu plus structuré et des méthodes de recherches ont été élaborées en fonction des programmes.

Ce travail permettra d'enrichir la base de données des logiciels figurant dans le XML et constituera une plateforme de comparaison avec les résultats fourni par Ptidej.

Le projet constitue une initiation au travail en groupe. L'expérience acquise nous a permis de nous familiariser avec les contraintes et les délais que présenterais un véritable projet dans un cadre plus professionnel.