IFT3051 – PROJET EN INFORMATIQUE – ÉTÉ 2005

Université De Montréal

Département d'Informatique et de Recherche Opérationnelle

IFT3051-E05

Projet en Informatique

Rapport Final

Ptidej-UI Standalone (v2.0) + Ptidej Multilingue

Travail présenté à :

M. Guéhéneuc Yann Gaël

M. Stefan Monnier

Par :

Lahlou Mehdi LAHM16068108

http://www.iro.umontreal.ca/~lahloume

Date de remise : 08-07-2005

TABLE DES MATIÈRES

TABLE DES MATIÈRES

I. REMERCIEMENTS

Je tiens à remercier M. Guéhéneuc, mon encadreur de projet, pour son excellent support ainsi que sa disponibilité. Son expertise m'a été très utile.
Travailler sous sa direction était très agréable.

Je remercie aussi Mme Sung Hui Park, coordonnatrice, ainsi que M. Stefan Monnier, directeur du cours pour leur supervision.

II. INTRODUCTION

Le premier contact qu'un utilisateur puisse avoir avec une application quelconque, est certainement, l'interface usager (UI : "User Interface"). Dans le cas d'un logiciel, cette dernière se traduit souvent par une interface graphique (GUI : "Graphical User Interface").
Or, ce premier contact est un facteur de poids jouant sur l'idée globale que se fait l'usager par rapport à l'application en entier.
Par analogie, une personne voulant s'acheter un objet quelconque, va tout d'abord avoir un contact visuel qui, d'un simple coup d'œil, va attirer ou au contraire, éloigner cette dernière de l'objet en question. Pour certains, ce facteur peut se voir attribuer un niveau d'importance plus bas, mais détrompez-vous, malgré nous, ce facteur est d'une transparence inégalé. Simple nature humaine !

Après, vient l'analyse de l'interface usager qui consiste en l'ergonomie, l'usabilité, la prise en main rapide, la conformité et la possibilité de personnalisation.
Heureusement pour le concepteur, certains de ces points sont fortement couplés entre eux (Ex : prise en main rapide VS conformité).

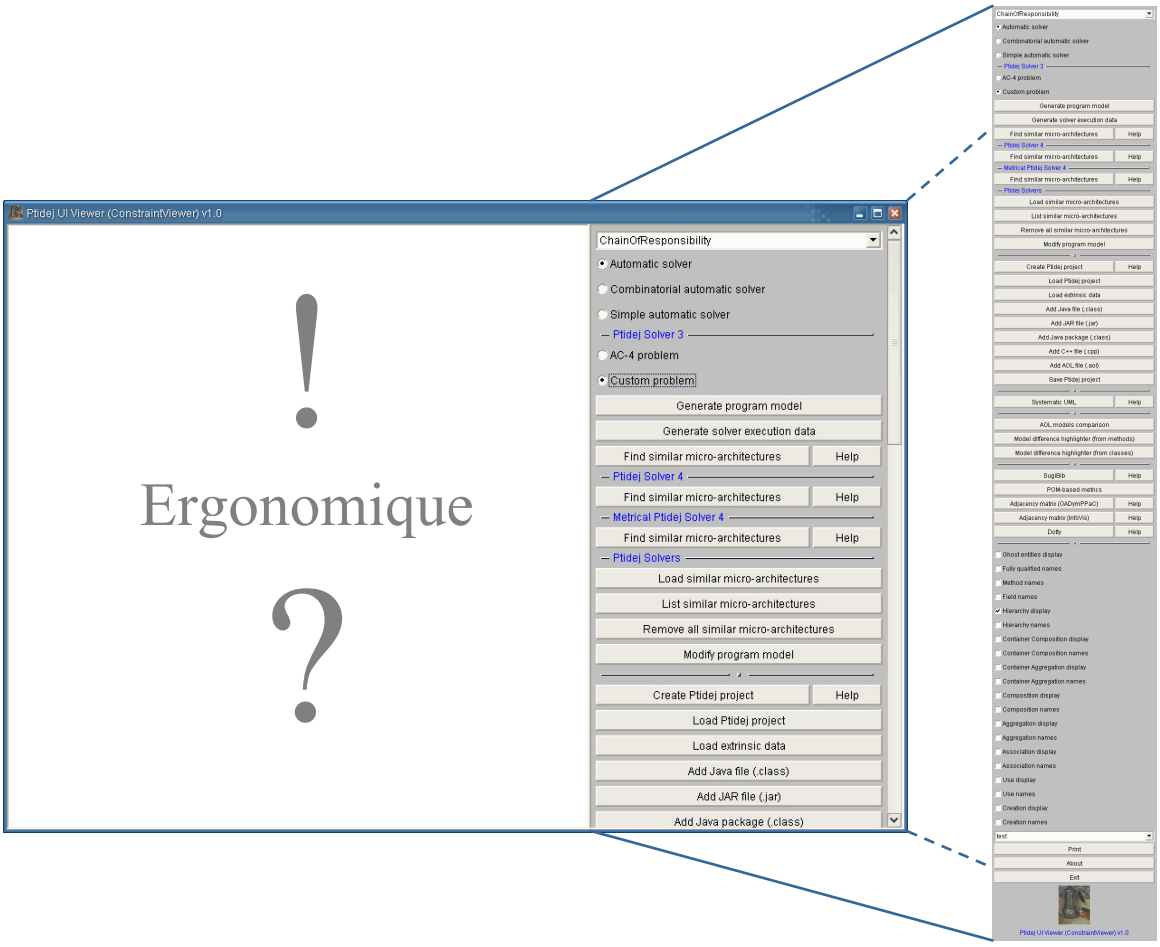
Pour rejoindre un plus grand nombre d'usagers, un autre facteur à considérer, serait que l'application soit multilingue. En effet, nous ne parlons pas tous la même langue, du moins, nous n'en maîtrisons que peu d'entre elles. Offrir donc un logiciel ayant un haut degré d'internationalisation, n'est pas un facteur à négliger. Non seulement la langue mais la culture ainsi que les formats localisés constituent un logiciel "international".

Il est donc important pour un concepteur de prendre en compte tout ces facteur lors de la conception architecturale. Car en omettant ces derniers, la facture de mise à niveau et de maintenance peut se voir gonflée considérablement. Ce qui a été le cas pour la suite d'outils Ptidej, ce qui a donc donné naissance au présent projet.

III. Le problème « Ptidej UI (v1.0) »

La suite d'outils Ptidej "Pattern Trace Identification, Detection ans Enhancement in java" Est, sans aucun doute, une application innovante, offrant une multitude de fonctionnalités intéressantes dans le domaine du génie logiciel, précisément, elle traite des patrons de conceptions.

Malheureusement, son interface graphique laisse à désirer. En effet elle est dotée d'une interface unilingue, peu intuitive, et d'un espace de travail rudimentaire.
De plus, elle ne se conforme ni aux principes d'ergonomie, ni à ceux de l'usabilité.



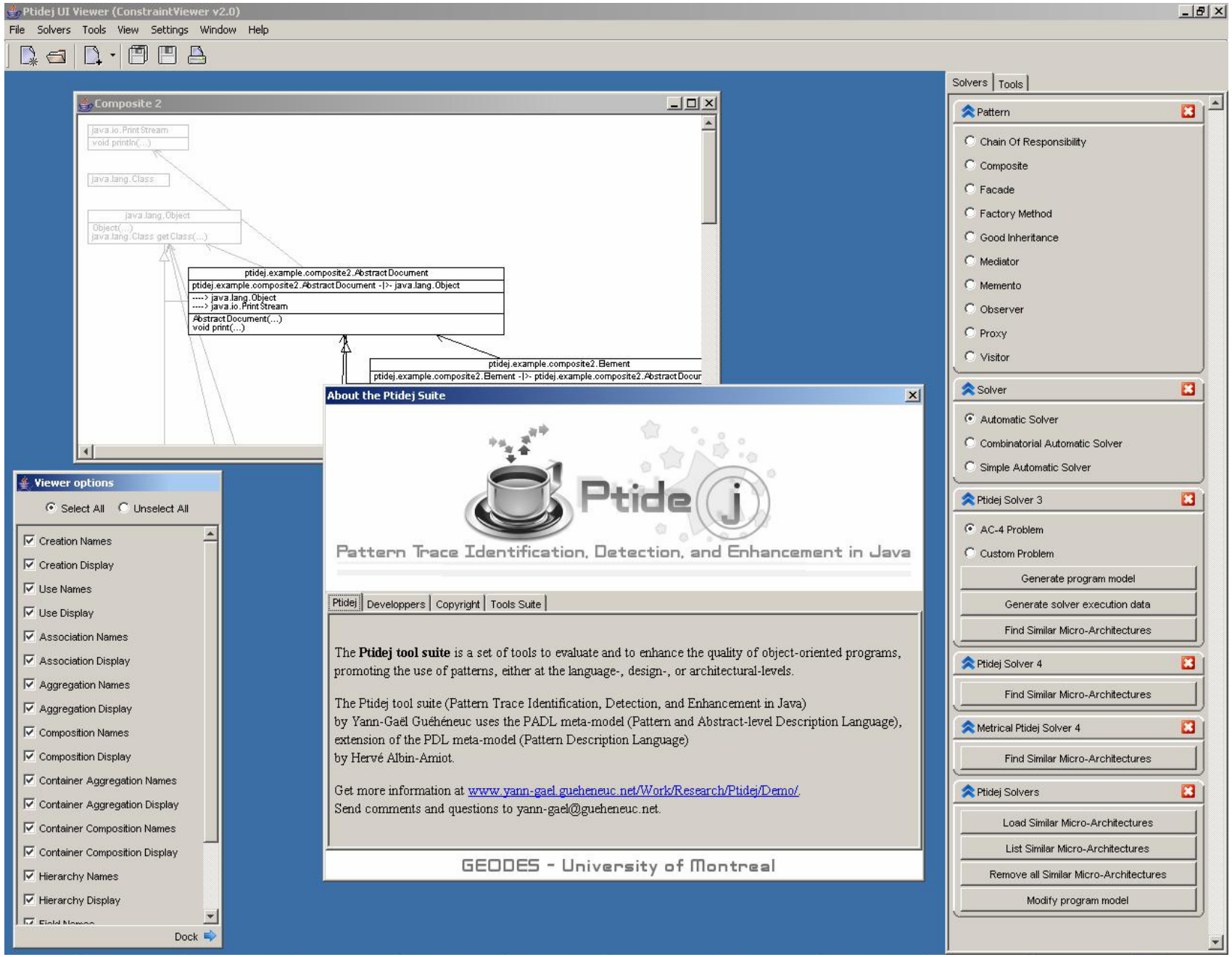
Remarquons le fait que toutes les commandes sont regroupées dans un même panneau. À chaque choix de commande, l'utilisateur doit utiliser l'ascenseur pour accéder au déclencheur de cette dernière.
De plus, le fait que l'interface graphique manque de modularité, rend l'application Plus difficile à comprendre, et donc, le temps de prise en main devient encore plus important.

IV. La Solution « Ptidej UI (v2.0) »

Mettons en valeur la suite d'outils Ptidej, et rendons la plus agréable à utiliser.
Comment ?
La conception d'une nouvelle interface ergonomique, intuitive, multilingue et dont l'espace de travail sera plus élaboré, est sans aucun doute, la solution la plus appropriée.

Une étude préalable est donc requise pour tirer les points forts de l'ancienne interface et veiller à les prendre en considération lors de la conception de la nouvelle interface graphique. De l'autre côté il faudra tirer chacune des lacunes de cette même interface et chercher à les éviter lors de la conception de la nouvelle interface.

Ci–dessous un aperçu de la nouvelle interface graphique :


Ptidej UI Viewer (ConstraintViewer v2.0)
File Solvers Tools View Settings Window Help
Composite 2
java.io.PrintStream
void println(...)
java.lang.Class
java.lang.Object
Object(...)
java.lang.Class getClass(...)
ptidej.example.composite2.AbstractDocument
ptidej.example.composite2.AbstractDocument -|>- java.lang.Object
----> java.lang.Object
----> java.io.PrintStream
AbstractDocument(...)
void print(...)
ptidej.example.composite2.Element
ptidej.example.composite2.Element -|>- ptidej.example.composite2.AbstractDocur
About the Ptidej Suite
Ptide j
Pattern Trace Identification, Detection, and Enhancement in Java
Ptidej
Developpers
Copyright
Tools Suite
The Ptidej tool suite is a set of tools to evaluate and to enhance the quality of object-oriented programs, promoting the use of patterns, either at the language-, design-, or architectural-levels.
The Ptidej tool suite (Pattern Trace Identification, Detection, and Enhancement in Java) by Yann-Gaël Guéhéneuc uses the PADL meta-model (Pattern and Abstract-level Description Language), extension of the PDL meta-model (Pattern Description Language) by Hervé Albin-Amiot.
Get more information at www.yann-gael.gueheneuc.net/Work/Research/Ptidej/Demo/.
Send comments and questions to yann-gael@gueheneuc.net.
GEODES - University of Montreal
Viewer options
Select All
Unselect All
Creation Names
Creation Display
Use Names
Use Display
Association Names
Association Display
Aggregation Names
Aggregation Display
Composition Names
Composition Display
Container Aggregation Names
Container Aggregation Display
Container Composition Names
Container Composition Display
Hierarchy Names
Hierarchy Display
Dock
Solvers
Tools
Pattern
Chain Of Responsibility
Composite
Facade
Factory Method
Good Inheritance
Mediator
Memento
Observer
Proxy
Visitor
Solver
Automatic Solver
Combinatorial Automatic Solver
Simple Automatic Solver
Ptidej Solver 3
AC-4 Problem
Custom Problem
Generate program model
Generate solver execution data
Find Similar Micro-Architectures
Ptidej Solver 4
Find Similar Micro-Architectures
Metrical Ptidej Solver 4
Find Similar Micro-Architectures
Ptidej Solvers
Load Similar Micro-Architectures
List Similar Micro-Architectures
Remove all Similar Micro-Architectures
Modify program model

V. L'internationalisation

1. La gestion du Multilingue :

L'internationalisation de logiciel, c'est le processus permettant à un logiciel d'être adapté à plusieurs langues et régions de manière facile, à moindre coût et sans avoir à changer l'architecture du logiciel à chaque ajout de langue.
Cela peut être conçu à l'aide du principe de localisation.
Dans la plateforme Java 2, le support de l'internationalisation est entièrement intégré aux classes et paquetages fournis, en utilisant le principe de localisation.

Supposons qu'on doit écrire un programme pour afficher les messages comme suit:

```
public class Sample {
   static public void main(String[] args) {
     System.out.println("Bonjour,");
     System.out.println("Je communique en Français.");
   }
}
```

Mais si nous voulions que notre programme « Sample » puisse afficher le même texte mais dans la langue de l'utilisateur (Français, Anglais, Espagnol, Arabe) !
Il nous faudrait récolter tous les messages à traduire, les mettre dans des fichiers que les traducteurs peuvent traduire, procéder avec la traduction, et, finalement échanger ces derniers dans notre programme source. Fastidieux !
Qu'en est-il pour l'ajout d'une deuxième, troisième ou dixième langue … ?!

Java a intégré quelques méthodes pour offrir le service du multilingue à ces applications. Selon SUN, une méthode d'implémentation simple et appropriée dont la maintenance reste à faible coût, est celle de l'utilisation des ResourceBundle.

Cette méthode consiste à emmagasiner tout ce qui est dépendant de la langue est culture ciblées dans des fichiers séparés (un fichier = [une langue] et/ou [un pays]).
Le programme java prend pour responsabilité de charger le fichier adéquat aux langue et pays du système d'exploitation (ces paramètres constituent le locale du système d'exploitation, ie : en_CA "anglais Canada", fr_CA "français Canada", en_US "anglais États unis", fr_FR "français France").

Nous avons le choix d'utiliser, ou bien, des fichiers ".properties" et invoquer leur contenu de la manière :

```
ResourceBundle localisedResources = ResourceBundle.getBundle("FicherProp",Locale.getDefault());
String resource = localisedResources.getString("resourceKey");
```

Ou bien, de créer des classes java héritant de la classe ListResourceBundle fournie par les frameworks java :
(Ce sera cette méthode que nous allons implémenter pour éviter tout problème de résolution de fichier externe)

2. Création des sous-classes de ListResourceBundle :

Nous devons créer une classe par locale supporté. Pour que notre programme puisse supporter les "Locales", (par exemple : en_CA, fr_CA et ja_JP). Il faudra créer les trois classes de ressources comme suit :
SampleResourceBundle_en_CA.java
SampleResourceBundle_fr_CA.java
SampleResourceBundle_ja_JP.java
Le code source de " SampleResourceBundle_xx_XX.class " prendra la forme suivante :

```
import java.util.ListResourceBundle;

public class SampleResourceBundle_fr_CA extends ListResourceBundle{

 static final Object[][] contents = {
    {"Greetings", new String("Bonjour,")}
    {"CurrCommunicatingLanguage", new String("Je communique en Français.")}
    {"CurrCountry", new String("Et je suis configuré pour suivre les normes canadiennes.")}
    {"CountryPopulation", new Integer(29000000)}
 };

 public Object[][] getContents() {
    return contents;
 }

}
```

Pour chaque "Locale" désiré, il suffit donc de créer le fichier .java approprié et le tour est joué.

3. Spécification du Locale :

```
Locale frenchCanadianLocale = new Locale("fr", "CA");
```

Chaque objet "Locale" correspond à une des classes SampleResourceBundle.
Par Exemple, le Locale "Français Canadien" (fr_CA) correspond à la classe SampleResourceBundle_fr_CA.
Aussi, ce qui intéressant, c'est de pourvoir mettre une class par défaut, par exemple :
Si, on aimerait avoir en_US comme Locale par défaut de notre programme "Sample", nous n'avons qu'à créer une class SampleResourceBundle (sans le en_US) qui contiendra les ressources spécifiques à l'anglais Américain.

4. Création du ResourceBundle :

```
ResourceBundle resources = ResourceBundle.getBundle("SampleResourceBundle", currentLocale);
```

La méthode, getBundle, recherché la classe dont le nom commence par SampleResourceBundle suivi par la langue et le pays du Locale configuré précédemment.

Par exemple :

```
ResourceBundle resources = ResourceBundle.getBundle("SampleResourceBundle", frenchCanadianLocale);
```

Retournera le contenu du fichier SampleResourceBundle_fr_CA.class.

5. Application à Ptidej (Spécificités):

a. **Exportation des ressources** :

La première tâche consistait à parcourir les 1700 classes constituant le code source de la suite Ptidej, extraire les données, et leur faire associer une clé. Après mûre réflexion concernant la règle de nommage des clés, j'ai décidé de leur donner le format suivant :

" ProjectName::ClassName::ResourceTypeIfAny_RESOURCE_KEY "

b. **Création des clés** :

La raison de présence du **ProjectName**, est, qu'il soit possible d'avoir le même nom de classe mais dans des projets différents. Il faut donc pouvoir les différencier.

Pour le **ClassName**, la raison est que, nous aurons certainement des noms de clé identiques dans des classes différentes, ex : "PANEL_TITLE".

Le **ResourceTypeIfAny_** sert à donner un patron au type de la clé en question, ex : "CMD_***_TEXT" = clé pour identifier le texte d'un bouton,
"CMD_***_ICON" = clé pour identifier l'icône d'un bouton, …
C'est donc une façon de rendre transparente, la gestion du multilingue à l'interface graphique, pe : pour récupérer la touche de raccourci d'un élément de menu, nous pouvons procéder en invoquant une méthode telle que : getMenuItemMnemonic ("RESOURCE_KEY") qui va s'occuper de construire la clé effective, qui grâce à elle, elle pourra nous retourner le bonne chaîne de caractères.
Les méthodes de construction des clés se trouvent dans la classe PtidejConstants sous le package utils. Cette même classe contient les clés de l'application graphique. La raison est simple. Alors qu'il est possible à l'utilisateur d'accéder à une commande à partir de plusieurs sources différentes, ie : LOAD_FILE, cette même ressource pourrait éventuellement servir à récupérer la chaîne de caractères en question est donc, à partir des source différentes nous pourrons récupérer le contenu de la clé comme suit :

```
itemLoadFile.setIcon(PtidejConstants.getMenuItemIcon(PtidejConstants.LOAD_FILE))
cmdLoadFile.setIcon(PtidejConstants.getButtonIcon(PtidejConstants.LOAD_FILE))
```

On se rend alors compte que nous pouvons configurer n'importe quel composant en entier en utilisant une et une seule clé. Ex : addButton(PtidejConstants.**LOAD_FILE**);

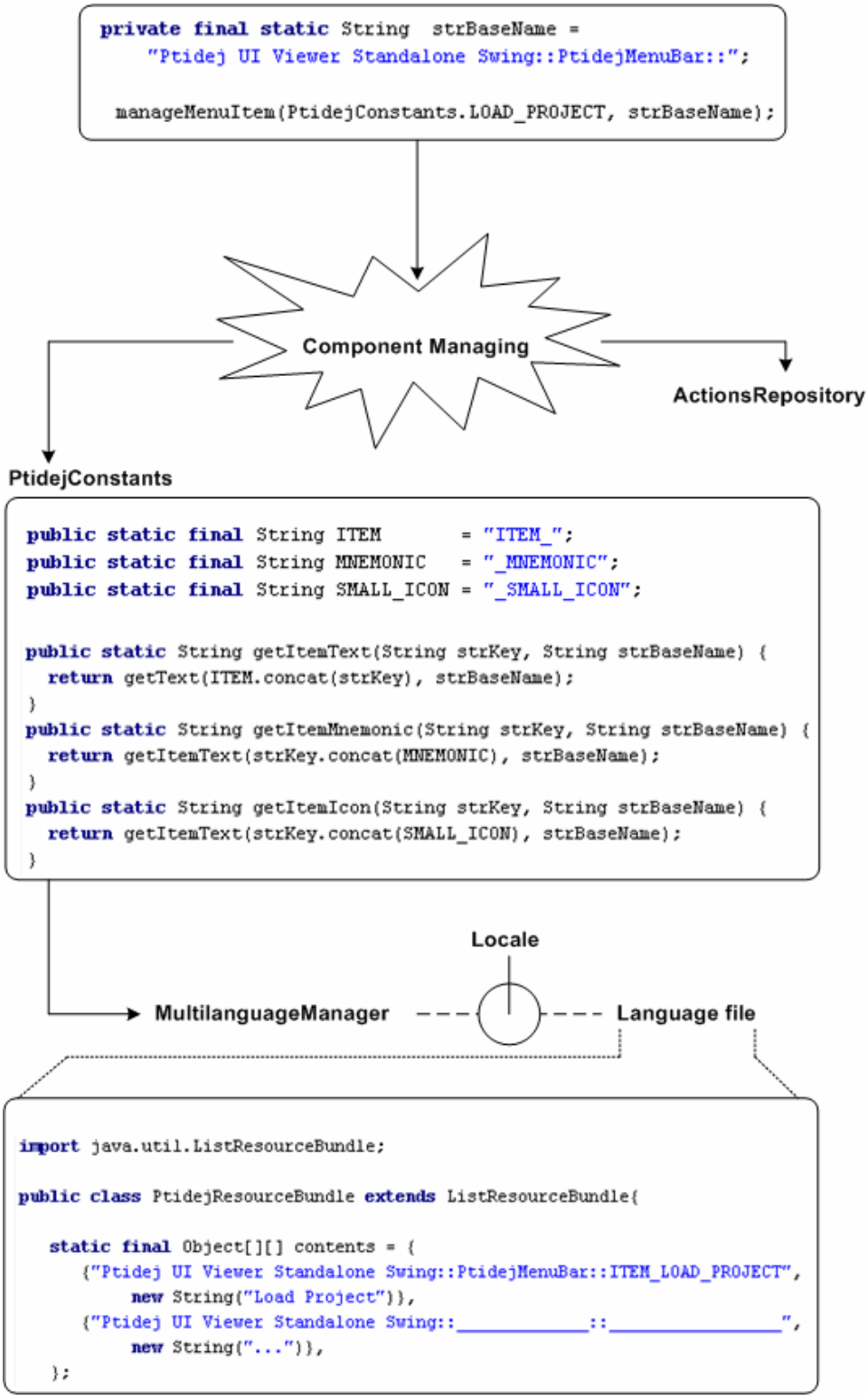
c. **Gestionnaire de multilingue** :

Un MultilanguageManager à été mis en place pour gérer la récupération de contenu des clés. Il représente un pont d'accès aux fichiers de ressources à partir de l'application :

Un simple appel à une des ses méthodes "getStrResource", retourne le contenu de la clé en question. Il est important de lui fournir la "RESOURCE_KEY" ainsi que le "RESOURCE_BASE_NAME" qui constitue la partie " ProjectName::ClassName::" de la clé. Avec ces informations, il peut donc consulter la langue présentement

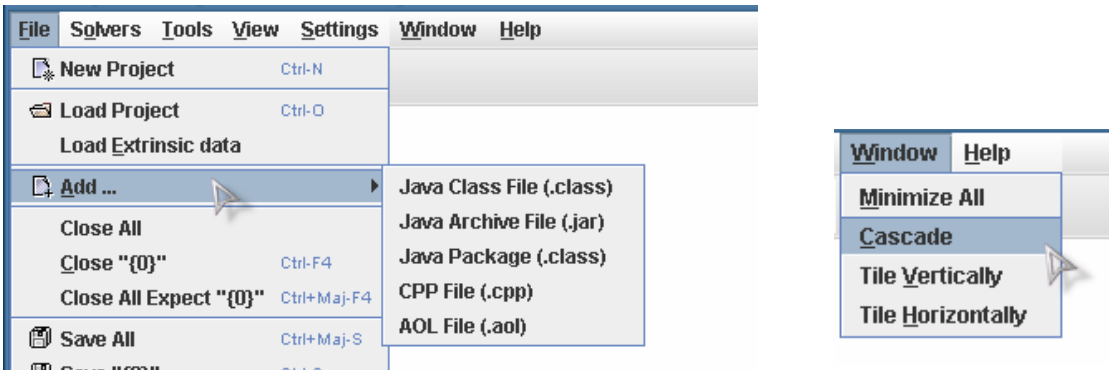
opérationnelle et choisir le fichier de langue en conséquence, et retourner le contenu désiré.

6. Architecture (Exemple):



VI. L'interface graphique (GUI) :

1. Les menus :



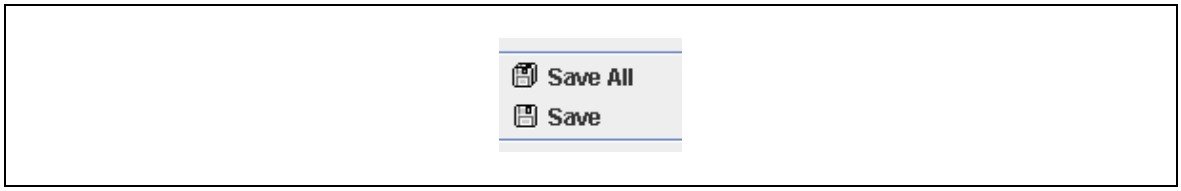
Dans la plupart des applications graphiques, les menus sont une voie principale pour exécuter des commandes. Pour offrir un maximum d'usabilité, les menus doivent être faciles d'accès et surtout, bien organisés.
Offrir un accès par groupe, aide l'utilisateur à retrouver chacune des fonctionnalités qu'il pourra vouloir utiliser sans avoir à trop chercher.

Lors de la conception des menus, il est préférable d'utiliser, le plus possible, une configuration familière à l'utilisateur (quand cela est applicable), tel que : Fichier, Édition, Affichage, Aide, … (Dans cet ordre).
Je précise que l'ordre dans lequel apparaissent les menus, est important. En effet, il est primordial de les placer par ordre d'importance et de fréquence d'utilisation.

a. Raccourcis claviers + mnémoniques :

L'utilisation de raccourcis claviers ainsi que des mnémoniques est aussi recommandable, offrant ainsi à l'utilisateur, un choix encore plus rapide pour accéder aux fonctionnalités (sans avoir à utiliser la souris).
Le choix des raccourcis claviers reste à la discrétion du concepteur, mais, le respect des conventions est préférable, p.e: (Ctrl-O = Open, Ctrl-S = Save, …). De cette façon, l'utilisateur n'aura pas à fournir un effort supplémentaire pour mémoriser ces raccourcis, le cas échéant, ils lui serviront pour d'autres applications.

b. Séparateurs :



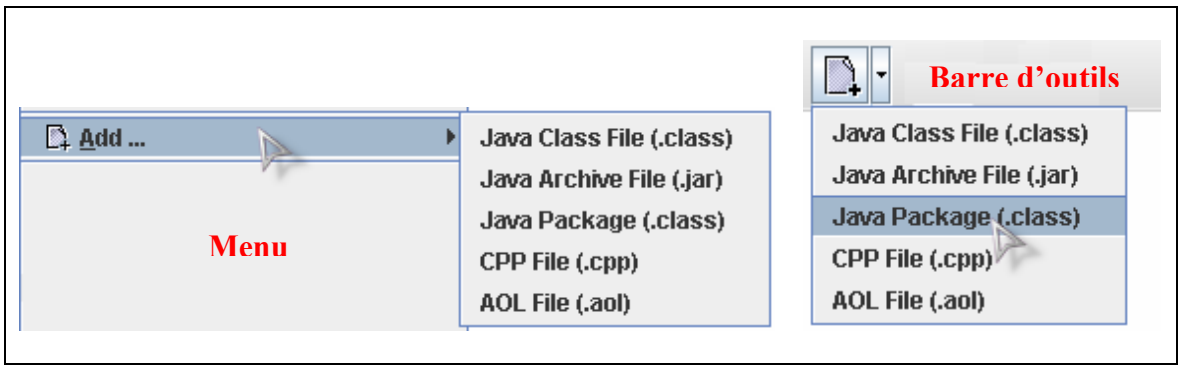
Lors d'une présence d'un sous-groupe de commandes non exhaustif, il est préférable de le délimiter par des séparateurs au lieu de le migrer dans un propre sous-menu.
Ex : Save, Save As, Save All, Save Active …

c. Ellipses (…) :

Les ellipses (…) représentent des points de ponctuation indiquant l'omission d'un ou de plusieurs mots pour que la phrase soit complète.
Ex : Save As …, Find …

L'utilisation de celles-ci, fait savoir à l'utilisateur que la configuration de l'action en question, doit être complétée, et que, lorsque ce dernier choisit cette action, il s'attendra à personnaliser son choix.

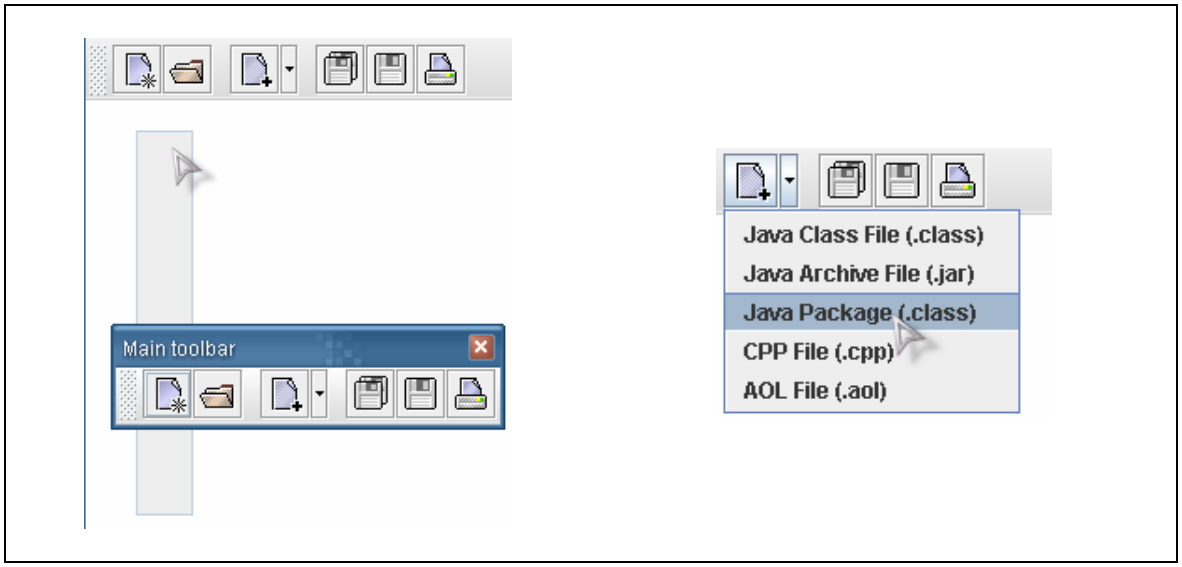
d. Icônes :



L'utilisation des icônes est appréciée, car elle offre un support visuel qui attire l'attention de l'utilisateur lorsque ce dernier cherche une commande en particulier.

Une même commande accessible à partir de plusieurs sources différentes doit avoir la même icône représentative, prévenant ainsi une éventuelle confusion chez l'utilisateur.

2. Les barres d'outils :



Une barre d'outils offre un accès rapide à un ensemble de commandes et options. Les barres d'outils contiennent, typiquement, des boutons, mais aussi, d'autres types de composantes comme des menus, champs de texte, boites à combos, etc.…

Dans le cadre du projet, seulement des boutons est menus (drop down buttons) ont été incorporés. D'autres composantes peuvent être ajoutés pour parfaire le rôle de cette dernière "Accès rapide aux fonctionnalités couramment utilisées".

Attention, un détail d'une grande importance réside dans le fait d'offrir une chaîne d'aide "ToolTip" pour chacune des actions, car ces dernières ne sont représentées que par des icônes. Il est donc primordial d'offrir à l'utilisateur une explication sur la commande écartant ainsi toute confusion dans le cas ou l'icône ne lui évoque pas de contexte.

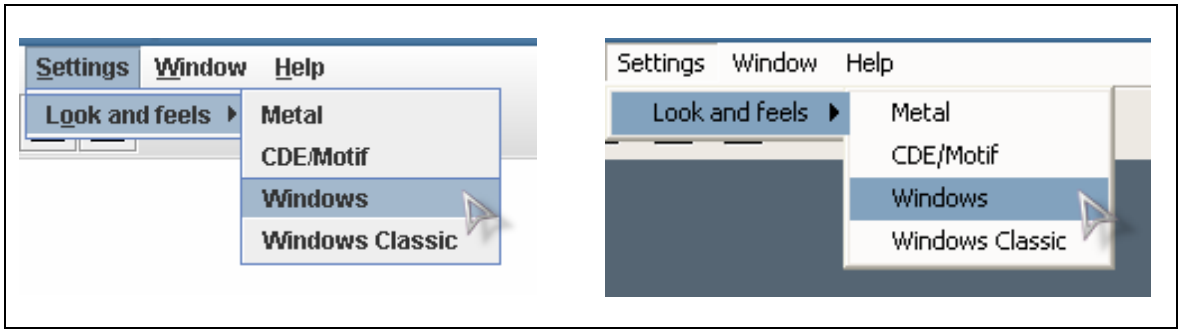
Une option à considérer lors d'une éventuelle mise à jour du modèle, serait d'offrir à l'utilisateur le choix d'afficher le texte correspondant à chaque bouton. Cela ne permettra pas seulement aux utilisateurs novices de se familiariser plus rapidement avec l'interface, mais aussi la possibilité aux usagers avec une vue moins aiguë, d'avoir un texte en plus grand format.

Une autre considération, serait d'offrir à l'utilisateur la possibilité de cacher chacune des barres t'outils (gestion d'espace de travail).

3. Les barres de statut (Status Bar) :

Lors d'une éventuelle mise à jour de l'interface graphique, il serait intéressant d'envisager l'implémentation d'une barre de statut pour afficher les messages de statut ainsi que de l'information concernant les droits d'écriture sur le fichier ptidej en avant plan …

4. Les thèmes (Look and Feels) :



J'ai choisis d'offrir aux utilisateurs de l'application, le choix de modifier l'apparence des composantes graphiques. Effectivement, Java nous offre une portabilité sans équivoque, il me parait donc judicieux, j'irais jusqu'à dire indispensable, d'offrir à chacun des utilisateurs une interface utilisant la même apparence que celle du système d'exploitation sur lequel l'application est exécutée. Mais aussi la possibilité de modifier cette apparence par toutes celles offertes par la machine virtuelle actuellement installée.

Un travail supplémentaire serait d'implémenter une architecture derrière cette fonctionnalité. Une architecture qui s'occupera d'une gestion supplémentaire spécialisée dans la disposition et la nomenclature des composantes, aussi complexes soient-elles.

5. Panneaux spéciaux :

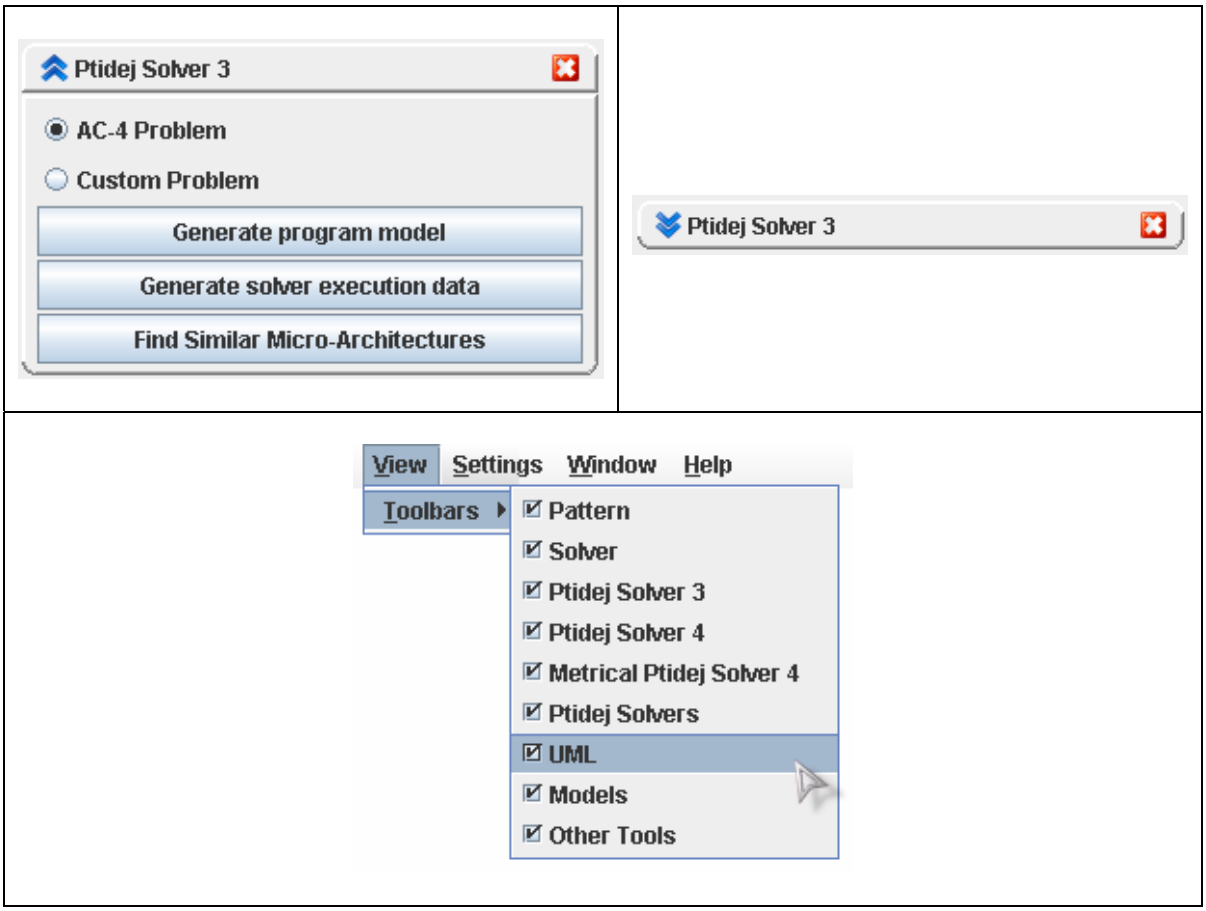
De nos jours, la possibilité de personnalisation de l'espace de travail d'une application est requise pour le confort de l'utilisateur de l'interface.
L'usager aime avoir le choix de disposer ses outils de travail de façon à être à l'aise avec son environnement de travail.

J'ai donc conçu quelques composantes personnalisées aidant ainsi l'usager à choisir plusieurs vues d'affichage augmentant ainsi le degré d'usabilité de l'interface graphique.

a. Collapsible panels :

Les " Collapsible panels ", sont des panneaux pouvant être retroussis d'un click de bouton.
L'ancienne interface graphique (Ptidej UI v1.0) offrait un seul panneau de contrôle de nature statique. L'utilisateur ne pouvait donc pas cacher et afficher des groupes de contrôles selon son choix. Ce qui rendait l'interface peu ergonomique.

Les " Collapsible panels " ont donc étés introduit pour palier à ce problème



Une possibilité d'afficher/cacher les panneaux de contrôle est aussi disponible à partir du menu "Affichage".

Pour créer un " Collapsible panels ", il suffit simplement de faire :

```
JPanel pnl = new CollapsablePanel(strTitle, contentPnl);
```

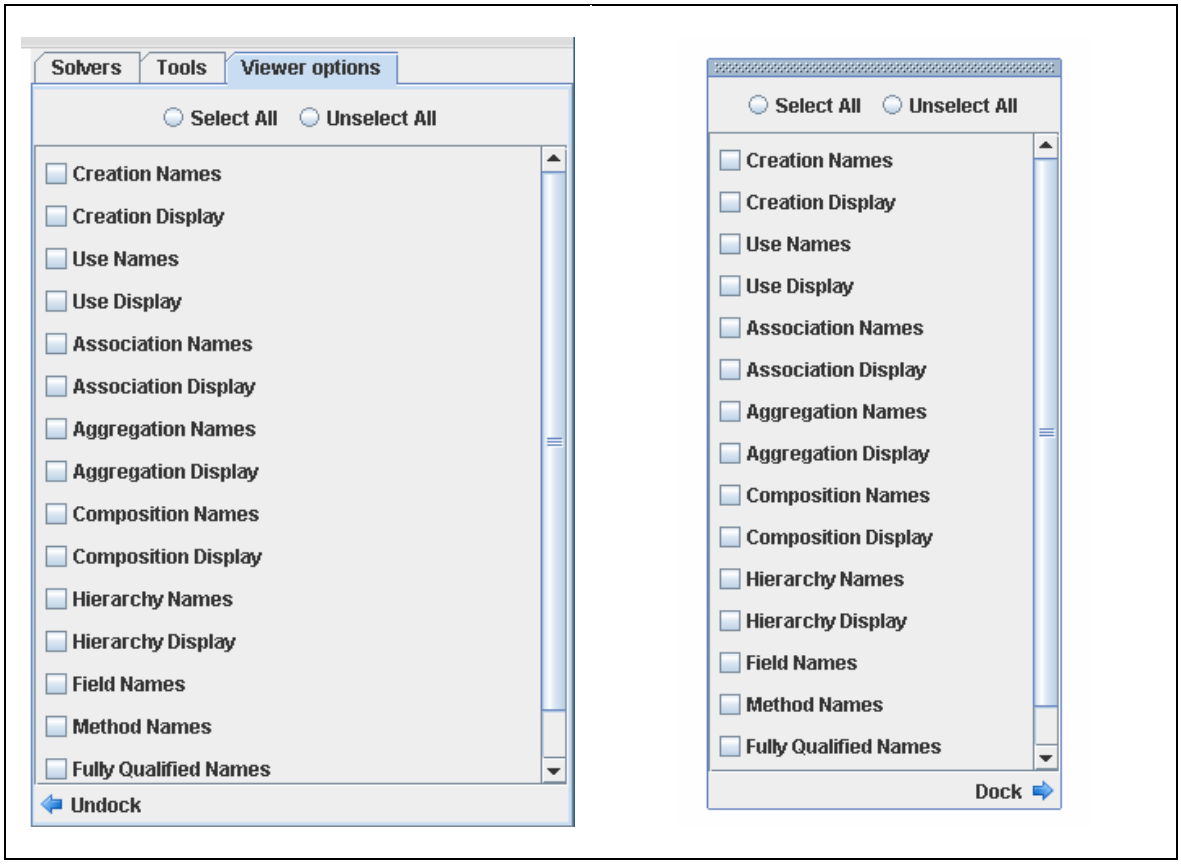
Le panel s'occupe donc de gérer les actions nécessaires pour se retrousser ainsi que pour se fermer. Il s'ajoute aussi au menu "Toolbars" sous le menu "View". Tout reste donc transparent au développeur.

Dans une éventuelle mise à jour, il serait préférable d'implémenter une fonctionnalité tel que l'application puisse garder les préférences de l'utilisateur en ce qui concerne l'état de chacun de ces panneaux.

b. Dock-able panels :

L'ancienne interface reposait sur un mono affichage de projets *.Ptidej. Ce qui limitait l'utilisateur à afficher un seul projet à la fois. Or, il s'est avéré intéressant de pouvoir offrir un affichage multiple, c.à.d, pouvoir afficher plusieurs modèles à la fois, ce qui nous permet donc de pouvoir comparer plusieurs architectures logiciel et bien plus ...

L'idée a introduit le concept de Bureau "Desktop". De ce fait, nous pouvons utiliser des palettes de contrôle d'affichage sur le bureau, et donc, pouvoir avoir accès à plus de commandes à la fois, celles qui sont sur la palette et ceux sur le panneaux de contrôle.

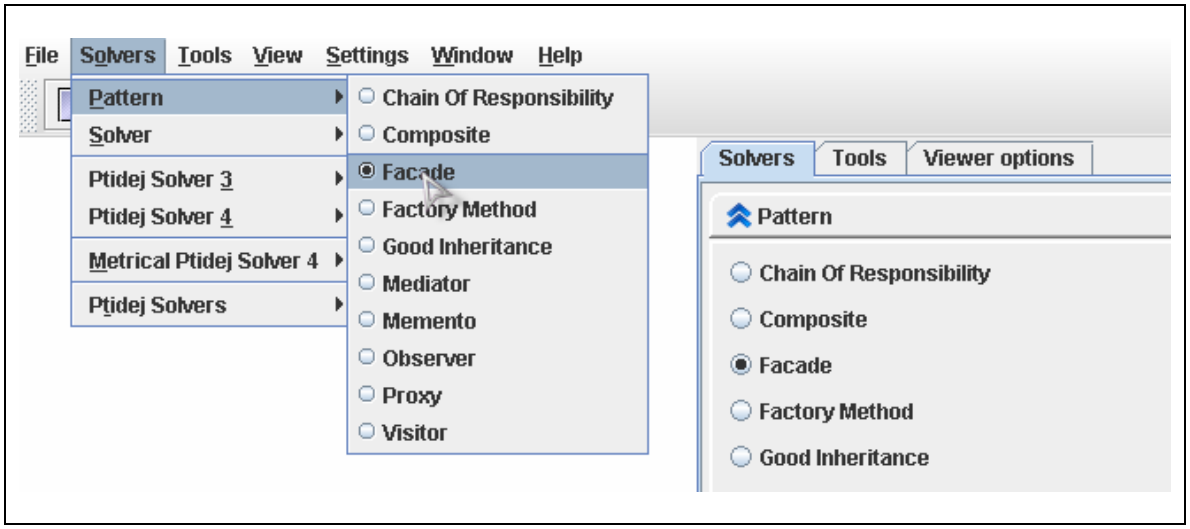


Pour créer un " Undock able panels ", il suffit simplement de faire :

```
JPanel pnl = new UndockablePanel(undockingComp, dockingComp, contentPnl, strTitle);
```

Le panel s'occupe donc de gérer les actions nécessaires pour l'ancrage. Tout reste donc transparent au développeur.

6. Synchronisations des sources d'événements :



Tout événement est associé à une action spécifique. Étant donné la multitude de sources pour un même événement, un problème doit être pris en considération : La synchronisation des sources. En effet lorsqu'un bouton connaît un état quelconque, son égale est aussi concerné, p.e : si un bouton radio est sélectionné, ses clones doivent l'être aussi. Aussi, si une des sources est actionnée, la même action doit être exécutée.

La solution s'est d'étendre une "Abstract Action" qui s'occupe de l'action désirée ainsi que la configuration de toute source du même événement.
En ce qui concerne la synchronisation, il suffit d'appliquer le même modèle à toute source appartenant à la même action.

VII. Conclusion :

Le projet s'est donc composé en 5 points principaux :

- Rendre l'application internationale,
- Étudier le logiciel Ptidej, et son ancienne interface,
- Conception d'une nouvelle interface
- Développer la nouvelle interface
- Effectuer les connexions plus complexes entre Ptidej et sa nouvelle interface.

Le projet a pris plus de temps que prévu, tel que mentionné sur les rapports de progrès disponibles sur le site web. J'ai dû changer l'architecture du système d'interfaçage au fur et a mesure pour délivrer un code extensible et facilement maintenable.
Des patron de conception ont aussi étés utilisés, tel que : MVC (Model View Controller), Singleton, …

L'internationalisation de l'application était aussi un bon défi, car il nous fallait veiller à ce que ce soit le plus découplé possible du reste de l'application.

VIII. Bibliographie :

http://java.sun.com/docs/books/tutorial/i18n/

http://java.sun.com/products/jlf/ed2/book/index.html
http://java.sun.com/products/jlf/at/book/index.html