Université de Montréal

EFREI
ECOLE D'INGENIEURS DES TECHNOLOGIES DE L'INFORMATION ET DU MANAGEMENT

IFT3150 – hiver 2008

Rapport de projet informatique

PEPS : Ptidej for EcliPSe

Etudiant : Nelson Cabral – CABN09078501

Tuteur : Yann-Gaël Guéhéneuc

Table des matières

Présentation du projet

Dans le cadre de mon projet informatique IFT3150, j'ai été amené à travailler sur Eclipse et Ptidej. Voici l'énoncé du projet tel qu'il m'a été présenté.

Responsable : Yann-Gael Gueheneuc, local 2345, poste 6782, guehene@iro.umontreal.ca

Bénefices attendus

En choisissant ce projet, les étudiants apprendront à développer des extensions pour la plate-forme de développement Eclipse. Ils apprendront aussi à manipuler des arbres de syntaxes abstraites et obtiendront des notions de méta-modélisation et de rétro-conception. Ces connaissances apporteront aux étudiants un avantage indéniable sur le marché du travail ou dans la poursuite de leurs études : Eclipse est devenue la plate-forme de développement de référence dans l'industrie (IBM WebSphere) et dans le milieu universitaire. De nombreux projets industriels utilisent Eclipse comme base et une expérience de développement sera un atout indéniable.

Du point de vue des responsables du projet, le travail réalisé permettra une meilleure interaction entre la plate-forme Eclipse et la suite d'outils de rétro-conception développées au DIRO, Ptidej.

Contexte

La plate-forme de développement Eclipse est devenue en quelques années la plate-forme de référence dans l'industrie comme dans le milieu universitaire. Elle offre un environnement cohérent pour intégrer des extensions et faciliter leur accès et leur utilisation par les développeurs.

La suite d'outil de rétro-conception Ptidej est un ensemble d'outils et de modèles avancés permettant l'analyse et la manipulation du code source de systèmes logiciels et d'autres sources de données pour en évaluer la qualité et en faciliter la compréhension.

A l'heure actuelle, il n'existe pas de pont entre Eclipse et Ptidej alors que les objectifs du projet Ptidej rentrent parfaitement dans les objectifs de la plate-forme Eclipse.

Travail demandé

Le travail consiste d'abord à se familiariser avec Eclipse puis comprendre les règles d'implantation d'extension pour Eclipse 2 et 3. Il s'agit ensuite d'implanter une interface entre Eclipse et Ptidej permettant d'appeler depuis Eclipse les fonctionnalités de Ptidej puis de répercuter dans Eclipse le résultats des analyses.

Environnement de travail

Le travail sera effectue par un groupe d'au plus quatre étudiants motivés et bons programmeurs, en Java, avec l'environnement de développement pour Java fourni avec la plate-forme Eclipse.

I Présentation de PTIDEJ

Cette présentation est issue d'une description de Ptidej par son responsable : Yann-Gaël Guéhéneuc, chercheur et enseignant à l'Université de Montréal au sein de la Direction Informatique et Recherche Opérationnelle.

Problème : fournir une architecture pour modéliser des programmes et pour appliquer des analyses, des algorithmes de conversion de données et des programmes tiers sur ces modèles.

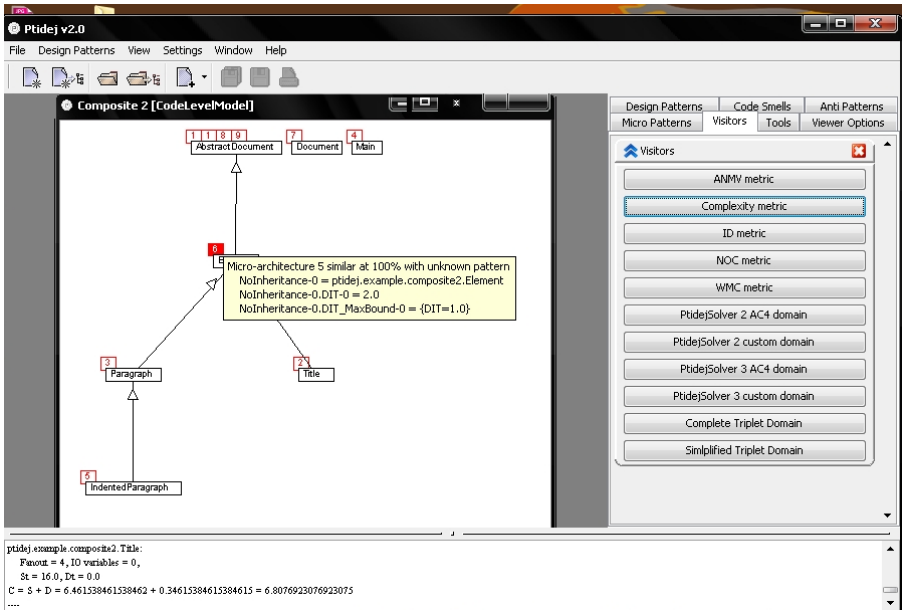
Solution : le projet Ptidej offre une suite d'outils extensibles pour reconstruire le modèle d'un programme et pour l'analyser à l'aide de plusieurs interfaces utilisateur. Il comporte 297+ packages, 2075+ classes et 340+ interfaces pour plus de 112 300 lignes de code Java. Il est activement développé et maintenu depuis 2001, suivant des conventions de codage stricts et des standards de qualité.

Conséquences : la suite Ptidej est activement utilisée à travers le monde pour effectuer des analyses de programmes informatiques et pour créer divers outils.

Parmi les projets qui constituent la suite Ptidej, on compte les projets ou les ensembles de projets :

Caffeine	Dynamic trace generator and analyser for Java using Prolog
DRAM	Dynamic Relationships with Adjacency Matrices
OADynPPaC	Outils pour l'Analyse Dynamique et la mise au Point de Programmes de Programmes avec Contraintes
CPL	Common Ptidej Library
PADL	Pattern and Abstract-Level Description Language
EPI	Efficient Pattern Identification
POM	Primitives, Operations, Metrics
JChoco	Java library to build explanation-based constraints solvers
Ptidej Solvers	Explanation-based constraint solvers to identify (complete and approximate) occurrences of design patterns in PADL models)
SAD	Software Architectural Defects
Ptidej	Pattern Trace Identification, Detection and Enhancement in Java
Ptidej UI	Ptidej graphical interfaces project set
Programmes tiers	DOT, InfoVis, SugiBib
P-MART	Pattern-like Micro-Architecture Repository

Parmi les interfaces utilisateurs complètement fonctionnelles, on peut compter sur Ptidej Viewer Swing Application, dont voici une capture d'écran :



Copie d'écran de l'interface Swing de Ptidej

Comme indiqué dans la description de Ptidej : il est possible de charger un projet Ptidej, afficher sa modélisation et lancer des analyses dessus.

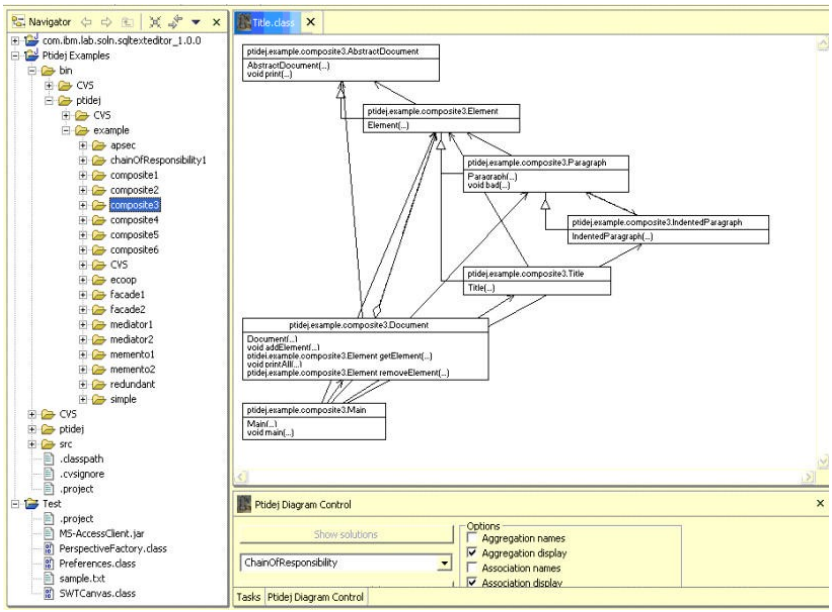
II Précédents

En 2004, deux autres étudiants ont effectué ce projet, Driton Salihu et Lulzim Laloshi.

Malheureusement le plugin qu'ils ont créé n'était pas assez fonctionnel, d'après Yann-Gaël Guéhéneuc : la seule fonctionnalité implémentée est l'affichage du diagramme de classes.

Le plugin n'était pas capable de faire appel à un solver ni faire appel à une autre outil de la suite, alors que ces fonctionnalités sont l'essence et l'intérêt même du projet Ptidej.

Pour plus de détails, leur rapport et leur présentation sont joints à ce rapport en annexe.



Les difficultés qu'ils ont rencontré étaient les suivantes :

- la difficulté de planifier les tâches à accomplir
- l'établissement d'un échéancier réaliste
- la recherche des documents pertinents sur les modules d'extensions
- le chargement dynamique des bibliothèques .jar
- le manque de temps
- le manque d'expérience dans le développement de plugin pour Eclipse et dans le fonctionnement de Ptidej.

Cependant leur travail n'est pas à négliger. En effet, comme précisé dans les parties suivantes, leur travail à grandement accéléré le miens car il m'a servi de documentation de départ.

III Fonctionnalités de la solution : PEPS, Ptidej for EcliPSe

Avant de présenter l'architecture du plugin, un tour des fonctionnalités déjà implémentées s'impose. Les fonctionnalités assurées par le plugin sont réparties entre deux projets, « Peps » et « Peps Extension Starter Pack ».

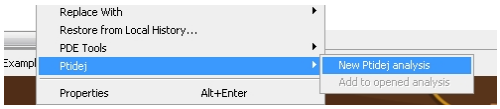
Peps

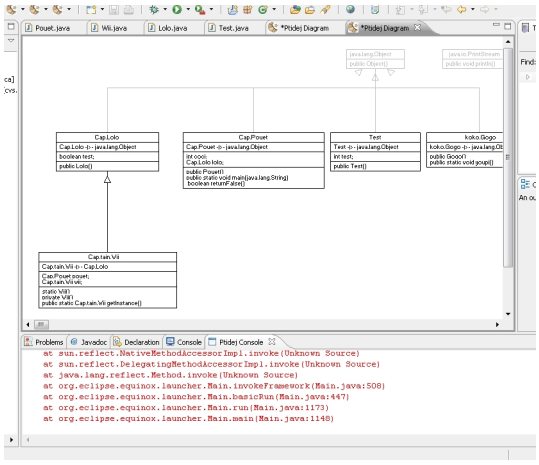
Ce projet est le cœur du plugin. Il assure la gestion des analyses, de leur modèle PADL et de leur canvas. Lui sont donc affectées la création d'une analyse (projet Ptidej) et la gestion de son cycle de vie, dont le chargement et la sauvegarde. Il n'est pas responsable de l'implémentation des outils de la suite Ptidej, mais de la gestion de leur cycle de vie et de leur mise en relation avec le modèle. Pour cela il gère un mécanisme d'extension qui encadre la création d'outils (couche d'abstraction).

Créer un nouveau projet Ptidej

La création d'un projet Ptidej (nouvelle analyse) se fait via la vue Package Explorer d'Eclipse. Sélectionnez un ou plusieurs éléments du navigateur, projet, package, classe, ou quelconque combinaison de ces trois éléments, faites un clic droit dessus (menu contextuel). Un nouveau menu apparaîtra, « Ptidej », et vous pourrez sélectionner « New Ptidej Analysis ». Un nouvel éditeur s'ouvrira avec un superbe schéma représentant votre projet et vos classes. Un bouton dans la toolbar effectue la même fonctionnalité : faites votre sélection comme précédemment et cliquez sur le bouton « New Ptidej Analysis ». Il y a un bug sur le bouton, il faut que la vue sélectionnée soit le Package Explorer pour que cela marche.

Résumé : clic droit sur un élément du Package Explorer > Ptidej > New Ptidej Analysis.

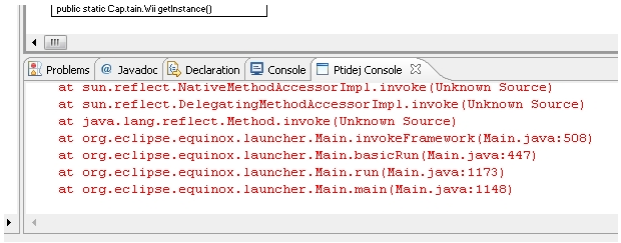




Deux vues : Editeur et Console

Vous constaterez qu'il existe deux vues :

- Une vue Editeur qui s'ouvre à chaque fois qu'un projet est chargé et qui l'affiche, c'est la vue centrale du plugin, visible dans la capture précédente.
- Une vue Console, dans laquelle s'affichent les messages en provenance du plugin. Cette vue, si elle n'est pas affichée par défaut, peut être ouverte en allant dans Window > Show View > Ptidej > Ptidej Console.
- Les vues Ptidej Explorer et Ptidej Tools ne sont pas encore implémentés.



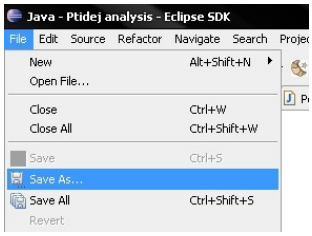
Ajouter des classes à un projet déjà ouvert

Il est possible, une fois une analyse créée, d'ajouter des classes ou des projets à un projet Ptidej déjà ouvert. Pour cela, sélectionnez un élément du Package Explorer que vous voulez rajouter à votre analyse, sélectionnez « Add to opened analysis ». Un bouton dans la toolbar effectue la même fonctionnalité : faites votre sélection comme précédemment et cliquez sur le bouton déroulant « New Ptidej Analysis » pour faire apparaître un menu, vous y verrez « Add to opened analysis ». Il y a un bug sur le bouton, il faut que la vue sélectionnée soit le Package Explorer pour que cela marche.

Résumé : clic droit sur un élément du Package Explorer > Ptidej > Add to opened analysis.

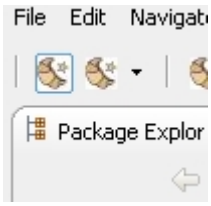
Sauvegarder un projet

Une sauvegarde très basique a été implémentée. Une fois qu'un projet est ouvert, sélectionnez File>Save As, choisissez le chemin de votre fichier et lancez la sauvegarde.



Charger un projet

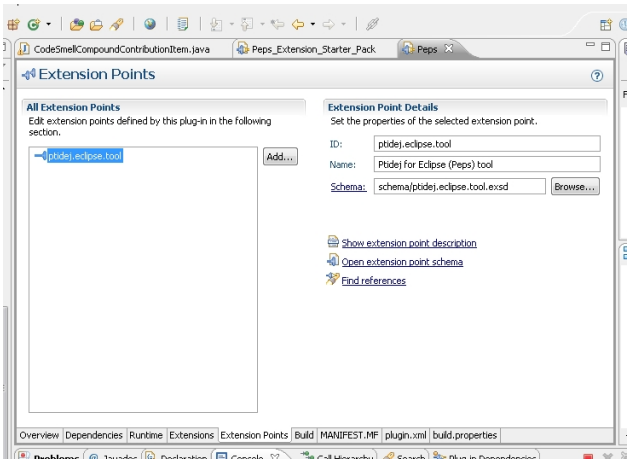
Vous voulez charger un projet Ptidej? Cliquez sur le bouton « Load Ptidej Analysis » dans la toolbar, sélectionnez votre fichier et validez. Attention le chargement est très basique, aucun test n'est fait sur le fichier donné en argument!



Mécanisme d'extension

Comme il peut exister un nombre indéterminé d'outils Ptidej, dont le développement peut être échelonné dans le temps et dont l'existence n'est pas bloquante à l'évolution du coeur du plugin, un mécanisme d'extension a été mis en place. Celui-ci, basé sur le système d'extension d'Eclipse, fournit une mini-API pour créer un outil, l'enregistrer de manière transparente au sein du plugin et accéder au modèle et au canvas PADL.

Le projet Peps fournit donc un point d'extension « ptidej.eclipse.tool » comme indiqué dans la capture ci-dessous. Voir « Etendre le plugin » pour plus d'informations.

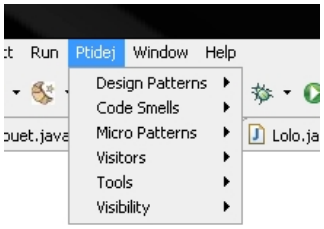


Peps Extension Starter Pack

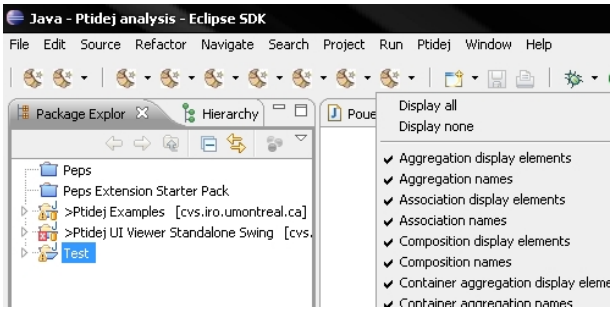
Comme indiqué ci-dessus, le projet « Peps » n'implémente aucun outil d'analyse mais il fournit un point d'extension qui sert à en créer. Un premier ensemble d'outils a été implémenté en utilisant cette interface. Sont déjà disponibles six catégories d'outils Ptidej, implémentées dans le projet « Peps Extension Starter Pack ».

Ces fonctionnalités sont activées uniquement quand l'éditeur ouvert est un éditeur Ptidej (PtidejEditorPart). Une fois cette condition remplie, elles sont accessibles de trois manières :

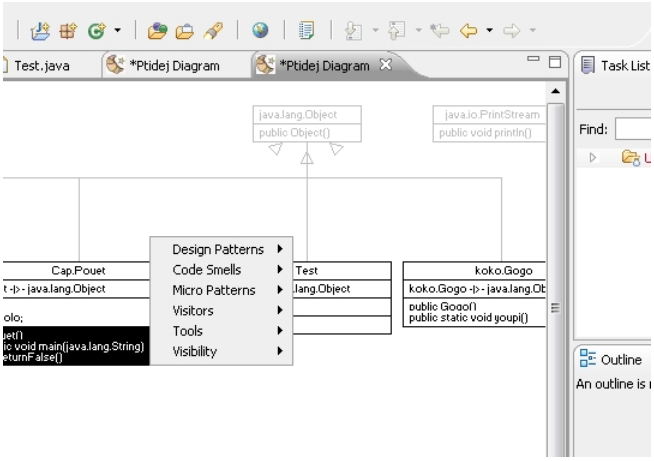
- Via un menu « Ptidej » qui apparaît en haut (méthode par défaut).



- Via une toolbar Ptidej.

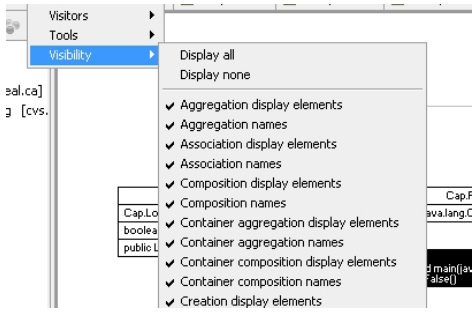


- Via le menu contextuel.



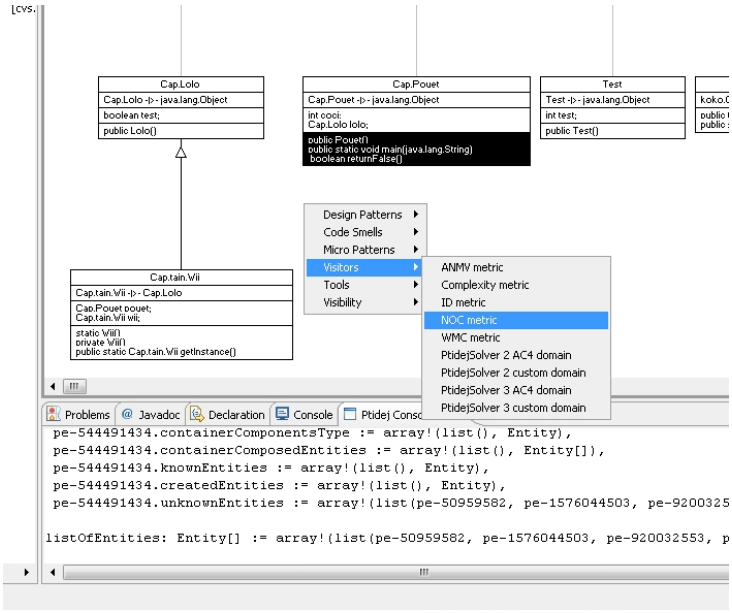
Gestion des éléments visibles (Visibility)

Une fois votre projet Ptidej ouvert et que vos diagramme est affiché, il est possible de masquer ou d'afficher certains éléments. Vous trouverez dans le menu Ptidej > Visibility toutes les options d'affichages existantes.



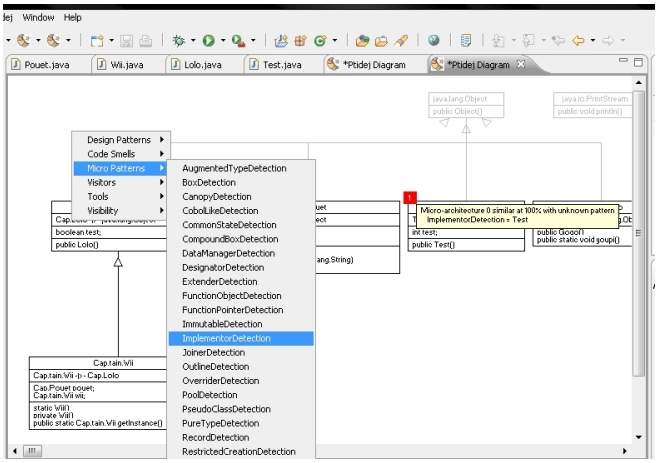
Visiteurs (Visitors)

Le menu Ptidej > Visitors donne accès à une catégorie d'analyses dont les résultats s'affichent généralement dans la vue console. La capture ci-dessous liste les fonctionnalités disponibles ainsi qu'un exemple de résultat.



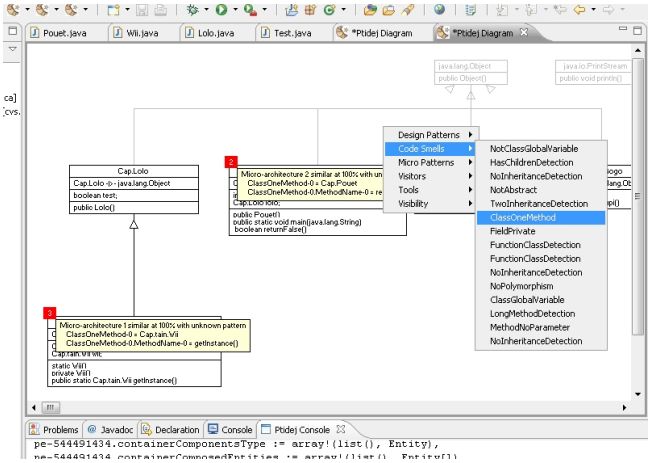
Micro Patterns

De même la détection de micro-patterns a été mise en place : Ptidej > Micro Patterns. Les résultats s'affichent graphiquement par des petites notes numérotées rouges. Essayez ImplementorDetection, il affiche toujours un résultat.



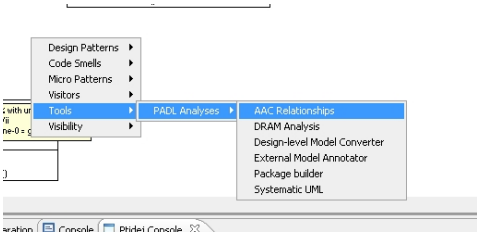
Code Smells

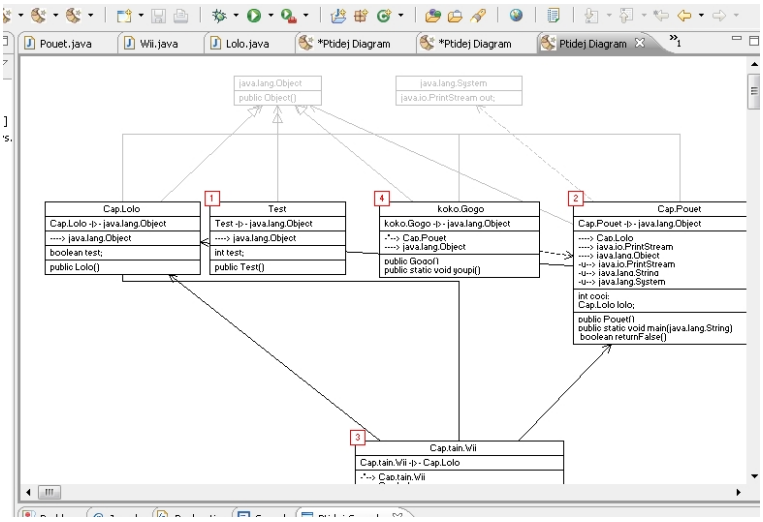
Les code smells sont aussi disponibles sous le menu Ptidej > Code Smells. Les résultats sont affichés graphiquement (carrés rouges).



Outils d'analyse PADL

La suite d'outil Ptidej a été créée pour la détection de designs patterns et la détection de défauts de conception. Une première étape est de construire un modèle comportant les relations de composition et d'agrégation entre classes. Cette étape correspond à la fonction disponible sous Ptidej > Tools > PADL Analyses > AAC Relationship.

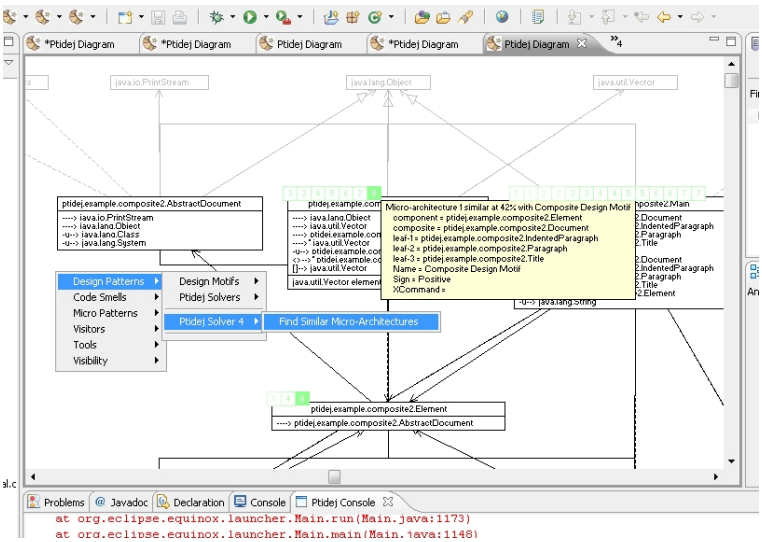




Design Patterns

Une fois la fonction AAC Relationship lancée, vous pouvez lancer la détection de design pattern. Pour cela allez dans la rubrique du même nom : Ptidej > Design Patterns. Sélectionnez un motif et un Ptidej Solver puis lancez l'analyse. Les résultats s'affichent graphiquement, par des carrés verts situés sur les classes concernées.

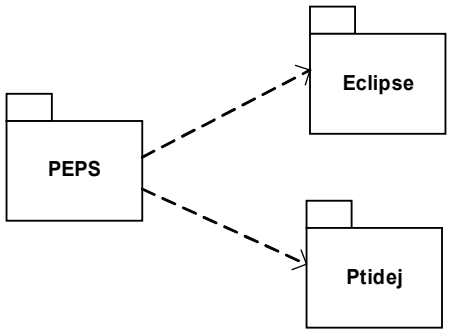
Un bon exemple est de rechercher le design pattern « Composite » dans le projet Ptidej Examples, package ptidej.exemple.composite2.



IV Architecture générale

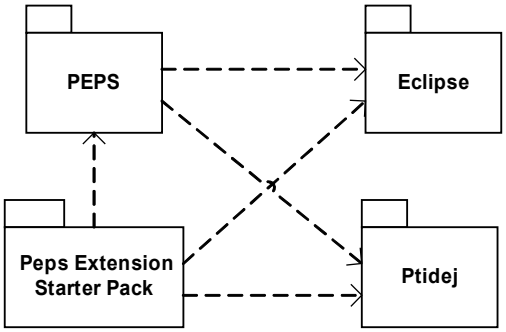
Note : cette partie est une version simplifiée de l'architecture du plugin détaillée en annexe.

Peps est un plugin de Ptidej pour Eclipse, il est donc dépendent techniquement de ces deux ensembles.



Le projet en lui-même est composé de plusieurs sous-projets :

- Peps, le coeur du plugin. Il fait un lien entre Eclipse et Ptidej sur lequel il est possible de rajouter des outils via son point d'extension.
- Peps Extensions, abstraction qui représente tous les outils qui se greffent au plugin. Il est dépendant de Peps qu'il étend, mais aussi d'Eclipse et de Ptidej. Il peut y avoir autant d'extension que l'on souhaite. Un premier ensemble d'extensions a été créé en même temps que le projet Peps, Peps Extension Starter Pack.



Architecture du projet « Peps »

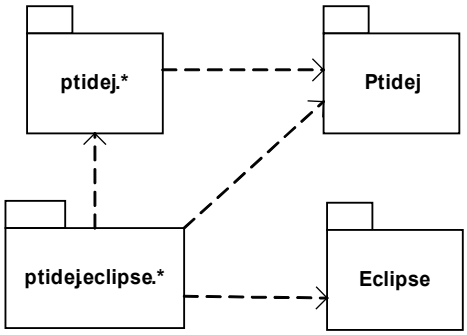
Le projet est décomposable en deux sous-parties :

- Un ensemble de classe indépendantes d'Eclipse. C'est une surcouche de Ptidej. Ces classes ont été créées pour simplifier la gestion de divers aspects de Ptidej, comme la gestion du modèle, la gestion du canvas ou encore la gestion du chargement et de l'enregistrement. Ces fonctionnalités étant indépendantes d'Eclipse, elles ont été logiquement factorisées. Il est même possible de séparer ces classes et les mettre dans un autre projet qui ne serait dépendant que de Ptidej.
- Un ensemble dépendant d'Eclipse. Ce sont ces classes qui intègrent réellement Ptidej dans

Eclipse.

Pour différencier ces classes, les dépendances sont indiquées dans les noms des packages :

- Les packages dont le nom commence par ptidej.eclipse regroupent les classes dépendantes de Ptidej et d'Eclipse. Ex : ptidej.eclipse.ui.part
- Les packages comportant ptidej SANS eclipse indiquent les package indépendants d'Eclipse mais uniquement de Ptidej. Ex : ptidej.manager.canvas



Les packages indépendants d'Eclipse

Un ensemble de classe indépendantes d'Eclipse. C'est une surcouche de Ptidej. Ces classes ont été créées pour simplifier la gestion de divers aspects de Ptidej, comme la gestion du modèle, la gestion du canvas ou encore la gestion du chargement et de l'enregistrement. Ces fonctionnalités étant indépendantes d'Eclipse, elles ont été logiquement factorisées. Il est même possible de séparer ces classes et les mettre dans un autre projet qui ne serait dépendant que de Ptidej.

Les packages concernés sont :

- ptidej.manager :
- ptidej.manager.model
- ptidej.manager.canvas
- ptidej.persistance
- ptidej.io

Les packages dépendants d'Eclipse

Un ensemble dépendant d'Eclipse. Ce sont ces classes qui intègrent réellement Ptidej dans Eclipse.
Les packages concernés sont :

- ptidej.eclipse.ui.part
- ptidej.eclipse.ui
- ptidej.eclipse.tools
- ptidej.eclipse.repository
- ptidej.eclipse.finder
- ptidej.eclipse.actions

Architecture du projet « Peps Extension Starter Pack »

Portage d'un premier ensemble d'outils de la suite Ptidej. Ce projet est dépendant d'Eclipse, de Ptidej et de Peps. Il s'appuie sur le point d'extension ptidej.eclipse.tool de celui-ci.

Voir la partie « Comment créer une extension pour Peps » du rapport pour plus d'informations.

Les packages concernés sont :

- ptidej.eclipse.tools.codesmells
- ptidej.eclipse.tools.designpatterns
- ptidej.eclipse.tools.micropatterns
- ptidej.eclipse.tools.padlanalyses
- ptidej.eclipse.tools.visibility
- ptidej.eclipse.tools.visitors

V Comment créer une extension pour Peps

Cette partie décrit comment créer une extension pour Peps, un nouvel outil, et l'intégrer dans l'interface d'Eclipse, pas à pas.

1. Créez votre projet plug-in

Il faut bien commencer par quelque chose. New > Project ... > Plug-in development > Plug-in project.

2. Établissez la dépendance avec Peps

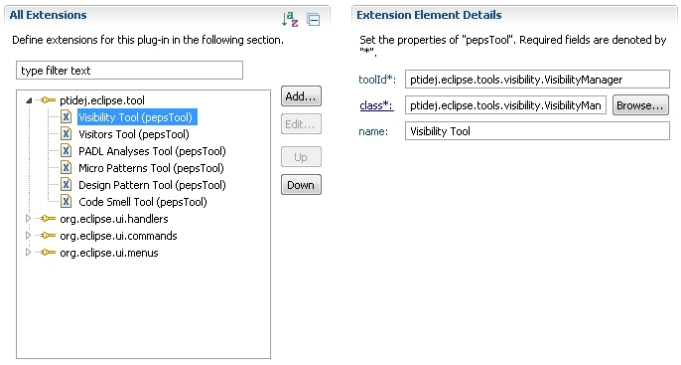
Dans votre projet, cliquez sur plugin.xml. Eclipse vous ouvrira un éditeur contenant des formulaires. Choisissez l'onglet Dependencies en bas de l'éditeur. C'est ici que sont déclarées les dépendances de votre plugin.

Attention : seules les dépendances déclarées dans ce formulaire seront exportées en même temps que votre plugin. Ne mettez donc pas vos dépendances dans le « build path » mais plutôt dans cette fenêtre. De plus si pendant l'exécution vous tombez sur une erreur ClassDefNotFound, c'est souvent parce que la dépendance est déclarée dans le build path et non dans plugin.xml. Ou alors parce que l'un des projets dont vous dépendez n'est pas un plugin, et qu'il n'est donc pas exporté en même temps que votre projet : dans ce cas vérifiez que les projets dont vous êtes dépendant sont aussi des plugin sinon convertissez-les.

3. Sélectionnez le point d'extension

Désormais le point d'extension « ptidej.eclipse.tool » est disponible. Pour le trouver, toujours dans plugin.xml allez dans l'onglet Extensions. Cliquez sur « Add... » et s'afficheront tous les points d'extension auxquels vous avez acces. Sélectionnez ptidej.eclipse.tool.

Pour déclarer votre outil, faites un clic droit sur « ptidej.eclipse.tool » qui devrait s'être ajouté à votre liste de dépendances et choisissez l'option : New > pepsTool. Un nouvel élément apparaît.



Les champs à remplir sont :

- toolId : l'identifiant de votre outil. Veillez à ce qu'il soit unique sinon il y a risque de conflit avec une autre extension. Cet identifiant sert à récupérer une instance de votre implémentation de IPtidejTool.
- class : la classe qui implémente IPtidejTool.
- name : un nom pour votre outil.

Maintenant que vous avez déclaré votre extension, il vous reste à la réaliser.

4. Partie métier : implémentation de IPtidejTool

Cette interface est destinée à accueillir la partie métier de votre extension. Comprenez par là qu'il ne devrait pas y avoir de code propre à Eclipse dans cette implémentation.

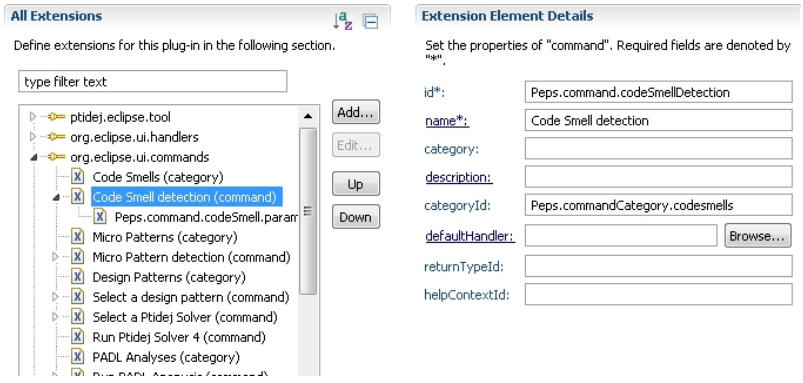
C'est dans cette classe que vous devriez implémenter les actions de votre outil, de sorte que les handlers n'aient qu'à les appeler.

Il faut savoir qu'il n'y a qu'une instance de cette classe par PtidejEditorPart, par couple ModelManager/ CanvasManager. Une fois que votre classe est instanciée, l'éditeur appelle directement setManagers().

5. Déclarez vos actions

Dans Eclipse, les actions sont la définition abstraite de vos boutons. Il peut y avoir plusieurs boutons qui font appel à une action, et plusieurs implémentations d'une action selon le contexte. Par exemple pour l'action « Display All » il y a un bouton dans le menu Ptidej, dans la toolbar et dans le menu contextuel (clic droit sur un editeur Ptidej), mais ces boutons exécutent tous la même action. De plus il est possible que l'action aie plusieurs implémentations (handler), selon le contexte dans lequel elle est appelée (« copier » dans un éditeur de texte n'a pas la même implémentation de « copier » dans un éditeur graphique).

Pour définir votre action, retournez dans la fenêtre Extensions de plugin.xml, faites « Add... » et choisissez le point d'extension « org.eclipse.ui.commands ». Définissez d'abord une catégorie de commande (New > category) puis définissez l'action en tant que telle (New > action).

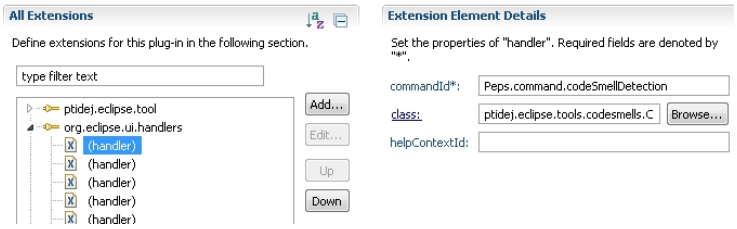


Cf documentation d'Eclipse pour plus de détails sur les actions et Peps Extension Starter Pack pour un exemple.

6. Déclarez vos handler

Dans Eclipse, un Handler est une classe qui implémente une action. Quand vous cliquez sur un bouton, c'est cette classe qui est exécutée.

Cette fois-ci ajouter le point d'extension org.eclipse.ui.handlers et choisissez New > handler. Faites le lien entre l'action que vous avez définit précédemment et le handler en question.



Un handler étend la classe AbstractHandler. Si votre action a un état (coché, décoché), c'est votre Handler qui met à jour l'état en question, pour cela implémentez en plus la classe IElementUpdater.

Pour récupérer une instance de votre IPtidejTool initialisée (avec le ModelManager et le CanvasManager de l'éditeur), faites appel à IPtidejTool PtidejEditorPart.getTool(String), où le String est l'identifiant de votre outil (déclaré au 1. normalement).

Notez aussi que vous êtes aussi responsable du rafraichissement de la fenêtre : effectuez vos opérations puis indiquez à l'éditeur que vous souhaitez qu'il soit rafraichi (dans le cas où vous modifiez son modèle ou son canvas). Voir PtidejEditorPart.refresh().

Pour plus d'information sur les handlers, voyez dans la documentation d'Eclipse, et pour un exemple d'implémentation voyez le projet Peps Extension Starter Pack.

7. Déclarez vos boutons et vos menus

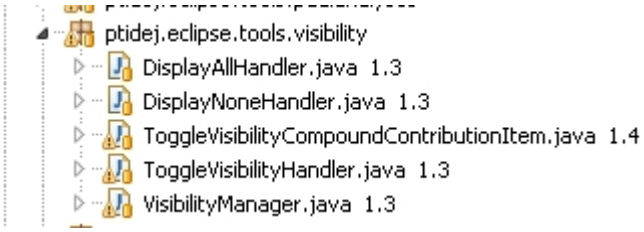
Il ne vous reste plus qu'à déclarer les boutons qui actionneront vos actions (et en cascade les handlers qui sont leur implémentation concrète).

Le point d'extension est org.eclipse.ui.menus. Pour plus d'information sur ce point d'extension voyez dans la documentation d'Eclipse, et pour un exemple d'implémentation voyez le projet Peps Extension Starter Pack.

Peps fournit déjà trois menus où insérer vos boutons :

- menu:Peps.menu.ptidej qui est le menu « Ptidej » qui ne s'affiche que quand l'éditeur ouvert est un PtdejEditorPart.
- popup:peps.ui.parts.PtidejEditorPart?after=additions qui est le menu contextuel de PtidejEditorPart (clic droit sur l'éditeur).
- toolbar:Peps.toolbar.ptidej , toolbar associée à l'éditeur Ptidej, ne s'affiche que quand celui-ci est ouvert.
- toolbar:Peps.toolbar.ptidej.new toolbar associée au plugin Peps, elle est tout le temps visible. Si vous placez votre bouton dedans, sachez qu'il est possible que l'éditeur ouvert ne soit pas de type PtidejEditorPart.

Les boutons peuvent être déclarés de manière statique (plugin.xml) ou de manière dynamique (dynamic, CompoundContributionItem).



A ce stade des boutons devraient s'afficher, cliquer dessus devrait actionner des handlers qui feraient appel à vos IPtidejTool. Félicitations!

VI Difficultés rencontrées

Les difficultés liées à ce projet ont deux causes : la complexité du projet et les contraintes de temps.

Je note la difficulté globale d'un projet ainsi (EFREI, Villejuif, France):

Difficulté globale = complexité (inhérente au projet, incontournable, soit difficulté inhérente + quantité) + complications (non inhérentes à la nature du projet, contournables)

Complexité du projet

La complexité de ce projet est liée à la fois à la complexité d'Eclipse, de Ptidej et de l'environnement dans lequel j'ai été amené à travailler.

Eclipse

Eclipse est un énorme projet et est très complexe. Sa complexité ne réside pas dans la difficulté des éléments à utiliser mais dans leur quantité, chacun des éléments pris à part étant modérément difficile à assimiler.

En regardant les fonctionnalités du projet, on voit directement qu'il fait appel aux notions suivantes d'Eclipse : les fenêtres, les menus, les actions, le mécanisme d'extension d'Eclipse, les éléments dont ces notions sont dépendantes et beaucoup d'autres.

A cela il faut ajouter que je ne connaissais rien à la mécanique d'Eclipse et qu'il n'existait personne dans mon entourage ni celui de mon responsable pour m'y former.

C'est pourquoi pour gagner du temps je suis parti du projet précédent. Même si il ne fonctionnait pas (d'après mon responsable) il m'a été possible de m'en servir comme documentation et point de départ, gagnant ainsi beaucoup de temps.

Il faut aussi noter que Eclipse est très structuré, très documenté et qu'il existe un newsgroup dédié à Eclipse où obtenir du support, ce qui fait aussi gagner pas mal de temps.

Ptidej

En comparaison Ptidej est malheureusement moins documenté, il n'y a pas de Javadoc ou de diagrammes UML par exemple. Par contre il y a beaucoup de commentaires en forme de 'boulettes', ce qui rend le code assez lisible (la phrase précédente mettait en péril mes crédits).

Bien que Ptidej soit assez bien structuré, le manque -relatif- de documentation rend plus difficile son assimilation. De plus Ptidej est, comme Eclipse, pas parfait et à certains endroits j'ai pu constater qu'il était possible de :

- Eclater certaines classes trop complexes et dont le rôle est flou en sous-classes ayant chacune une fonctionnalité différente et plus simple.
- Factoriser du code.

- Simplifier l'architecture et l'organisation des classes (mais je suppose que Ptidej le fait déjà!)

Je pense notamment à AbstractRepresentationWindow et de ses classes filles qui mélange IHM, code métier et controler et sur lesquelles j'ai bien sué.

C'est pourquoi une partie du projet Peps est une surcouche de l'ensemble Ptidej, il s'agit en fait de factorisation de code dans des classes métier.

Java 1.4

Pour terminer la troisième difficulté est liée à l'environnement de travail dans lequel je pouvais travailler. En effet pour des raisons tout à fait compréhensibles les étudiants travaillent sous Eclipse 2 et avec Java 1.4. Ce fut une difficulté supplémentaire, de l'ordre de la complication plutôt que de la complexité, car j'ai toujours travaillé avec Java 6 et Eclipse 3.

Bien qu'il soit vrai qu'en entreprise je sois amené à travailler sur de plus vieilles technologies (par exemple COBOL dans le monde de la finance), ces deux technologies augmentent considérablement la difficulté du projet :

- Passer à Java 1.4 aurait diminué ma productivité et rajouté une période d'adaptation supplémentaire.
- La documentation d'Eclipse 2 est de plus en plus vieille et n'est plus mise à jour, il est aussi plus difficile de trouver du support.
- Le mécanisme de plugin d'Eclipse 2 a changé lors du passage à Eclipse 3, sur lequel vont se baser les prochaines versions d'Eclipse. Développer un plugin pour Eclipse 2 pose des risques à long terme.
- Combien de nouveaux étudiants arrivent en connaissant Eclipse 2? Cela implique que plus le temps passe, moins d'étudiants connaitront cet outil et plus il faudra de temps pour s'y adapter.

Ptidej étant compatible avec Java 1.4 et Java 6, j'ai choisi de développer sous Java 6 et Eclipse 3 afin de profiter un maximum des derniers outils disponible, de documentation à jour et de support.

Si ce choix paye moins à cours terme, il ne peut que s'avérer rentable à moyen et long terme : les nouveaux étudiants seront directement opérationnels pour travailler sur le plugin et trouveront plus de documentation et de support avec le temps.

Finalement d'après mon cours de génie logiciel, un système d'information qui n'évolue pas avec ses utilisateurs est un système d'information destiné à disparaître.

Charge de travail

Ce projet m'a demandé énormément de travail. Pour être honnête il a représenté plus de travail que mes deux autres cours de maîtrise et mon cours d'anglais réunis.

Je pense que la réussite (relative) de ce projet est due à une organisation efficace : durant la session

d'automne j'ai constaté qu'il y a moins de travail en début de session (deux premiers mois) que durant la seconde moitié de la session. Je m'étais donc fixé comme objectif de terminer les ¾ du projet avant la mi-session. J'ai donc travaillé d'arrache-pied durant les deux premiers mois pour avancer un maximum possible, ce qui a été concluant puisqu'à la mi-session j'ai réussi à atteindre l'objectif optimiste que je m'étais fixé, avoir une base du projet qui fonctionne.

A cela il faut ajouter que par manque d'expérience je manquais de visibilité sur le projet, ne pouvant pas faire de planning précis. A ce sujet je dirais qu'un planning serré pour un étudiant qui souhaite travailler sur ce projet serait :

- 2 à 3 semaines pour apprendre le fonctionnement de Ptidej, Eclipse et Peps (1 mois sans cette documentation, jusqu'à 2 mois selon l'étudiant).
- 2 mois pour réaliser ses objectifs.
- 3 semaines pour réaliser la documentation de son travail.

Ce planning est serré, demandant à l'étudiant de beaucoup s'investir les 3 premières semaines.

VII Perspectives, notes et améliorations

A l'avenir, ce plugin devrait assurer toutes les fonctionnalités assurées par l'application Swing.

Une partie de ces fonctionnalités sont les outils de la suite Ptidej, dans ce cas elles ne devraient pas toucher au projet Peps, tandis qu'une autre concerne la base même du plugin.

Au niveau de Ptidej :

- transformer les dépendances en plugins et non en simples projets
- débugger ou améliorer certaines classes, comme SWTCanvas

Améliorations du projet Peps :

- ajout de vues
- support de nouveaux formats
- amélioration de l'intégration au sein d'Eclipse
- ...

Améliorations au niveau des extensions :

- implémenter l'appel de tous les solvers
- implémenter les fonctionnalités manquantes de Ptidej

Améliorations globales :

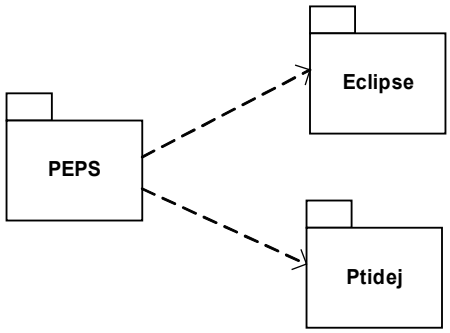
- définition de normes de qualité
- création de documentation exhaustive (Javadoc?)

Pour plus de détails sur les problèmes existants et les améliorations possibles sont énumérées dans la partie « notes et bugs divers » en annexe.

Annexes

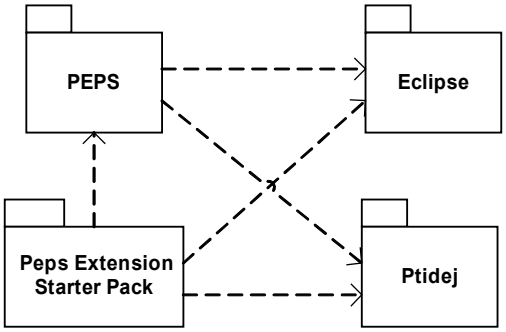
Architecture générale

Peps est un plugin de Ptidej pour Eclipse, il est donc dépendent techniquement de ces deux ensembles.



Le projet en lui-même est composé de plusieurs sous-projets :

- Peps, le coeur du plugin. Il fait un lien entre Eclipse et Ptidej sur lequel il est possible de rajouter des outils via son point d'extension.
- Peps Extensions, abstraction qui représente tous les outils qui se greffent au plugin. Il est dépendant de Peps qu'il étend, mais aussi d'Eclipse et de Ptidej. Il peut y avoir autant d'extension que l'on souhaite. Un premier ensemble d'extensions a été créé en même temps que le projet Peps, Peps Extension Starter Pack.



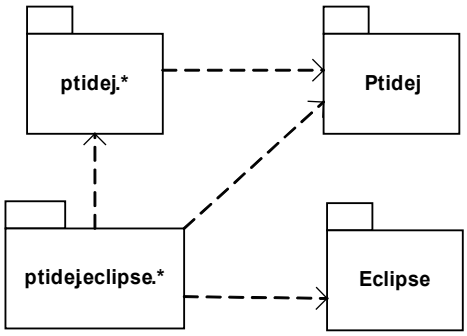
Architecture du projet « Peps »

Le projet est décomposable en deux sous-parties :

- Un ensemble de classe indépendantes d'Eclipse. C'est une surcouche de Ptidej. Ces classes ont été créées pour simplifier la gestion de divers aspects de Ptidej, comme la gestion du modèle, la gestion du canvas ou encore la gestion du chargement et de l'enregistrement. Ces fonctionnalités étant indépendantes d'Eclipse, elles ont été logiquement factorisées. Il est même possible de séparer ces classes et les mettre dans un autre projet qui ne serait dépendant que de Ptidej.
- Un ensemble dépendant d'Eclipse. Ce sont ces classes qui intègrent réellement Ptidej dans Eclipse.

Pour différencier ces classes, les dépendances sont indiquées dans les noms des packages :

- Les packages dont le nom commence par ptidej.eclipse regroupent les classes dépendantes de Ptidej et d'Eclipse. Ex : ptidej.eclipse.ui.part
- Les packages comportant ptidej SANS eclipse indiquent les package indépendants d'Eclipse mais uniquement de Ptidej. Ex : ptidej.manager.canvas



Les packages indépendants d'Eclipse

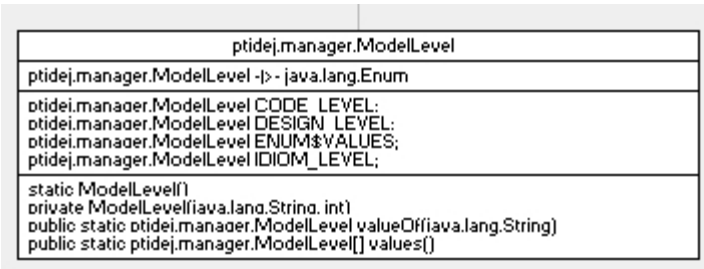
Un ensemble de classe indépendantes d'Eclipse. C'est une surcouche de Ptidej. Ces classes ont été créées pour simplifier la gestion de divers aspects de Ptidej, comme la gestion du modèle, la gestion du canvas ou encore la gestion du chargement et de l'enregistrement. Ces fonctionnalités étant indépendantes d'Eclipse, elles ont été logiquement factorisées. Il est même possible de séparer ces classes et les mettre dans un autre projet qui ne serait dépendant que de Ptidej.

Les packages concernés sont :
- ptidej.manager
- ptidej.manager.model
- ptidej.manager.canvas
- ptidej.persistance
- ptidej.io

ptidej.manager

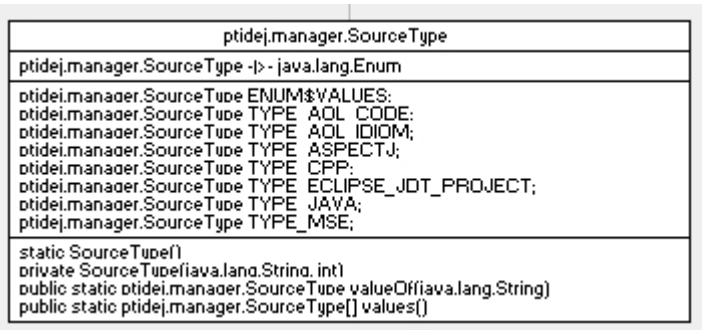
Ce package regroupe les énumérations utilisées dans ptidej.manager.model et ptidej.manager.canvas.

ModelLevel



Les différents niveaux de modèles PADL qui existent. Utilisé notemment par ModelManager pour choisir l'algorithme de build(). Existait déjà mais repris pour faire un switch et rendre le code plus lisible qu'une série de equals().

SourceType

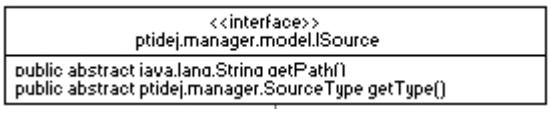


Les différents types de sources supportées par Ptidej. Existait déjà mais repris pour faire un switch et rendre le code plus lisible qu'une série de equals().

ptidej.manager.model

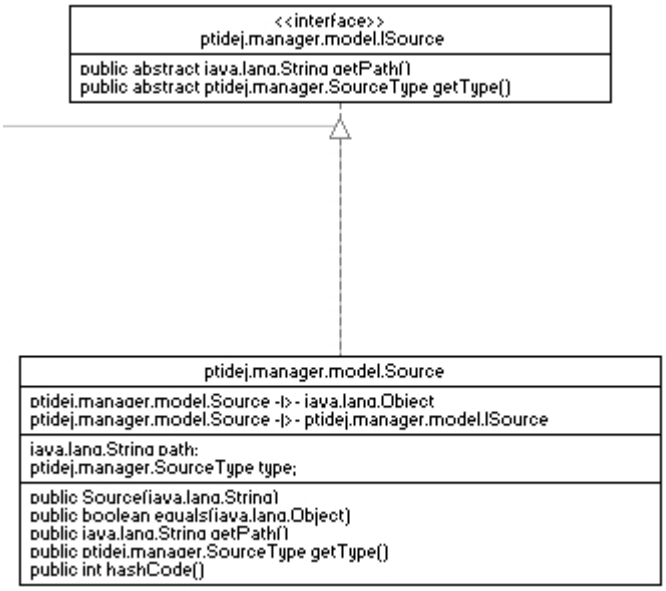
Ce package est une surcouche du modèle PADL. S'y trouve les classes chargées de gérer le cycle de vie d'un IAbstractLevelModel, de ses sources et de ses occurrences. C'est le modèle du plugin (modèle dans le sens modèle du patron de conception MVC).

ISource



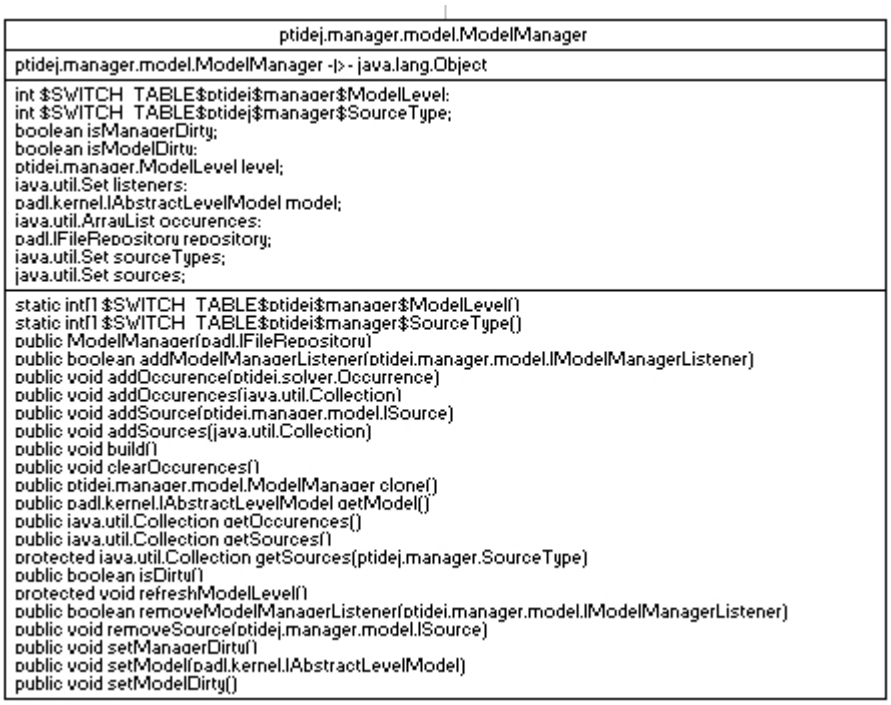
Interface représentant une source. Ses fonctions sont à discuter ainsi que les classes de retour de ses méthodes.

Source



Représente un fichier source. Implémentation basique (et très discutable) de l'interface ISource.

ModelManager



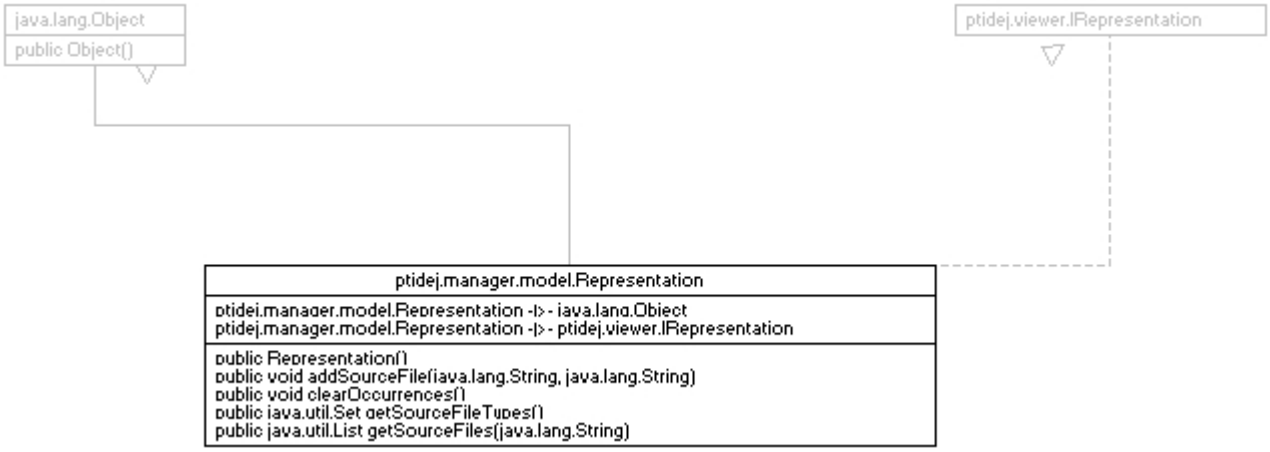
Gère IAbstractLevelModel et son cycle de vie. C'est le modèle (MVC) du plugin, il contient les données métier de l'application.
Pour l'utiliser : lui ajouter des sources, des occurrences, et appeler sa fonction build() qui se chargera de construire (ou reconstruire) le modèle. Dans le cas où on souhaite associer un modèle à des sources, utiliser sa fonction setModel(). En pratique ajouter des occurrences n'entrainera pas la reconstruction

du modèle.
Envoie un signal à ses IModelManagerListener quand il devient sale (dirty). Pour remettre isDirty() à true, il faut reconstruire le modèle (fonction build()). Si vous insérez un model autre qu'un ICodeLevelModel, n'ajoutez pas de source sinon ça le fera planter. La fonction build() nécessite d'être réécrite pour gérer les autres cas que CodeLevel.
A long terme devrait implémenter IRepresentation (mais le rôle de IRepresentation est flou).

IModelManagerListener

Interface pour observer ModelManager. Par exemple le CanvasManager va écouter le ModelManager pour mettre à jour son état.

Representation

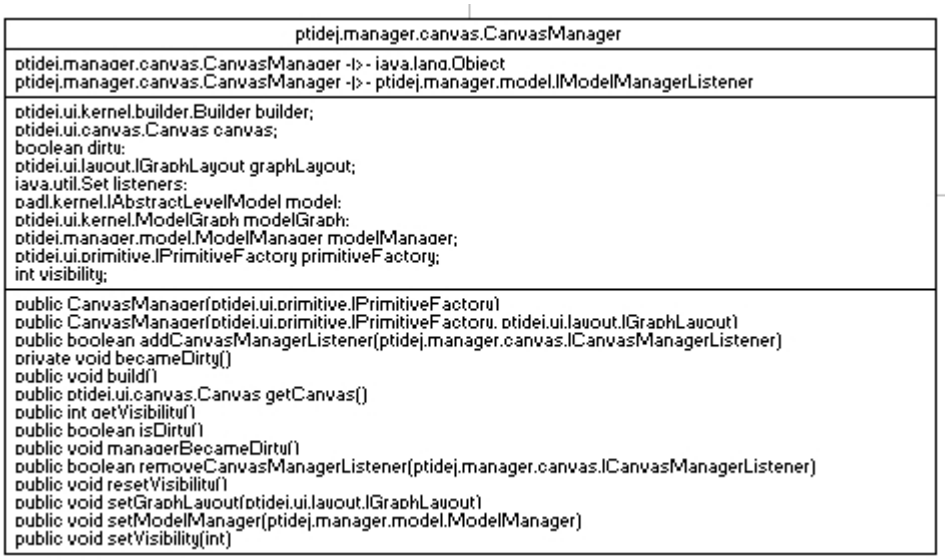


Implémentation vide de IRepresentation, est présente à titre de documentation seulement.
ModelManager remplit le même rôle, à long terme ModelManager devrait implémenter IRepresentation mais actuellement toutes les implémentations de IRepresentation font 5 ou 6 trucs en même temps. Par précaution j'ai construit ModelManager en me basant sur Representation mais jusqu'à ce que ce soit nécessaire et que ce soit plus clair, j'ai préféré ne pas faire l'implement tout de suite.

ptidej.manager.canvas

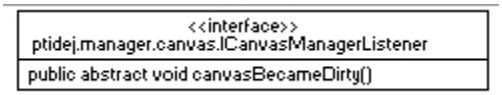
Ce package est une surcouche du Canvas (pas du SWTCanvas). De même que pour ptidej.manager.model, il contient le gestionnaire du Canvas.

CanvasManager



Gère le Canvas et écoute le ModelManager dont il dépend. A chaque build() il recrée un nouveau Canvas. Il est donc nécessaire de faire un getCanvas() après un build() pour obtenir le canvas mis à jour. Utiliser setVisibility(int) pour changer les éléments visibles.
Envoie un signal à ses ICanvasManagerListener quand le Canvas nécessite d'être reconstruit (isDirty()==true => call build()).

IcanvasManagerListener

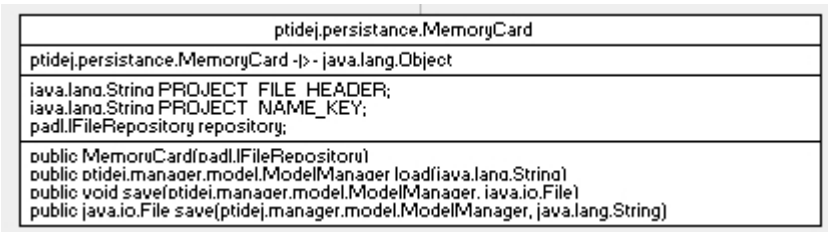


Interface pour observer CanvasManager. Par exemple le PtidejEditorPart va écouter le CanvasManager pour mettre à jour le SWTCanvas (et en conséquence l'écran).

ptidej.persistance

Package en charge du chargement et de l'enregistrement d'un ModelManager (c'est à dire IAbstractLevelModel + Occurrences + tout autre donnée métier).

MemoryCard

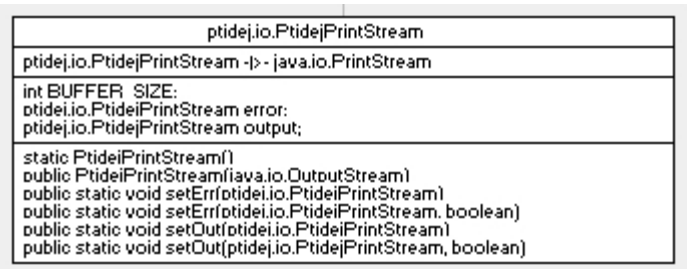


Classe chargée de l'enregistrement et du chargement d'un ModelManager. J'ai pas trouvé de meilleur nom.

ptidej.io

DEPRECATED. Ce package a pour rôle de remplacer java.util.io.

PtidejPrintStream



DEPRECATED. Cette classe a pour rôle de remplacer System.out et System.err, afin d'en faire un exemplaire à portée de Ptidej, mais il existait déjà util.io.Output. Il faudrait la marquer comme deprecated, et changer la redirection des flux dans PtidejConsoleViewPart.

Les packages dépendants d'Eclipse

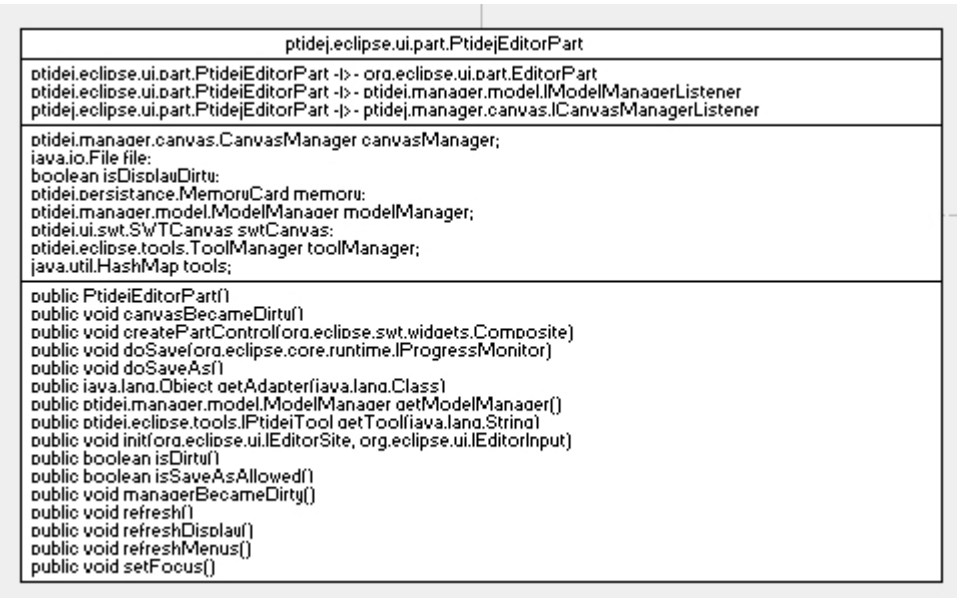
Un ensemble dépendant d'Eclipse. Ce sont ces classes qui intègrent réellement Ptidej dans Eclipse. Les packages concernés sont :

- ptidej.eclipse.ui.part
- ptidej.eclipse.ui
- ptidej.eclipse.tools
- ptidej.eclipse.repository
- ptidej.eclipse.finder
- ptidej.eclipse.actions

ptidej.eclipse.ui.part

Ce package regroupe toute les implémentations de IViewPart, c'est à dire toutes les vues du plugin.

PtidejEditorPart



Vue centrale du plugin. Représente l'éditeur associé à une instance de ModelManager (et donc de IAbstractLevelModel).
Utiliser la fonction IPtidejTool getTool(String) pour récupérer une instance d'un IPtidejTool associé à son ModelManager.

Contrôleur d'une analyse Ptidej (modèle : ModelManager, vue : CanvasManager + SWTCanvas).
Contrôle le cycle de vie du fichier .ptidej associé, de l'instance de CanvasManager, ModelManager, SWTCanvas mais aussi des IPtidejTool associés à son instance de ModelManager - instanciation, établissement et mise à jour des dépendances.

Eclipse associe toujours à une instance d'un éditeur (EditorPart) une instance de EditorInput (cf doc Eclipse). Ici le lien est implémenté par le couple PtidejEditorPart/PtidejEditorInput.

Snippet pour créer et ouvrir une instance de PtidejEditorPart :

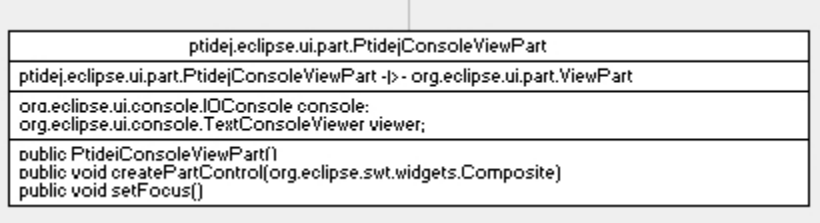
```
ModelManager manager = new ModelManager(JarFileRepository.getCurrentFileRepository());
manager.addSources(sources);

PtidejEditorInput input = new PtidejEditorInput(manager);
IWorkbenchPage page = site.getWorkbenchWindow().getActivePage();
page.openEditor(input, "peps.ui.parts.PtidejEditorPart", true);
```

Voir les classes PtidejEditorInput pour plus d'informations sur l'input et SelectionNewAnalysisHandler ou SelectionAddToOpenedAnalysisHandler pour plus d'informations sur ce snippet.

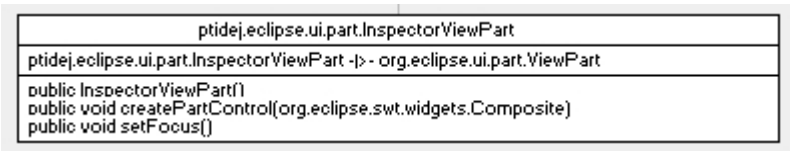
PtidejConsoleViewPart

Vue console dans laquelle le plugin écrit ses messages. En théorie parce que actuellement elle redirige



les flux System.out et System.err. Implémentation à améliorer.

InspectorViewPart



Vue non implémentée. Est sensée afficher le panneau latéral disponible dans le Viewer Swing. Nécessite une remise à plat de l'interface usager avant d'être implémentée.

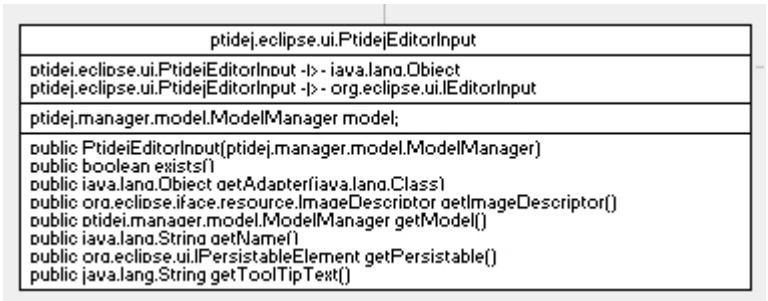
TreeViewPart

Vue non implémentée. Le contenu n'est pas encore déterminé. A l'avenir peut soit implémenter la vue hiérarchique du Viewer Swing, soit réimplémenter une toute nouvelle vue, comme une vue Package Explorer simplifiée (voir suggestions à la fin du rapport).

ptidej.eclipse.ui

Contient les classes dont dépend ptidej.eclipse.ui.part .

PtidejEditorInput



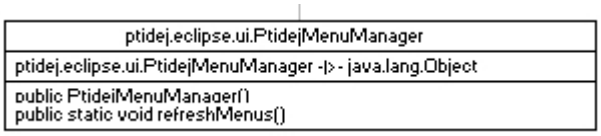
EditorInput associé à un PtidejEditorPart. Prend en argument le ModelManager que le PtidejEditorPart récupérera lors de son instantiation (fonction init() de PtidejEditorPart).

Snippet pour créer et ouvrir une instance de PtidejEditorPart :

```
ModelManager manager = new ModelManager(JarFileRepository.getCurrentFileRepository());
manager.addSources(sources);
```

```
PtidejEditorInput input = new PtidejEditorInput(manager);
IWorkbenchPage page = site.getWorkbenchWindow().getActivePage();
page.openEditor(input, "peps.ui.parts.PtidejEditorPart", true);
```

PtidejMenuManager



Gère le rafraichissement des menus de Peps. Pourrait être une nested class dans PtidejEditorPart (à étudier).

ptidej.eclipse.tools

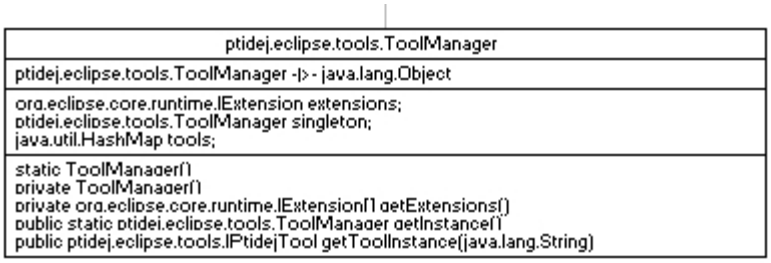
Ce package contient les classes dédiées à la gestion des extensions de Peps.

IptidejTool

Interface que les extensions de Peps doivent implémenter. Lorsqu'une instance de IPtidejTool est créée, sa fonction setManagers() est directement appelée. Il y a une instance max de chaque implémentation de IPtidejTool par ModelManager.

Voir IPtidejTool PtidejEditorPart.getTool(String) pour plus d'information sur son utilisation et le point d'extension ptidej.eclipse.tool.

ToolManager

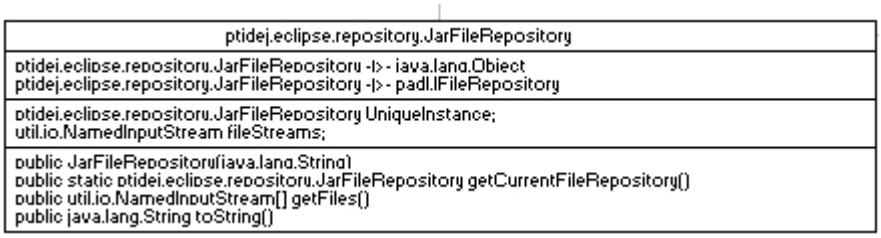


Gère les extensions de Peps qui implémentent le point d'extension ptidej.eclipse.tool. Est utilisé par PtidejEditorPart pour récupérer une instance d'un IPtidejTool, voir IPtidejTool getToolInstance(String) pour plus d'informations.

ptidej.eclipse.repository

Contient les IFileRepository propres à Eclipse.

JarFileRepository



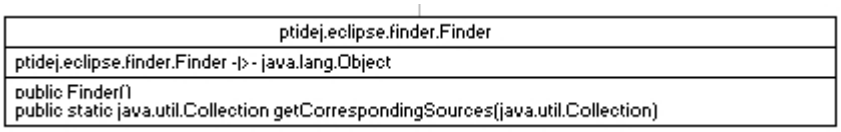
Implémentation de IFileRepository pour Eclipse.

Pour instancier un JarFileRepository :
public JarFileRepository(String bundleName)
A discuter : utilisation d'un singleton pour chaque bundleName.

ptidej.eclipse.finder

Contient les outils chargés de récupérer les .class dans Eclipse.

Finder

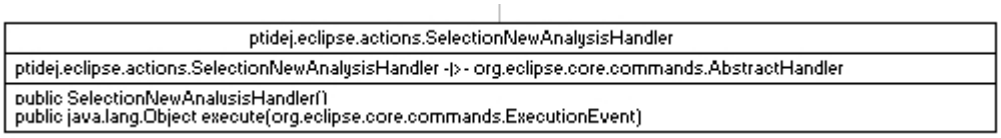


Utilitaire pour retrouver les .class générés par Eclipse. Voir `getCorrespondingSources()` pour plus d'informations.

ptidej.eclipse.actions

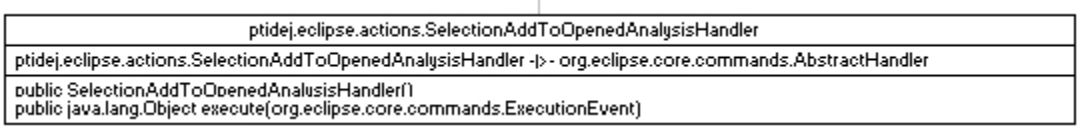
Regroupe les actions implémentées par Peps.

SelectionNewAnalysisHandler



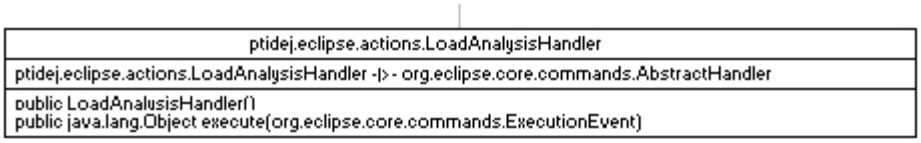
Handler qui implémente l'action « New Ptidej Analysis ». Voir plugin.xml pour plus de détails sur la commande et les boutons associés (onglet Extensions).

SelectionAddToOpenedAnalysisHandler



Handler qui implémente l'action « Add to opened analysis ». Voir plugin.xml pour plus de détails sur la commande et les boutons associés (onglet Extensions).

LoadAnalysisHandler



Handler qui implémente l'action « Load Analysis ». Voir plugin.xml pour plus de détails sur la commande et les boutons associés (onglet Extensions).

Architecture du projet « Peps Extension Starter Pack »

Portage d'un premier ensemble d'outils de la suite Ptidej. Ce projet est dépendant d'Eclipse, de Ptidej et de Peps. Il s'appuie sur le point d'extension ptidej.eclipse.tool de celui-ci.

Voir la partie « Comment créer une extension pour Peps » du rapport pour plus d'informations.

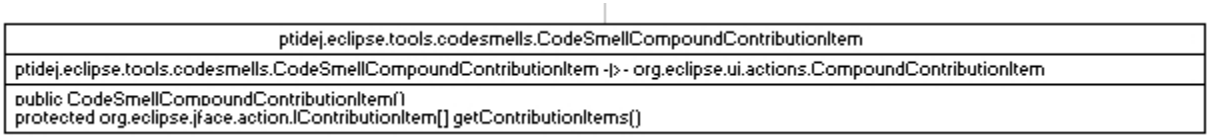
Les packages concernés sont :

- ptidej.eclipse.tools.codesmells
- ptidej.eclipse.tools.designpatterns
- ptidej.eclipse.tools.micropatterns
- ptidej.eclipse.tools.padlanalyses
- ptidej.eclipse.tools.visibility
- ptidej.eclipse.tools.visitors

ptidej.eclipse.tools.codesmells

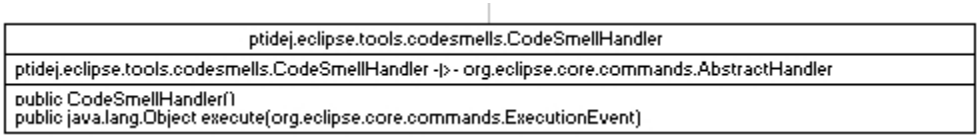
Ce package regroupe les classes implémentant les fonctionnalités de la catégorie Code Smells.

CodeSmellCompoundContributionItem



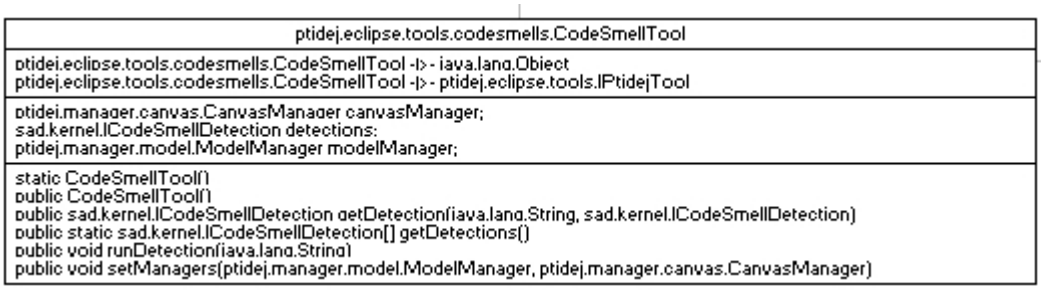
Classe qui génère la liste des code smells. Voir plugin.xml, onglet Extensions pour plus d'information.

CodeSmellHandler



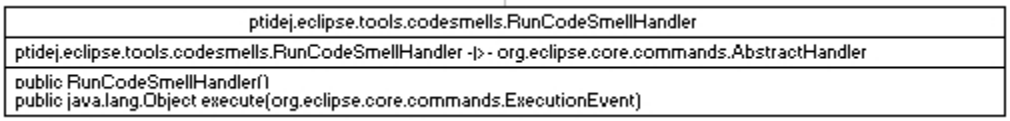
Handler qui gère l'exécution des commandes de la catégorie code smell. S'appuie sur CodeSmellTool pour le code métier (indépendant d'Eclipse). Voir plugin.xml, onglet Extensions pour plus d'information.

CodeSmellTool



Implémentation de IPtidejTool. Classe métier de la fonctionnalité Code Smells.

RunCodeSmellHandler

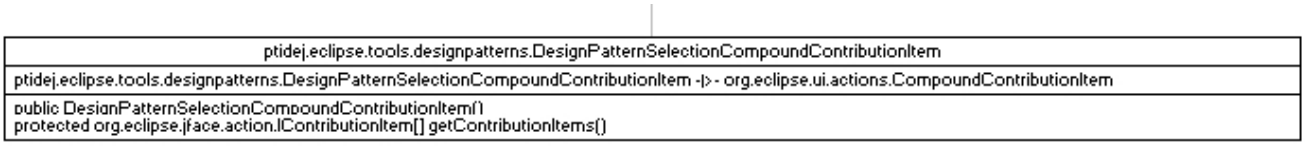


DEPRECATED. Ancien handler du groupe, remplacé par CodeSmellHandler. Sert de documentation sur les handlers.

ptidej.eclipse.tools.designpatterns

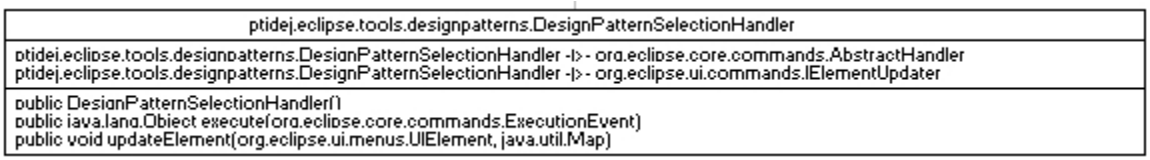
Ce package regroupe les classes implémentant les fonctionnalités de la catégorie Design Patterns.

DesignPatternSelectionCompoundContributionItem



Classe qui génère la liste des design patterns. Voir plugin.xml, onglet Extensions pour plus d'information.

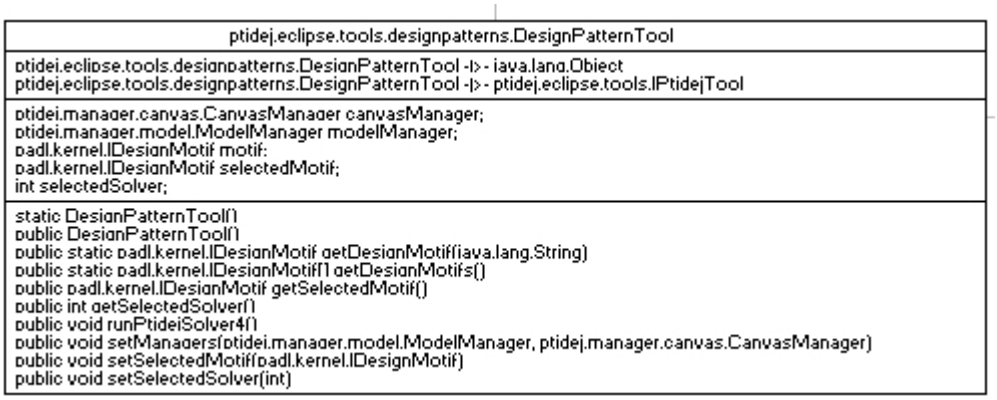
DesignPatternSelectionHandler



Handler qui gère l'exécution la sélection du motif (design pattern) que l'on souhaite analyser. S'appuie sur DesignPatternTool pour le code métier (indépendant d'Eclipse). Voir plugin.xml, onglet Extensions pour plus d'information.

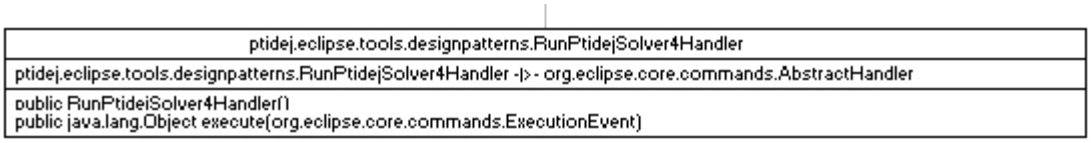
Implémente `IElementUpdater` pour mettre à jour l'état de la commande qu'il implémente.

DesignPatternTool



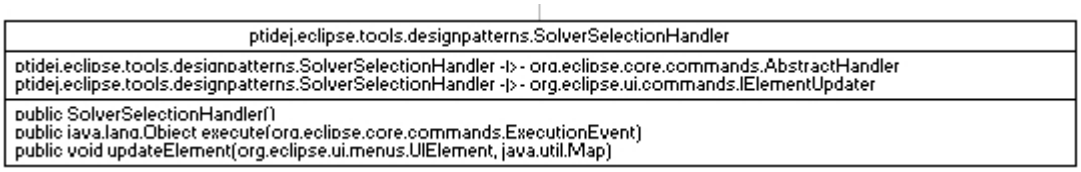
Implémentation de IPtidejTool. Classe métier de la fonctionnalité Design Pattern.

RunPtidejSolver4Handler



Handler qui gère lance le PtidejSolver4. S'appuie sur DesignPatternTool pour le code métier (indépendant d'Eclipse). Voir plugin.xml, onglet Extensions pour plus d'information.

SolverSelectionHandler



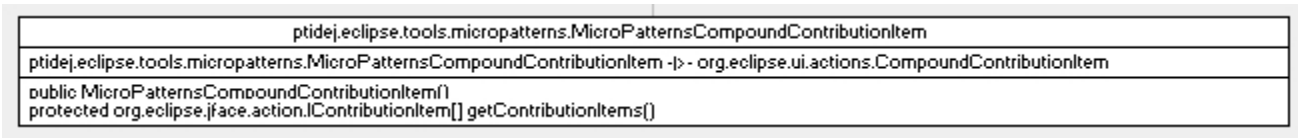
Handler qui gère l'exécution la sélection du solver qui va faire l'analyse. S'appuie sur DesignPatternTool pour le code métier (indépendant d'Eclipse). Voir plugin.xml, onglet Extensions pour plus d'information.

Implémente `IElementUpdater` pour mettre à jour l'état de la commande qu'il implémente.

ptidej.eclipse.tools.micropatterns

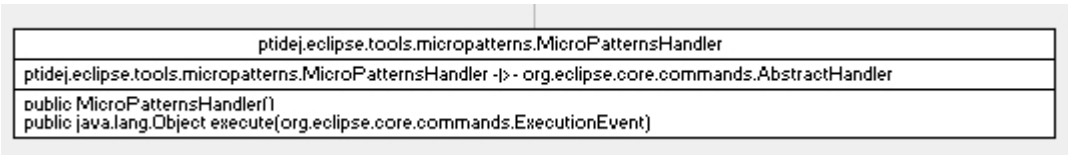
Ce package regroupe les classes implémentant les fonctionnalités de la catégorie Micro Patterns.

MicroPatternsCompoundContributionItem



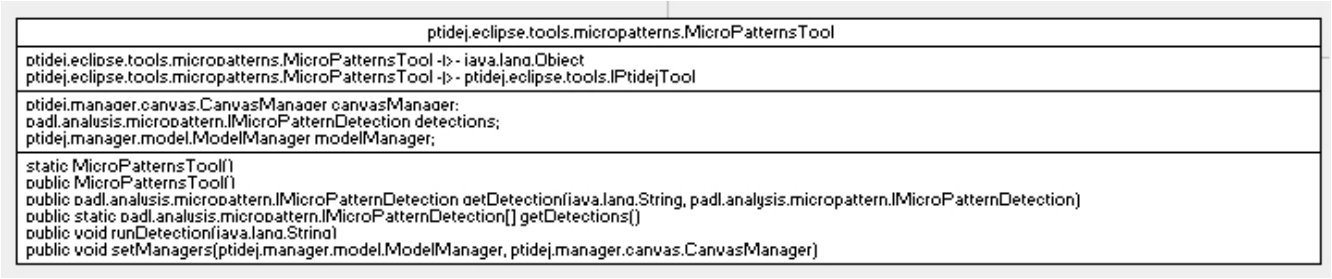
Classe qui génère la liste des micro patterns. Voir plugin.xml, onglet Extensions pour plus d'information.

MicroPatternsHandler



Handler qui gère l'exécution des commandes de la catégorie micro patterns. S'appuie sur MicroPatternsTool pour le code métier (indépendant d'Eclipse). Voir plugin.xml, onglet Extensions pour plus d'information.

MicroPatternsTool

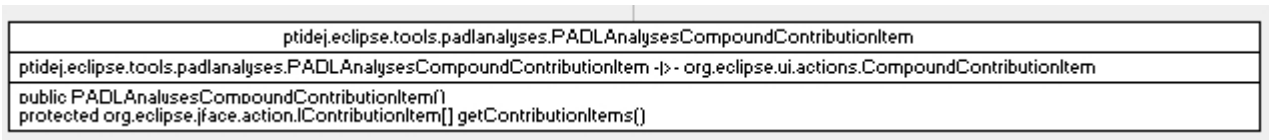


Implémentation de IPtidejTool. Classe métier de la fonctionnalité Micro Pattern.

ptidej.eclipse.tools.padlanalyses

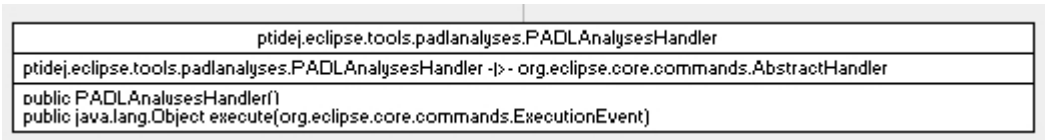
Ce package regroupe les classes implémentant les fonctionnalités de la catégorie PADL Analyses, située dans Ptidej > Tools.

PADLAnalysesCompoundContributionItem



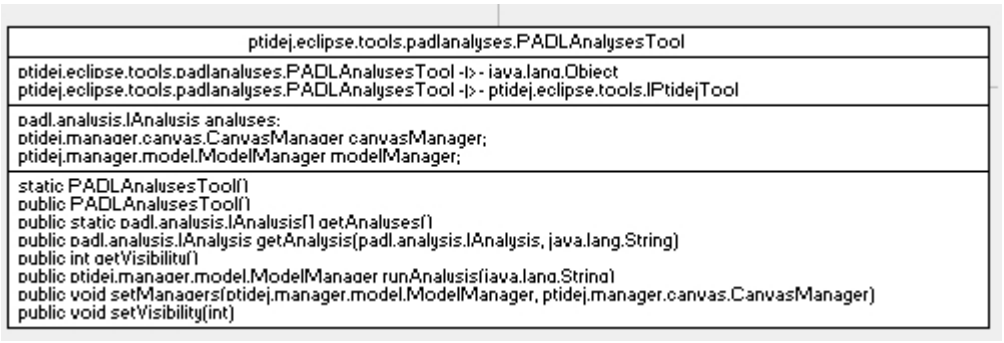
Classe qui génère la liste des analyses PADL. Voir plugin.xml, onglet Extensions pour plus d'information.

PADLAnalysesHandler



Handler qui gère l'exécution des commandes de la catégorie PADL Analyses. S'appuie sur PADLAnalysesTool pour le code métier (indépendant d'Eclipse). Voir plugin.xml, onglet Extensions pour plus d'information.

PADLAnalysesTool

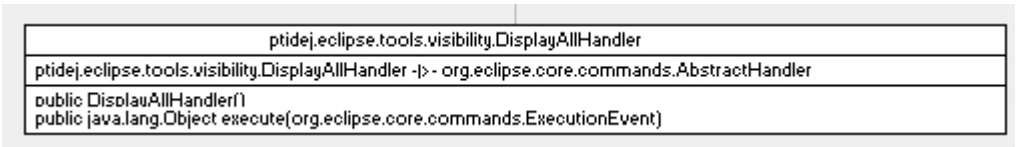


Implémentation de IPtidejTool. Classe métier de la fonctionnalité PADL Analyses.

ptidej.eclipse.tools.visibility

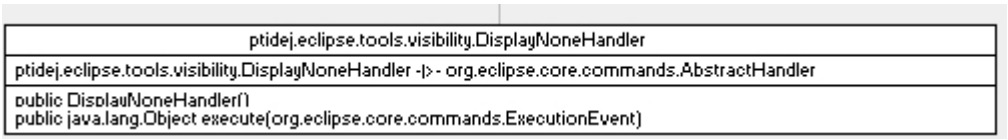
Ce package regroupe les classes implémentant les fonctionnalités de la catégorie Visibility.

DisplayAllHandler



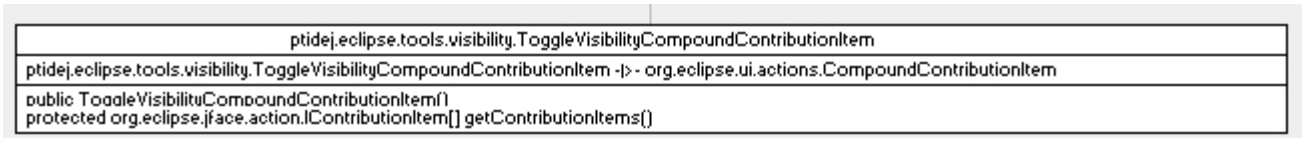
Handler qui gère l'exécution de la commande « Display All ». S'appuie sur VisibilityManager pour le code métier (indépendant d'Eclipse). Voir plugin.xml, onglet Extensions pour plus d'information.

DisplayNoneHandler



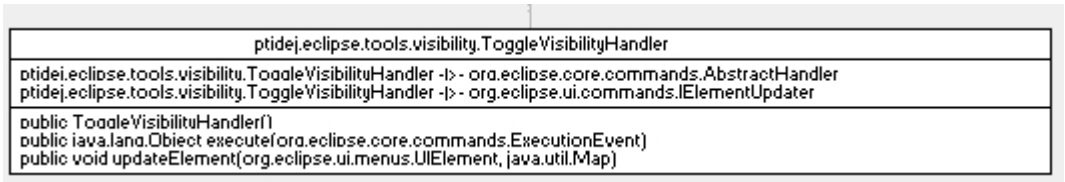
Handler qui gère l'exécution de la commande « Display None ». S'appuie sur VisibilityManager pour le code métier (indépendant d'Eclipse). Voir plugin.xml, onglet Extensions pour plus d'information.

ToggleVisibilityCompoundContributionItem



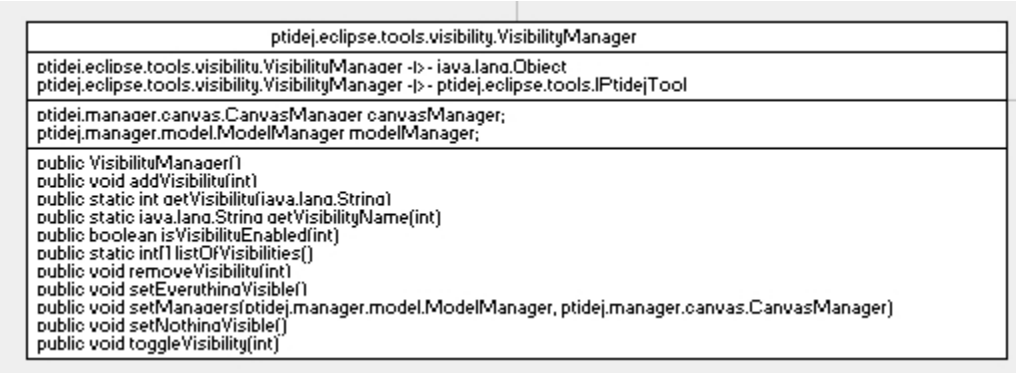
Classe qui génère la liste des options de visibilité. Voir plugin.xml, onglet Extensions pour plus d'information.

ToggleVisibilityHandler



Handler qui gère l'exécution des commandes de sélection de la visibilité. S'appuie sur VisibilityManager pour le code métier (indépendant d'Eclipse). Voir plugin.xml, onglet Extensions pour plus d'information.
Implémente `IElementUpdater` pour mettre à jour l'état de la commande qu'il implémente.

VisibilityManager

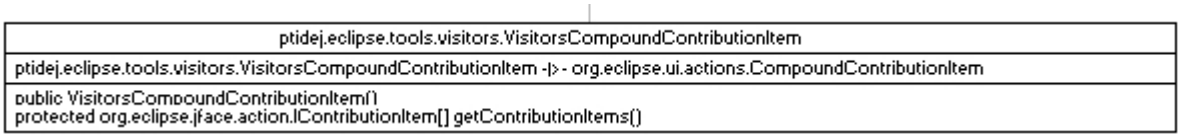


Implémentation de IPtidejTool. Classe métier de la fonctionnalité Visibility.

ptidej.eclipse.tools.visitors

Ce package regroupe les classes implémentant les fonctionnalités de la catégorie Visitors.

VisitorsCompoundContributionItem



Classe qui génère la liste des visiteurs disponibles. Voir plugin.xml, onglet Extensions pour plus d'information.

VisitorsHandler

Handler qui gère l'exécution des commandes de la catégorie Visitors. S'appuie sur VisitorsTool pour le code métier (indépendant d'Eclipse). Voir plugin.xml, onglet Extensions pour plus d'information.

VisitorsTool

Implémentation de IPtidejTool. Classe métier de la fonctionnalité Visitors.

Notes, bogues & divers

Cette partie liste les notes en vrac que j'ai pu accumuler au fur et à mesure du développement (merci Mylyn). Elles sont en français ou en anglais et la correction orthographique est désactivée, bonne chance.

- Vrai faux bug récurrent : ClassDefNotFound

Faux bug, lié au fait que les projets dont le plugin est dépendant ne sont pas des plugins. Vérifier que pour chaque dépendance, un .jar aie été défini et qu'il existe déjà. Note de rappel.

- Vrai faux bug : java 1.6

Par défaut le workspace utilise Java 1.4, or je travaille en 1.6. Il faut donc remplacer pour tous les projets java sdk 1.4 par jdk 6. Eclipse le fait par défaut mais je le note au cas où.

- Write Javadoc?

Oh no!

- Ajouter un nom à l'éditeur

A l'origine il était possible de nommer les analyse, il faudrait implémenter cette fonctionnalité, et donc créer un wizard qui demande le nom (penser aussi à l'interface).

- Analysis lifecycle

Now that a quick&dirty save/load implementation is done, a new problem appears : the analysis lifecycle. When is it dirty, when is it not, when to save it, when to load... ?

- Bug : automatic build before opening ptidej analysis

- bug : CTRL+S

Why is it not working?

- Bug : file init exception (ptidej solver 4)

Quick fix by Yann Gael. Quick and dirty fix, though.

When running ptidej solver 4, a file not found exception occurs. The same error occurs when running some other analyses. This is due to the fact that implementations are done for simple java projects, not plugin projects. File references between projects (here Ptidej Solver Data) are broken. As no IFileRepository is used in Ptidej Solver 4 implementation, what should we do ?

Error details :

```
java.io.FileNotFoundException: ..\Ptidej Solver Data\ConstraintResults.ini (Le chemin d'accès spécifié
est introuvable)
            at java.io.FileOutputStream.open(Native Method)
            at java.io.FileOutputStream.<init>(Unknown Source)
            at java.io.FileOutputStream.<init>(Unknown Source)
            at java.io.FileWriter.<init>(Unknown Source)
            at ptidej.solver.OccurrenceGenerator.callJavaConstraintSolver(OccurrenceGenerator.java:160)
            at ptidej.solver.OccurrenceGenerator.callPtidejSolver4(OccurrenceGenerator.java:202)
            at ptidej.solver.OccurrenceGenerator.getOccurrences(OccurrenceGenerator.java:564)
            at ptidej.solver.OccurrenceGenerator.getOccurrences(OccurrenceGenerator.java:524)
            at ptidej.tools.DesignPatternTool.runPtidejSolver4(DesignPatternTool.java:66)
            at
ptidej.eclipse.handlers.designpatterns.RunPtidejSolver4Handler.execute(RunPtidejSolver4Handler.java:31)
            at org.eclipse.ui.internal.handlers.HandlerProxy.execute(HandlerProxy.java:239)
            at org.eclipse.core.commands.Command.executeWithChecks(Command.java:475)
            at
org.eclipse.core.commands.ParameterizedCommand.executeWithChecks(ParameterizedCommand.java:429)
            at org.eclipse.ui.internal.handlers.HandlerService.executeCommand(HandlerService.java:165)
            at
org.eclipse.ui.internal.handlers.SlaveHandlerService.executeCommand(SlaveHandlerService.java:247)
            at
org.eclipse.ui.menus.CommandContributionItem.handleWidgetSelection(CommandContributionItem.java:555)
            at org.eclipse.ui.menus.CommandContributionItem.access$8(CommandContributionItem.java:541)
            at
org.eclipse.ui.menus.CommandContributionItem$3.handleEvent(CommandContributionItem.java:531)
            at org.eclipse.swt.widgets.EventTable.sendEvent(EventTable.java:66)
            at org.eclipse.swt.widgets.Widget.sendEvent(Widget.java:938)
            at org.eclipse.swt.widgets.Display.runDeferredEvents(Display.java:3682)
            at org.eclipse.swt.widgets.Display.readAndDispatch(Display.java:3293)
            at org.eclipse.ui.internal.Workbench.runEventLoop(Workbench.java:2389)
            at org.eclipse.ui.internal.Workbench.runUI(Workbench.java:2353)
            at org.eclipse.ui.internal.Workbench.access$4(Workbench.java:2219)
            at org.eclipse.ui.internal.Workbench$4.run(Workbench.java:466)
            at org.eclipse.core.databinding.observable.Realm.runWithDefault(Realm.java:289)
            at org.eclipse.ui.internal.Workbench.createAndRunWorkbench(Workbench.java:461)
            at org.eclipse.ui.PlatformUI.createAndRunWorkbench(PlatformUI.java:149)
            at org.eclipse.ui.internal.ide.application.IDEApplication.start(IDEApplication.java:106)
            at org.eclipse.equinox.internal.app.EclipseAppHandle.run(EclipseAppHandle.java:169)
            at
org.eclipse.core.runtime.internal.adaptor.EclipseAppLauncher.runApplication(EclipseAppLauncher.java:106
)
            at
org.eclipse.core.runtime.internal.adaptor.EclipseAppLauncher.start(EclipseAppLauncher.java:76)
            at org.eclipse.core.runtime.adaptor.EclipseStarter.run(EclipseStarter.java:363)
            at org.eclipse.core.runtime.adaptor.EclipseStarter.run(EclipseStarter.java:176)
OCCURENCES : 0
            at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
            at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)
            at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source)
            at java.lang.reflect.Method.invoke(Unknown Source)
            at org.eclipse.equinox.launcher.Main.invokeFramework(Main.java:508)
            at org.eclipse.equinox.launcher.Main.basicRun(Main.java:447)
            at org.eclipse.equinox.launcher.Main.run(Main.java:1173)
            at org.eclipse.equinox.launcher.Main.main(Main.java:1148)
```

- Bug : SWTCanvas background

Sometimes a ptidejeditorpart has no background, why? To reproduce the bug : open and close several times the same analyses, at one moment the background of some views will be gray (means no background is paint, default eclipse background). => bug with scrollbars

- Clean dependencies

Le plugin est dépendant de projets qui ne sont pas des plugins, ça craint. Faudrait transformer tous les projets en plugins propres.

- Continue adding extensions

Some functionnalities of the Swing application haven't been rewritten for Peps (solvers panel & tool panel from swing).

- Create real icons

Only one icon is used and it is ugly. There is a need to define and create new icons for the Ptidej plugin.

- JarFileRepository singleton

il y a trop d'instanciation de JarFileRepository. Voir si il est possible de le remplacer par un singleton.

- Load & save functions (.pdj?)

New : a quick & dirty fix has been done

what about saving an idiomlevelmodel or a designlevelmodel
is it possible to save the model without the occurences? the model with the occurences? etc.

Ability to load and save a Ptidej analysis (proposal : new extension .pdj).

The Swing application allows saving and loading analyses' information. Still the plugin cannot accept files from outside the workbench. Should we recreate a new format?

- ModelManager Implementation Bug : build idiom and design level models ?

The current impementation of the model manager can only build codelevelmodels. What if an analysis updates it to idiom level model and then it needs to be rebuild? This case occurs when you update a model in an idiomlevelmodel and then add a source to it : the model needs to be rebuild.

Current implementations of the build() function recreate a codelevelmodel. Should we choose this behaviour? It would be up to the user to update the model level.

- New extension point?

If a view is created for a graphical implementation in original Peps project, should we think about an extension point so that it would be easy to add new panels for each peps tool?

Pour chaque extension, il peut y avoir ou non un panneau d'options.
Il y a deux manières de concevoir la solution :
- Peps conçoit un panneau dans lequel les extensions peuvent rajouter des éléments (tabs par exemple), l'extension se charge de récupérer les tabs et de les intégrer dans son panneau.
- Chaque extension qui veut son propre panneau crée sa propre ViewPart (onglet dans eclipse). Dans ce cas il faudrait définir un site (viewpart.getSite) dans lequel ces panneaux s'enregistreraient. Il faut donc creuser la question.
Dans l'idéal il faudrait créer une perspective qui s'assure que les vues sont bien disposées.

- Normer et optimiser le code

Norme dans les noms de classe, de package, de variables, de fonctions...
Optimisation des algorithmes (exemple : affichage...).

- Package level

It would be great if Ptidej implements the package level.

- Préconfigurations de visibilité

Ce serait cool qu'en plus de visible all et visible none il y ait plusieurs configurations de visibilité.
Sauvegarder la configuration?

- Ptidej Inspector View

All is in the title.

- Ptidej Perspective

If several views are implemented, should we add a Ptidej perspective?

- Ptidej Treeview

What about an error view (done-console view), output view(done-console view), TreeView part, OptionView (panel with options)?

- Remove class from analysis

You can add a class to an analysis, how to remove one?

- Rename new analysis in new project

- Rename packages

Several packages have only one class, reorganise these packages (including the one with ISource).

- Revoir Source, implémentation de ISource

L'implémentation actuelle est bidon

- Simplifier la classe PtidejEditorPart

Créer une interface bidon avec getTool() ?
Créer des nested classes pour séparer les diverses sous-fonctions de PtidejEditorPart (une sous-class avec le code eclipse, une autre pour la gestion des menus...) ?

- Toolbar bug

The toolbar is buggy. It should show only when the opened editor is an instance of PtidejEditorPart, instead of that it is allways shown.
Partially fixed : this is a combination of two eclipse bugs.
https://bugs.eclipse.org/bugs/show_bug.cgi?id=222447
https://bugs.eclipse.org/bugs/show_bug.cgi?id=201589
The location bug cannot be fixed, the visibility bug is partially fixed : in each command (submenu) of the toolbar, I added a visibleWhen element.

- utiliser util.io.Output

Je viens d'essayer de le faire mais ça marchait pas (seules les erreurs s'affichaient). J'ai laissé tel quel en attendant.

- Where to save ptidej analyses?

Où sauvegarder les analyses? Dans le workspace? Créer un ptidej folder dans lequel enregistrer les analyses? Dans le cas d'une analyse avec plusieurs projets, où placer ce répertoire ?