

IFT3051 - Projet d'informatique

Détection manuelle de micro-architectures  
semblables aux patrons de conception du Gang of Four

présenté à

Yann-Gaël Guéhéneuc  
et  
Stefan Monnier

par

Nicola Grenon  
Abdeljabar Hammodan  
Rafik Ouanouki

Laboratoire de Génie Logiciel  
Département d'informatique et de recherche opérationnelle  
Université de Montréal

Jeudi 14 septembre 2006

## Table des matières

Mandat .....	page 3
Introduction .....	page 4
Organisation du temps .....	page 6
Méthodologies .....	page 10
Les logiciels .....	page 13
Résultats .....	page 16
Conclusion .....	page 19
Appréciations personnelles .....	page 21
Bibliographie .....	page 23
Annexe ( <i>Petit guide pratique</i> ) .....	page 24

## Mandat

Le mandat de notre équipe de travail était d'extraire, à partir de code donné de projets déjà existants, les micro-architectures pouvant être associées aux patrons de conception (Design Patterns) tels que définis par le Gang of Four dans leur livre de référence<sup>1</sup>. Ceci afin de poursuivre la récolte de données de référence pour générer une base de donnée en format XML de patrons de conception. Cette dernière servira ensuite de modèle de travail pour la poursuite du projet Ptidej<sup>2</sup> par nos collègues du GELO.

Les logiciels dont nous devons analyser le code devaient être parmi les suivants:

GANTT PROJECT v1.10.2	616	Tasks management software ganttproject.sourceforge.net
HOLUBSQL v1.0	151	Embedded SQL interpreter www.holub.com/software/holubSQL/
JHOTDRAW v5.1	266	Graph drawing framezork www.jhotdraw.org
JSETTLERS v1.0.5	255	Settlers of Catan board game jsettlers.sourceforge.net
JTANS v1.0	206	Tangram puzzle game jtans.sourceforge.net
JUNIT v3.7	289	Unit testing framework www.junit.org
JUZZLE v0.5	99	Simple puzzle game juzzle.sourceforge.net
LEXI v0.0.1α	216	Text editor lexi.sourceforge.net
RISK v1.0.7.5	256	Strategy game javarisk.sourceforge.net

Les bénéfices escomptés pour les participants consistaient en une familiarisation avec les patrons de conception et leur emploi. De l'acquisition d'expérience avec l'environnement Eclipse (un standard de l'industrie), en travail d'équipe, en techniques de rétro-conception et d'un approfondissement des connaissances en programmation orientée objet.

---

<sup>1</sup> Design Patterns. Elements of Reusable Object-Oriented Software. par Erich Gamma et al. Éditions Addison-Wesley.

<sup>2</sup> <http://ptidej.iro.umontreal.ca/>

## Introduction

Lorsque nous avons entrepris ce projet, nous avons tous les trois très peu de connaissances en ce qui a trait aux patrons de conception. Néanmoins décidés à bien faire, notre enthousiasme nous a, au départ, joué de vilains tours. Ce qui semblait une tâche relativement simple cachait en fait un grand besoin d'apprentissages.

En effet, pour parvenir à notre objectif, il nous a fallu bien nous organiser à la fois en terme d'échéancier et en terme de méthodologie. Il faut surtout admettre sans fausse pudeur que notre premier élan a été coupé net lorsque nous nous sommes empêtrés dans un domaine avec lequel nous n'étions décidément pas familiers.

C'est à la suite de cette réalisation, disons en un second départ, que nous avons pu matérialiser notre ambition: En prenant le temps d'étudier adéquatement ce que c'est qu'un patron de conception, à quoi ça sert, quels types existent et dans quels contextes chacun d'entre eux peut être utilisé, nous avons pu assimiler le *feeling* qui nous a ensuite servi à les reconnaître. La nécessité de procéder par étapes successives et correctement planifiées plutôt que de foncer tête baissée nous a en fait poussé, nous y reviendrons plus tard, à proposer la création d'un guide à l'intention des autres étudiants qui se lanceront dans ce projet à l'avenir. Nous sommes intimement convaincus que ne pas brûler les étapes évitera beaucoup de temps perdu à ceux-ci.

Dans ce document, nous aborderons en premier lieu l'organisation du temps, l'échéancier que nous avons mis sur pied pour structurer notre travail de même que la façon dont nous avons abordé le travail d'équipe.

Par la suite, nous détaillerons les méthodologies développées (nous en avons conçues plusieurs) afin de détecter manuellement ou semi automatiquement les patrons de conception dans le code java.

L'analyse des logiciels employés, bons ou mauvais, ayant été mainte fois développée par nos prédécesseurs, nous n'effectuerons ici qu'un bref survol de ceux qui nous ont réellement servi.

Enfin, nous présenterons un résumé des résultats. Les résultats concrets étant fournis au professeur sous la forme d'un fichier XML, nous nous contenterons dans ce document d'en extraire les faits saillants et quelques statistiques.

Pour terminer, nous conclurons en deux temps, avec un rappel sur notre expérience, nos conclusions globales et nos appréciations individuelles.

Avant d'aborder le vif du sujet, nous tenons ici à remercier de sa chaleureuse collaboration Duc-Loc Huynh, membre du projet Ptidej, à qui nous devons beaucoup en temps que parrain (en quelque sorte) de notre équipe de travail et notre professeur Yann-Gaël Guéhéneuc pour sa chaleureuse jovialité qui a su nous rassurer lorsque le manque de résultats nous inquiétait.

## Organisation du temps

Voici l'échéancier de travail que nous avons suivi. Il sera détaillé par la suite.

Avril	3	4 Rencontre initiale	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19 Mini rencontre	20	21	22
23	24	25	26	27	28	29
30	<b>Mai</b>	2	3 Réunion hebdo Rencontre prof organisation générale	4 organisation générale	5 organisation générale	6 organisation générale
7 organisation générale	8 organisation générale	9 organisation générale	10 Réunion hebdo Rencontre prof Design patterns Gantt Project	11 Design patterns Gantt Project	12 Design patterns Gantt Project	13 Design patterns Gantt Project
14 Design patterns Gantt Project	15 Design patterns Gantt Project	16 Design patterns Gantt Project	17 Réunion hebdo	18 Design patterns Gantt Project	19 Design patterns Gantt Project	20 Design patterns Gantt Project
21 Design patterns Gantt Project	22 Design patterns Gantt Project	23 Design patterns Gantt Project	24 Réunion hebdo	25 Design patterns Gantt Project	26 Design patterns Gantt Project	27 Design patterns Gantt Project
28 Design patterns Gantt Project	29 Design patterns Gantt Project	30 Design patterns Gantt Project	31 Réunion hebdo Rencontre prof Analyses de code	<b>Juin</b> Analyses de code	2 Analyses de code	3 Analyses de code

4	5	6	7 Réunion hebdo	8	9	10
Analyses de code	Analyses de code	Analyses de code	Analyses de code	Analyses de code	Analyses de code	Analyses de code
11	12	13	14 Réunion hebdo	15	16	17
Analyses de code	Analyses de code	Analyses de code	Analyses de code	Analyses de code	Analyses de code	Analyses de code
18	19	20	21 Réunion hebdo	22	23	24
Analyses de code	Analyses de code	Analyses de code	Analyses de code	Analyses de code	Analyses de code	Analyses de code
25	26	27	28 Réunion hebdo	29	30	Juillet
Analyses de code	Analyses de code	Analyses de code	Analyses de code	Analyses de code	Analyses de code	Analyses de code
2	3	4	5 Réunion hebdo	6	7	8
Analyses de code	Analyses de code	Analyses de code	Analyses de code	Analyses de code	Analyses de code	Analyses de code
9	10	11	12 Réunion hebdo	13	14	15
Analyses de code	Analyses de code	Analyses de code	Analyses de code	Analyses de code	Analyses de code	Analyses de code
16	17	18	19 Réunion hebdo Rencontre prof	20 Analyse nouveaux code projets	21 Analyse nouveaux code projets	22
Analyses de code	Analyses de code	Analyses de code				
23	24 Analyse nouveaux code projets	25 Analyse nouveaux code projets	26 Réunion hebdo Séparation projets	27 Travail individuel	28 Travail individuel	29
30	31 Travail individuel	Août	2 Travail individuel	3 Travail individuel	4 Travail individuel	5
6	7 Travail individuel	8 Travail individuel	9 Travail individuel	10 Travail individuel	11 Travail individuel	12

13	14	15	16	17	18	19
20	Correction Rédaction	Correction Rédaction	Correction Rédaction	Correction Rédaction	Correction Rédaction	26
27	Correction Rédaction	Correction Rédaction	Correction Rédaction	Correction Rédaction	<b>Septembre</b> Correction Rédaction	2
3	4	5	6	7	8	9
10	11	12	13 Réunion de relecture ...	14 Présentation finale Remise du rapport Fin	15	16

- Au mois d'avril, nous nous sommes rencontrés deux fois afin de choisir le projet et former l'équipe.
- Au début mai, les 3 et 10, nous avons rencontré le professeur pour établir quels étaient les objectifs du projet et ensuite pour détailler sur quels blocs de code nous allions spécifiquement travailler et dans quel environnement de travail cela allait se faire.
- L'ensemble du mois de mai a servi à d'abord nous familiariser avec le concept de Design Patterns, nous organiser comme équipe et entamer le travail en s'attaquant au volumineux code du Gantt Project.
- Au mois de juin, après avoir de nouveau rencontré le professeur, celui-ci nous a fourni de nouveaux noms de projet et nous nous sommes lancés dans leur analyse

(JSettler, JHotDraw et Risk), mais ayant développé alors de nouvelles approches, nous avons également revisité le très gros Gantt Project. Peu après nous avons reçu une liste élargie de projets à étudier.

- Après notre rencontre à la mi-juillet avec le professeur, nous avons déjà produit des résultats de qualité, mais le nombre des patrons découverts restait mince, nous avons donc opté pour étendre la durée initiale du projet (prévue pour prendre fin à la fin juillet) jusqu'à la mi-septembre. Il était alors entendu que cela permettrait de visiter les autres projets dont le code nous avait été fournis.
- C'est au mois d'août que nous avons élaboré le petit guide pratique, ceci après avoir fait part à notre professeur du constat selon lequel il était dommage que chaque équipe travaillant sur ce projet doive repartir sur des bases neuves, sans vraiment de références précises. Bien sûr il y a les conseils et les rapports des précédentes équipes, mais une feuille de route, du moins pour entamer le travail, nous semblait être une nécessité pour obtenir une meilleure productivité.

## Méthodologies

Au cours de nos travaux, nous avons utilisé/développé 7 méthodologies de recherche de micro-architectures distinctes. Certaines ont été inspirées de nos prédécesseurs ou par de précieux conseils reçus, mais d'autres sont de notre cru. Les voici, cotées par des étoiles exprimant notre estimation de leurs efficacités relatives:

### **Recherche par mots-clés:** \*

Les programmeurs commentant souvent leur code (on l'espère), on peut détecter certains patrons de conception facilement en recherchant dans les commentaires des mots clés tels "Factory" ou autres parmi les nom des patrons ou les rôles joués par certaines classes dans ceux-ci. Aussi, une bonne pratique de programmation étant de donner un nom significatif aux classes que l'on défini, la recherche par mots clés nous amène aussi à découvrir des indications utiles dans le nom même de celles-ci.

Il faut cependant faire attention ici, car le vœu du programmeur ne fait pas foi de l'implémentation réalisée. Cette méthode demande donc une vérification par une autre stratégie ou à tout le moins une bonne vérification des assertions de l'auteur.

### **Recherche des interfaces:** \*\*\*

Par mot clé on peut également chercher toutes les "INTERFACE" du programme ou du package. Une fois qu'on a obtenu la liste de celles-ci, on peut en choisir une et analyser quelles sont les classes qui dérivent de cette interface et leur relations pour en extraire un patron. Notons que presque tous les patrons de conception ont une INTERFACE en racine.

### **Les singletons:** \*

Ne sachant pas au début qu'ils étaient de peu de valeur, nous avons développé une méthode de détection très simple (et très efficace) pour les Singleton. Il suffit de faire une recherche sur "==NULL", cette expression étant toujours (...) utilisée au moment d'instancier l'objet en question dans ce patron. Ce test étant modérément utilisé dans d'Autres contexte, ça en fait un outil valable et rapide.

**Relations inter patrons: \*\*\***

En se fiant à la page de couverture en fin de livre (ou à la liste des patrons de notre petit guide), on peut voir les liens qui existent souvent entre les patrons de conception. Ceci nous a donc amené à étendre nos recherches dès qu'un patron était détecté dans du code, afin de déterminer si certains autres pouvaient être décelés «à proximité». Cette méthodologie, bien que développée/apprise un peu tard est en fait essentielle pour être vraiment efficace dans la détection des patrons de conception. En effet, la plupart de ceux-ci, surtout parmi les plus évolués, sont intrinsèquement liés à certains de leur congénères et il serait fou de ne pas en tirer partie.

**À partir d'une petite classe: \*\***

En trouvant une petite classe dans le code du projet étudié, on peut en faire une analyse «en la développant». On recherche et note ensuite les classes qui l'instancient et que celle-ci instancie. (Recherche par mots-clés sur "new cetteclasse" et les new dans le code de cette classe-ci.) On illustre en parallèle sur une feuille un schéma des classes que l'on lie peu à peu. Il faut évidemment faire attention de ne pas tout noter (par exemple les classes natives de Java), mais bien les classes qui ont un lien logique étroit avec la première. Il ne restera alors qu'à comparer le schéma avec les diagrammes de structure des patrons du livre. En ayant pris soin auparavant d'analyser un peu le code et son utilité, on peut restreindre la recherche à une seule catégorie de patrons.

**À partir d'une grosse classe: \*\***

Lorsqu'on trouve une classe qui contient beaucoup de commentaires ou qui a un code plus long que la moyenne, on peut décider de prendre quelques instants pour en analyser la fonctionnalité, l'utilité, les objectifs. On peut alors, à l'aide du livre ou d'une liste de référence reconnaître des exemples types. On peut alors essayer de voir si la classe est en effet partie prenante d'un patron réalisant cet objectif. Parfois aussi ce sera le nom d'un rôle dans un patron qui deviendra évident, il ne restera plus qu'à retrouver à quel patron il appartient.

**Omondo:**

\*\*\*\*

*(ou tout autre analyseur UML)*

En affichant le diagramme des classes sur un package, on peut directement tenter de reconnaître le schéma de structure des patrons (via le livre). Il faut toutefois faire attention au fait que le logiciel ne permet que de voir le contenu d'un seul package à la fois, ce qui peut scinder des patrons et les rendre inidentifiable. Une façon de contourner le problème est de créer un package vide et d'y glisser des classes une à une qui proviennent des autres packages. Cela peut toutefois entraîner beaucoup de tâtonnements, mais par contre être un très bon outil de vérification.

Il est à noter que c'est lors du développement de certaines de ces méthodologies que nous avons cru bon nous faire une petite liste rapide de référence sur les patrons. Cette dernière a été intégrée au petit guide d'aide à la détection de patrons fournis en annexe.

## Les logiciels

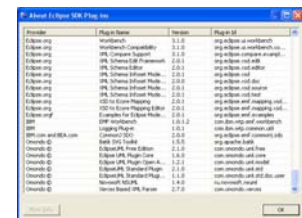
Voici un bref aperçu des logiciels qui nous ont servi concrètement à la réalisation de notre projet. Bien sûr il y en a d'autres qui auraient pu nous servir. Les équipes qui nous ont précédé ont fait un excellent travail d'analyse sur chacun d'eux, mais en ce qui nous concerne, il ne nous semblait pas pertinent de réitérer cette dernière si dans les faits le logiciel n'était pas «efficace». Dans le petit guide en annexe, l'analyse d'autres logiciels ayant été utilisés par nos prédécesseur a toutefois été intégrée afin de laisser plus de flexibilité.

### Eclipse<sup>3</sup>

Comme nous travaillons dans un univers objet, et par le fait même que le code étudié est le java, le programme de choix recommandé pour l'étude des patrons de conception est évidemment Eclipse. Il est déjà en place sur les machines du labo, mais suite à certains problèmes de configuration, nous avons préféré le réinstaller sur nos espaces personnels (Disque R:\) afin de passer outre le besoin d'assistance technique pour la configuration. Après quelques aléas<sup>4</sup>, nous avons pu configurer le CVS et récupérer les données nécessaires.



Les principaux avantages du travail avec Eclipse sont, dans un premier temps, le haut niveau d'intégration des outils complémentaires tels les plugins. En effet, une fois correctement configuré, le fait que tous les fichiers du CVS soient disponibles d'un simple clic, déjà accessibles, déjà au bon endroit simplifie grandement la «mise en place». La quantité de plugins disponibles sur Internet est aussi, sans conteste, un atout de poids... le fait qu'IBM en ait fait sa plateforme principale de développement (WebSphere) y contribue certainement. Parmi ces plugins, nous avons utilisé Omondo, un outil incontournable.



<sup>3</sup> <http://www.eclipse.org> (version 3.1)

<sup>4</sup> Il serait intéressant qu'automatiquement les étudiants du groupe de génie logiciel se voient octroyer un espace disque supplémentaire par la DGTIC, la limite initiale de 60Mo étant insuffisante pour ce cas de figure.

Un autre côté très intéressant d'Eclipse, non le moindre, est la performance de l'outil de recherche. Sa versatilité et son excellent niveau de «configurabilité» nous ont permis d'effectuer des recherches beaucoup plus simplement.

Finalement, le *class viewer* offre une façon de voir la hiérarchie des classes de façon rapide, ce qui a son utilité.



## Omondo<sup>5</sup>

Ce plugin pour Eclipse permet d'afficher les schémas UML. C'est vraiment un outil très intéressant et recommandable. Nous avons commencé à nous en servir un peu trop tard pour visualiser les liens entre les classes que nous étudions. Ce délais nous a occasionné beaucoup d'efforts inutiles. Quand nous l'avons correctement intégré à notre méthodologie, il est devenu clair que nous pouvions maintenant, d'un coup d'œil (ou presque), nous donner une idée claire de la structure entourant une classe en particulier... Avec un peu de chance et de l'expérience, cela nous a permis de mettre en relief les similitudes avec des patrons de conception à partir de leurs schémas de structure directement.



Il a fallu cependant pallier à un défaut de celui-ci. Omondo ne permet pas de visualiser les interactions inter-packages, occultant ainsi des patrons possibles. Dans la description de la méthodologie Omondo que nous avons développé, nous avons décrit comment, en déplaçant les classes une à une dans un package vierge on pouvait tout de même s'en tirer.

## Oxygen<sup>6</sup>

Bien qu'il soit possible de manuellement mettre à jour les informations trouvées lors de nos recherches dans le fichier des résultats, il s'agit là d'un outil intéressant pour travailler directement les fichiers XML.

---

<sup>5</sup> <http://www.omondo.com>

<sup>6</sup> <http://www.oxygenxml.com>

## CVS

Évidemment il s'agit ici d'un logiciel «serveur». Mais nous en parlerons également pour mettre en relief l'usage spécifique que nous en avons fait. Bien sûr, en premier lieu il nous a permis d'accéder aux données déjà disponibles pour le groupe de travail Ptidej, mais bientôt nous avons aussi créé notre propre branche et avons rapidement intégré ses possibilités à notre effort: Nous pouvions directement y conserver nos commentaires sur l'évolution de nos méthodologies ainsi que sur les résultats obtenus, programme par programme. Ce fut très pratique, bien plus, en fait, que la procédure originale prévue consistant en l'utilisation d'un espace ftp commun.

Mentionnons toutefois ici que CVS nous semble, comme logiciel de cette nature, peut-être pas le meilleur choix: il a quelques limitations agaçantes, telles que le simple fait d'être dans l'impossibilité de renommer certaines branches. Une alternative intéressante pourrait être, par exemple, *Subversion*.

## Les résultats

Avant d'aborder la détection dans le code fourni comme telle, notons d'abord qu'à la suite de l'étude du livre de référence et des exemples de micro-architectures développées par Duc-Loc Huynh (il est vraiment très intéressant d'avoir accès à des exemples clairs en Java), nous avons cru bon d'offrir quelques améliorations à ces derniers qui sont déjà en place dans le CVS. Principalement nous avons modifié un peu le code de ceux-ci pour les rendre plus clairs et en rendre l'utilisation plus aisée.

Voici la liste des programmes qui ont été analysés:

Programme analysé	Classes	Creational	Structural	Behavioral	Patterns
GanttProject v1.10.2	616	3	2	0	5
HolubSQL v1.0	151	0	2	1	3
Jsettlers v1.0.5	255	2	0	2	4
Jtans v1.0	206	0	0	1	1
Lexi v0.0.1	216	2	0	0	2
Juzzle v0.5	99	0	0	0	0

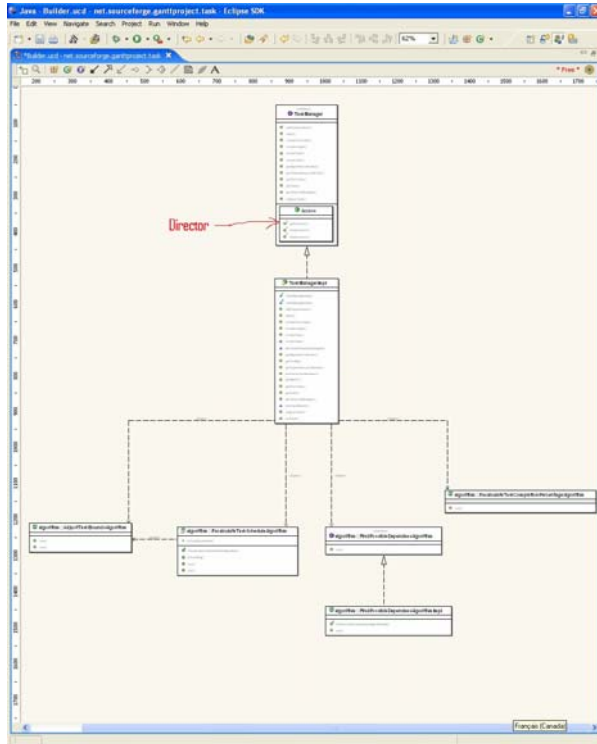
Il convient de mentionner que quelques uns des patrons de conception déterminés par nos efforts l'ont également été par l'équipe qui nous a précédé. À l'origine, à la lecture de leurs résultats, il nous était vite apparu que nous ne partagerions pas toutes leurs conclusions. En effet, plusieurs patrons que nous avons réexaminés nous sont apparus sous un autre jour et en vérifiant avec nos différentes méthodologies ces résultats, nous n'avons pas toujours tiré les mêmes conclusions. Par la suite, donc, nous avons tenté, sur le même code qu'eux de voir à quels résultats nous pouvions directement parvenir, pour ensuite comparer nos conclusions aux leurs.

Par exemple, dans **JTANS v1.0**, pour «`src.net.phbwt.jtans.calc`», nous en somme arrivé à la conclusion que le patron détecté par nos prédécesseurs un *Composite* n'en était pas un, puisque le lien d'héritage n'était pas confirmé. (Notes à ce sujet dans le fichier XML).

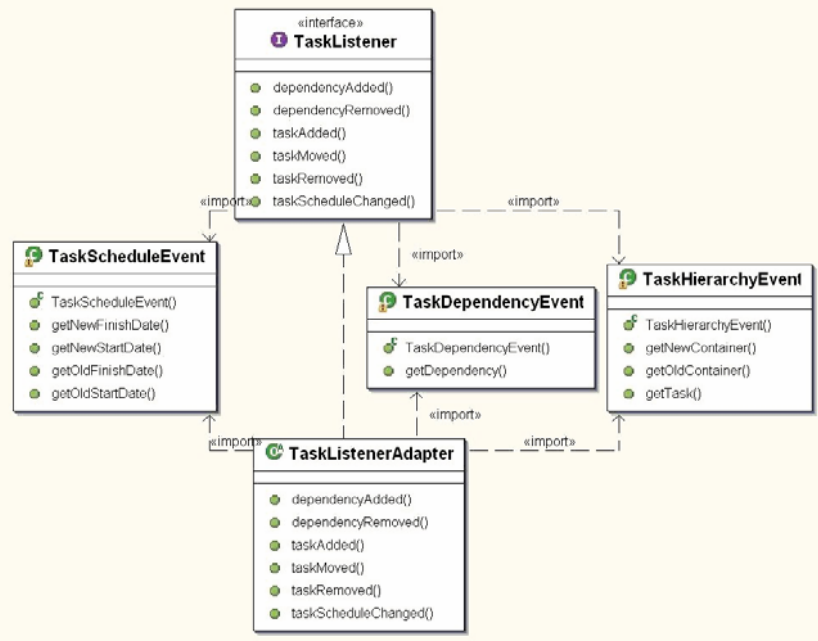
Mentionnons au passage que **Juzzle v0.5** est vraiment trop simple (lire petit) pour contenir un *Design Pattern*. Après un lecture intensive et exhaustive du code, nous n'avons pas réussi à déterminer un patron quel qu'il soit.

Voici maintenant quelques schémas tirés de notre analyse du GanttProject (y compris la *Façade* dont nous parlions plus haut:

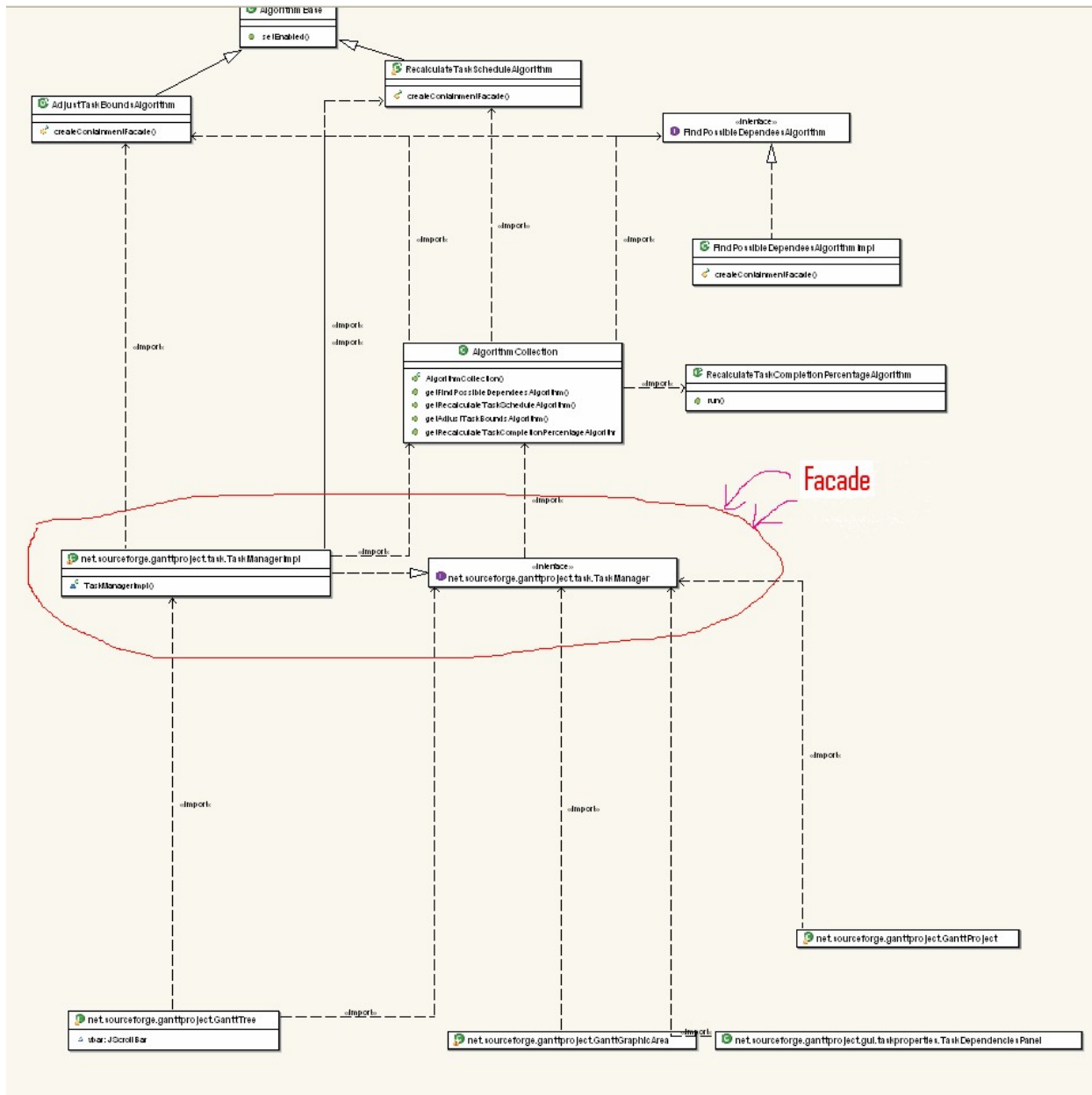
Ici on voit un *Builder* ayant pour Director Access (TaskManager) , tel qu'indiqué:



Nous voyons ci-contre un *Adpter* à partir de l'interface TaskListener.



Finalement, ici voici la *Facade* découverte dans Gantt autour de TaskManagerImpl.



## Conclusion

Au cours de notre périple, un élément parmi tous les autres s'est de lui-même imposé comme une évidence. La recherche de patrons de conception, en l'occurrence «manuellement», est un travail fastidieux et plein d'ambiguïtés. En effet, chaque *détection* dépend intrinsèquement de son contexte. Un *Design Pattern*, ça n'existe pas dans un *vacuum* et donc la forme qu'ils prennent est très liée au programmeur qui l'a mis en place et à sa manière de programmer.

Certains patrons sont relativement clairement indiqués et on peut conclure que l'auteur du code avait l'intention de structurer son code à la manière prescrite par le *Gang of Four*. Par contre, souvent il n'est même pas possible de discerner un seul patron dans une masse de code et il devient alors évident que la préoccupation du ou des auteurs du code était beaucoup plus dans la simplification de leur approche que dans l'idée de produire un code aisément maintenable ou très efficace. Dans l'ensemble, la détection de patrons est donc un exercice long et qui nécessite beaucoup de vérifications. Nous avons donc développé une nouvelle appréciation du travail impliqué pour la réalisation de l'ensemble du projet Ptidej et de son importance.

Le travail en lui-même sur cette quantité appréciable de code provenant de divers horizons nous aura à tout le moins apporté une meilleure appréciation de l'importance de produire un code clair et commenté. Nous avons sans doute appris plusieurs façons de procéder dans la création de micro-architectures. Aussi il est important de mentionner qu'incontestablement, nous avons grandement évolué dans notre compréhension de ce que c'est un *Design Pattern*, leur importance, quand et comment les utiliser. Nous ne sommes sans doute pas des experts, mais nous sommes certainement maintenant de meilleurs programmeurs.

Tout au cours de notre projet, nous avons conservé à l'esprit une vision de qualité (Notre professeur ayant fortement insisté sur le sujet). C'est donc avec fierté que nous présentons nos résultats, forts de l'exactitude de ceux-ci et espérant que leur utilisation pourra aider à la réalisation du projet mère.

Enfin, pour terminer, c'est dans un esprit de travail d'équipe que nous avons eu l'idée et avons développé *Le petit guide pratique*. Nous espérons franchement qu'il saura éviter des embûches biens inutiles à nos collègues et qu'ainsi ils pourront à leur tour mettre leur épaule à la roue et faire avancer nos connaissance en programmation, mais cette fois, avec un peu plus de facilité.

## Commentaires personnels

### **Abdeljabar:**

Au départ, la détection des patrons de conceptions me semblait facile vu ce qu'on a appris au cours de génie logiciel (IFT2251); mais en franchissant la porte de commencement, je me suis trouvé devant un énorme travail, dont je ne savais même pas de quoi commencer, et comment le faire. Donc au premier moment, j'ai commencé à me familiariser avec les différents patrons dans le livre de Partha Kuchana<sup>7</sup>. Personnellement, ce livre m'a beaucoup aidé. Il m'a donné une idée d'ensemble sur les patrons et leurs architectures sous forme de schémas et aussi sous forme d'exemples de programmes JAVA (Code). Après, et avec l'aide de quelques personnes qui fréquentent le laboratoire de génie logiciel (par exemple M. Loc), on a pu se lancer au boulot. Enfin je me suis trouvé capable de chercher les patrons dans n'importe quelle architecture.

Le plus important, selon moi, pour bien réussir un projet de ce type (d'après l'expérience que j'ai vécu) il ne faut jamais hésiter à demander l'aide des autres. Il faut faire parler même les gens qui semblent non sociables.

### **Nicola:**

Personnellement, lorsque j'ai entrepris ce projet, je *croyais savoir* ce qu'étais un Design Pattern. Je dis bien *croyais savoir* parce le cours de génie logiciel de base que nous avons en premier au baccalauréat (IFT2251) ne nous prépare en rien à la complexité et à la diversité de ceux-ci. Alors disons qu'en plus d'une humilité certaine face à ce domaine complexe qu'est la science du génie logiciel, j'aurai appris énormément sur les structures qui ont été développées pour résoudre les problèmes en programmation. Maintenant, je sais que j'aurai l'instinct, lors de l'élaboration d'un projet de programmation, de chercher d'abord à déterminer quelle est la meilleure micro-architecture pour obtenir le résultat le plus cohérent et efficace possible. Pour cela surtout je suis très content d'avoir accompli cette tâche.

---

<sup>7</sup> «Software Architecture Design Patterns in Java», voir la note bibliographique. Il s'agit d'un document se trouvant sur le CD.

D'un point de vue humain, je suis aussi très heureux d'avoir participé à ce groupe de travail. J'ai toujours été ouvert à tous et chacun et aux opinions diverses, mais là franchement, j'ai trouvé nos discussions (quelque fois sur le génie logiciel, mais aussi sur les religions et la politique) plus qu'enrichissantes. J'en ressors donc avec de meilleures appréciations à bien des niveaux, techniques et humains, et avec de nouvelles habiletés. Merci mes amis!

### **Rafik:**

Pour commencer une lettre ou un paragraphe, c'est parfois un peu difficile, parce qu'on se dit qu'on pourrait trouver mieux ou plus approprié ou tout simplement parce qu'on ne trouve pas. Une solution que je propose à ce problème, qui sera après directement reliée aux patrons de conception, est de commencer la lettre et après si on a le temps revenir dessus ou de directement écrire et passer à autre chose.

Pourquoi je raconte tout ça? Pas pour écrire une lettre bien sûr, mais juste pour vous avouer qu'un des plus grands problèmes que j'ai eu lors de ce projet était le fait de ne pas savoir quand m'arrêter ou quand est-ce que le travail était présentable. (bon? très bon? mauvais?) J'étais perdu, parfois je montrais mon travail (qui à mon avis était pas mal) à des personnes qualifiées, comme Duc et il me disait que c'était nul tandis que d'autre fois non, je présentais quelque chose en me disant «c'est n'importe quoi» et que je ne ferai que déranger Duc dans son sommeil au labo, mais non il me disait que c'était excellent.

Tous ça pour dire qu'il faut d'abord savoir commencer! Et surtout tout noter, même les choses qu'on trouve un peu faciles ou un peu bizarres et bien sûr savoir s'arrêter un jour! Et pour ça, il n'y a pas de recette, il faut parler avec son professeur et voir quelles sont ses attentes: quantité? qualité? rapport? travail en équipe? Chaque professeur est différent et généralement chacun veut quelque chose de différent, alors il faut aller consulter. Je dirai donc: osez déranger souvent 😊!

## Bibliographie

### Livres:

Gamma Erich et al., *Design Patterns : Elements of Reusable Object-Oriented Software*, Addison-Wesley, Indianapolis IN USA, 1995, 395 pages

### Documents Internet:

Kuchana Partha, *Software Architecture Design Patterns in Java*, Auerbach, USA, 476 pages

Deitel Deitel Nieto Lin & Sadhu, *How to program XML*, Deitel, 968 pages.

## Annexe

Ci-joint vous trouverez le «Petit guide pratique» que nous avons rédigé à l'intention d'éventuels étudiants reprenant ce projet.