CONSERVATOIRE NATIONAL DES ARTS ET METIERS

CENTRE REGIONAL RHÔNE-ALPES

CENTRE D'ENSEIGNEMENT DE GRENOBLE

EXAMEN PROBATOIRE

en INFORMATIQUE

présenté par

LÉPY Nathalie

Étude de l'environnement ouvert de développement intégré Eclipse dans l'optique d'une extension

---

Soutenu le 1er juillet 2005, à Grenoble, devant le jury :

Présidente :	Mme DONZEAU-GOUGE Véronique (CNAM Paris)
Examinateurs :	M. COURTIN Jacques (UPMF Grenoble)
	M. GIRAUDIN Jean-Pierre (CNAM, UPMF Grenoble)
	M. PLISSON André (CNAM Grenoble)
Tuteur :	M. LANG Frédéric (INRIA Rhône-Alpes)

Remerciements

Pour m'avoir fait l'honneur de participer au jury de ce probatoire, je remercie Mme Véronique Donzeau-Gouge, M. Jacques Courtin, M. Jean-Pierre Giraudin, et M. André Plisson.

Je remercie M. Frédéric Lang, mon tuteur tout au long de ce probatoire, pour sa disponibilité, la qualité de son encadrement et la finesse de sa relecture. Je le remercie aussi pour la proposition de ce sujet d'actualité et de grand intérêt.

Merci aussi à Jean-Marc, à Luc et à l'Association des Ingénieurs du CNAM Dauphiné Savoie (AIPST) pour leur relecture attentive, ainsi qu'à Fanny, Grégory et Hubert pour leur soutien moral et les discussions animées autour de sympathiques repas de travail.

Je remercie également mon mari d'avoir pris temporairement tout à sa charge, l'ensemble des petites tâches ménagères qui parsèment le quotidien d'une famille.

Merci enfin à mes enfants, Maxime et Cassandra, qui n'ont pas hésité à plonger dans les couches et les purées de bébé pour s'occuper par intérim de notre petite Aricia. Et merci aussi à Aricia pour avoir eu le bon goût de faire de longues siestes pour permettre à sa maman de travailler de manière suffisamment continue pour produire, je l'espère, un travail qui se tient…

Sommaire

Liste des figures

Liste des tableaux

Introduction

La plupart des développeurs utilisent un environnement de développement, plus ou moins sophistiqué, pour développer leur projet. Nous nous intéresserons ici en particulier à l'environnement de développement intégré (EDI) Eclipse.

De manière générale, l'EDI « *est une interface qui permet de développer, compiler et exécuter un programme dans un langage donné* » (Dico du net[1]). La plupart des langages de programmation (Java, C, C++, etc.) sont associés à un outil permettant de saisir du code, compiler, déboguer et exécuter des programmes. Cet outil combine les fonctionnalités d'un éditeur de texte, d'un compilateur et d'un débogueur. Il rend plus pratique la programmation, avec par exemple des fonctions de complétion automatique de texte pour accélérer la saisie. Il fournit en général une documentation sur les outils du langage concerné et facilite l'accès aux propriétés et méthodes des librairies utilisées. Il est naturellement possible de programmer sans EDI, en utilisant un éditeur de texte classique et le compilateur en ligne de commande que fournissent la plupart des langages courants, mais l'usage d'un EDI, outre un confort d'utilisation réellement appréciable, apporte généralement un gain de temps important pour le développement.

Il existe une multitude d'EDI et le choix n'est pas toujours aisé. Ce choix est généralement guidé par un souci de performance, mais ce n'est bien souvent pas le seul critère qui entre en ligne de compte. Certains EDI sont payants, dédiés à la programmation dans un langage donné sur une plate-forme donnée et pas ou difficilement, personnalisable (exemple, Visual C++, qui permet de programmer uniquement en C++ sous Windows). À l'opposé, d'autres EDI sont gratuits, conçus pour être personnalisés selon les différents besoins du programmeur et, de fait, multilangages et multiplateformes (exemples, Eclipse et NetBeans).
Même si nous évoquons brièvement cette question du choix dans la suite de ce rapport, ce n'est pas le thème central de notre propos. Nous nous plaçons ici dans le cas de figure où l'utilisation de l'EDI Eclipse est imposée dans le cadre d'un projet de développement donné.

Ce rapport n'a pas non plus pour ambition de présenter en détail le projet Eclipse ni l'intégralité de tout ce qu'il permet. Il serait en effet bien trop long et sans grand intérêt de présenter l'intégralité des fonctionnalités d'Eclipse. Par ailleurs, Eclipse étant en constante évolution, une présentation exhaustive deviendrait rapidement obsolète. Ce rapport de probatoire s'attache à présenter les concepts de base nécessaires pour développer sous Eclipse.

Eclipse est pour le moment souvent utilisé comme EDI Java. Dans le cadre de cette étude, le point de vue choisi sera celui d'un développeur d'outils logiciels souhaitant intégrer à Eclipse un nouveau langage et des outils associés, qui ne soient pas nécessairement développés en Java.

En dehors d'une appréhension globale du projet Eclipse, nous nous attacherons plus particulièrement ici à comprendre l'environnement Eclipse — comment le personnaliser — et ses possibilités d'extension via l'ajout de *plug-ins*. C'est pourquoi nous présenterons brièvement dans une première section la philosophie sous-jacente au développement du projet Eclipse. Puis, nous étudierons les différentes connaissances nécessaires pour pouvoir développer sous Eclipse. Nous détaillerons en particulier l'architecture de la plate-forme Eclipse afin de situer le module de l'interface utilisateur (*workbench*), objet des extensions envisagées dans le cadre de cette étude. Enfin, dans une dernière section, nous présenterons

[1] http://www.dicodunet.com/definitions/developpement/ide.htm (consulté le 24 mai 2005).

les notions centrales de *plug-in* et d'extension et décrirons les moyens mis à notre disposition par Eclipse pour faciliter le développement de nouveaux *plug-ins*.

Petite note terminologique préliminaire

Nous utilisons ici le terme « *plug-in* », plutôt que celui d'« extension » ou de « module », parce que son usage en est largement répandu au sein de la communauté des développeurs en général[2], et des utilisateurs Eclipse en particulier. Par ailleurs, le terme « extension » ne traduit pas complètement le concept de « *plug-in* ». Le verbe anglais *to plug* signifie brancher, raccorder, connecter. Le terme in précise que l'on raccorde quelque chose d'amovible en un point précis de la structure fonctionnelle d'Eclipse prévu pour recevoir cette extension (Gamma et Beck, 2004). Le terme *plug-in* est donc plus précis que celui d'extension : il précise comment doit être réalisée l'extension ou l'ajout de module.

Nous utilisons également l'expression « logiciel *open source* » plutôt que « logiciel à source ouvert » qui est très peu utilisé dans le monde des développeurs. Par ailleurs, on fait souvent l'amalgame « *open source* » et « logiciel libre » ; or, même si dans la pratique les différences entre les deux catégories sont minimes, ces deux catégories ne se recouvrent pas complètement : certaines licences sont acceptées en *open source*, alors qu'elles sont considérées trop restrictives pour la catégorie logiciel libre et inversement, certaines licences de logiciel libre n'ont pas été acceptées par l'OSI (Open Source Initiative) (Stallman, 2002). Même si Eclipse est délivré sous la licence CPL (Common Public Licence) — licence de logiciels libres — il n'est pas recensé sur le FSF[3]/UNESCO[4] Free Software Directory[5], et est présenté par les membres de la Fondation Eclipse comme un logiciel *open source*. C'est pourquoi nous parlerons ici d'*open source*.

[2] Les non-puristes n'hésitent d'ailleurs pas à utiliser le terme *plugin* plutôt que *plug-in.* Ce dernier est cependant le terme correct si l'on souhaite respecter les usages de la langue anglaise.

[3] Free Software Foundation

[4] United Nations Education, Scientific and Cultural Organization

[5] http://directory.fsf.org/index.html (consulté le 31 mai 2005).

1. Présentation générale du projet Eclipse

L'objectif de cette section étant de présenter Eclipse et la philosophie sous-jacente qui a guidé son évolution, nous présentons d'abord un bref historique, les objectifs et les enjeux économiques afin de situer le contexte de son développement. L'objectif final étant d'utiliser Eclipse comme EDI pour un nouveau langage de programmation, il est également pertinent de préciser le cadre légal de diffusion d'un tel produit, ainsi que d'examiner ses points forts et ses points faibles comparativement aux principaux autres produits de même catégorie disponibles sur le marché.

1.1. Contexte de développement d'Eclipse

1.1.1. Historique

C'est en avril 1999 que le développement d'Eclipse a été initialisé par la société Object Technology International (OTI), filiale d'IBM ayant une grande expérience du développement de produits. Depuis 1988, elle a produit VisualAge Smalltalk, VisualAge pour Java et VisualAge Micro Edition. Ainsi, Eclipse n'est pas né du néant, il est issu d'une longue lignée de développements.

Après deux ans d'effort de développement, le 5 novembre 2001, IBM donnait Eclipse sous licence *open source* à la communauté des développeurs (Girard, 2001). Ce présent a été estimé à 40 millions de dollars.

Initialement le projet était géré par un consortium, regroupant de grands éditeurs de logiciels : OTI, Borland, IBM, Merant, QNX Software Systems, Rational Software, Red Hat, SuSE, TogetherSoft et Webgain. Depuis, le consortium s'est étoffé et compte aujourd'hui plus de 50 membres dont Sysbase, BEA systems, Hitachi, Oracle, Hewlett-Packard et Intel[6]. On ne parle d'ailleurs plus de consortium depuis février 2004, quand le projet Eclipse a changé de statut pour devenir la fondation Eclipse, association à but non lucratif.

Le projet Eclipse a ainsi connu une croissance extrêmement rapide, et la fondation gère aujourd'hui beaucoup d'activités organisées en projets et sous-projets (voir section 2.1 et figure 2), encadrés par quatre groupes de directeurs : les développeurs stratégiques, les consommateurs stratégiques, les fournisseurs d'ajouts et les meneurs du projet *open source*.

Malgré sa relative jeunesse, on trouve déjà une littérature étonnamment abondante et variée sur Eclipse (voir Annexe A). Les ouvrages fondateurs tels que Gamma et Beck (2004) et Holzner (2004) sont même déjà édités en français.

1.1.2. Définition et objectifs généraux du projet Eclipse

Eclipse se définit comme une plate-forme universelle pour intégrer des outils de développement — un EDI ouvert, extensible, pour tout et n'importe quoi.

> *« Eclipse is a kind of universal tool platform — an open extensible IDE for anything and nothing in particular. »*[7]

[6] Voir http://www.eclipse.org/org/ pour une liste complète et à jour (consulté le 2 juin 2005).

[7] http://www.eclipse.org/ (consulté le 31 mai 2005).

Les objectifs généraux du projet Eclipse sont les suivants :

- fournir une plate-forme ouverte pour des outils de développement d'applications qui puisse tourner sur une large variété de systèmes d'exploitation, que ce soit pour des applications GUI (Graphical User Interface) ou non GUI ;
- ne pas imposer l'usage d'un langage (informatique ou langue naturelle) ni même d'un type de langage ;
- faciliter l'intégration d'outils sans problème d'interfaçage, que ce soit au niveau de l'interface utilisateur ou à un plus bas niveau, ou pour ajouter de nouveaux outils à des produits existants déjà installés ;
- attirer une communauté de développeurs d'outils (en capitalisant notamment sur la popularité de Java pour écrire des outils).

1.1.3. Enjeux économiques

Comme pour tout logiciel développé en *open source*, le modèle économique sous-tendant sa distribution peut ne pas paraître évident au premier abord. Comment rentabiliser une activité de développement lorsque l'on développe un logiciel *open source ?* En fait, que le développement soit *open source* n'exclut pas la possibilité d'en tirer, d'une manière ou d'une autre, des bénéfices commerciaux directs (voir section 1.2, les modalités des licences sous lesquelles Eclipse est distribué).

Perens (1999) et l'Open Source Initiative (OSI, 2005) ont défini un ensemble de critères que doit satisfaire un logiciel pour bénéficier de l'appellation *open source*. Ces critères indiquent qu'un logiciel *open source* permet à ses utilisateurs d'accéder au code source, de le modifier et de le distribuer eux-mêmes ; rien n'empêche sa distribution commerciale*,* seul l'accès aux sources doit être gratuit.

Par ailleurs, même la distribution gratuite d'un logiciel *open source* peut s'avérer rentable au bout du compte. Dans son modèle économique, Viseur (2002) présente les différents modèles de financement d'un logiciel *open source* et les stratégies sous-jacentes. Il est par exemple possible d'adopter une stratégie de partage des coûts (Raymond, 1999). C'est l'un des aspects qui motivent IBM dans son soutien de projets *open-source (Girard,* 2001), cela lui permet de diminuer le coût de développement de ses services. Par exemple IBM a basé le développement de l'EDI WSAD (*WebSphere Studio Application Developer)* sur Eclipse ; ce dernier continue d'évoluer sans qu'IBM ait à rémunérer les développeurs, l'effort de développement peut donc être totalement consacré à WSAD.

D'autre part, le marché de l'*open source* n'existe qu'avec un marché de services. Bon nombre de sociétés proposent des services permettant de faciliter l'utilisation des logiciels libres. Eclipse n'échappe pas à la règle et fait par exemple l'objet d'un grand nombre de formations et d'ouvrages commerciaux (voir Annexe A).

Une autre stratégie décrite par Raymond (1999) consiste à vendre à perte pour se positionner sur le marché. Par exemple, avec Eclipse, IBM souhaitait proposer une nouvelle étape dans la standardisation, cherchant entre autre à contrer Microsoft et son modèle VisualStudio.net.

Enfin, même si le développement ouvert de produits informatiques menace directement la propriété intellectuelle de l'entreprise qui s'engage dans cette voie, il recèle un fort potentiel en matière d'innovation. Eclipse est là encore un exemple édifiant en la matière.

1.2. Aspects légaux pour la distribution

La plate-forme Eclipse était initialement distribuée sous la *Common Programming Licence* (CPL), reconnue par la corporation Open Source Initiative (OSI). La fondation Eclipse a commencé la transition de la CPL à l'*Eclipse Public Licence* (EPL) le 9 septembre 2004.

Les licences CPL et EPL ont été réalisées pour satisfaire des intérêts commerciaux. Par exemple, elles autorisent la diffusion conjointe d'un logiciel *open source* avec un autre logiciel soumis à des conditions commerciales restrictives de par sa licence (IBM commercialise par exemple son EDI WSAD, basé sur la plate-forme Eclipse). Cette démarche se démarque d'autres dispositions plus contraignantes telles que le *copyleft*[8], qui requiert que toutes les versions modifiées et étendues du programme soient libres également.

Que l'on soit un contributeur, un redistributeur, un développeur ou même un simple utilisateur de produits basés sur Eclipse, il est conseillé de lire le guide sur la documentation légale proposé sur le site de la fondation Eclipse[9].

1.3. Comparaison d'Eclipse avec d'autres EDI

Il est intéressant de savoir comment Eclipse se positionne par rapport à d'autres EDI. Cela permet entre autre de relever les points forts et les points faibles concrètement perçus, dans la pratique, par les utilisateurs.

Selon Holzner (2004), parmi les nombreux environnements de développement intégrés disponibles pour Java, Eclipse est à ce jour le meilleur. Cet avis se trouve conforté par un sondage réalisé par developpez.com en 2004[10], où Eclipse a remporté la première place du meilleur EDI Java, loin devant IntelliJ IDEA, JCreator, Emacs, et bien d'autres ; il devançait même les pointures du marché que sont JBuilder et NetBeans.

Petite anecdote

Eclipse en tant qu'EDI Java, se positionne en concurrent direct de NetBeans développé par Sun. Des pourparlers entre IBM et Sun autour de l'intégration de Sun dans le consortium Eclipse ont un temps animé les colonnes des journaux informatiques. Sun a finalement refusé d'abandonner NetBeans, condition *sine qua non* de son intégration. Depuis, peut-être pour jeter un peu d'huile sur le feu et ainsi ranimer la flamme, certains se sont posés la question du choix du nom « Eclipse » et ont suggéré qu'IBM avait peut-être volontairement choisi ce nom très polémique (l'éclipse est un phénomène qui conduit à cacher/éclipser le soleil, *sun* en anglais). Mike Milinkovch, directeur exécutif de la fondation Eclipse s'en défend dans une interview réalisée par Girard et Rafal (2005) où il explique qu'Eclipse est juste un nom, qui a été choisi parce qu'IBM était à cette époque très orienté « e-business » et que de ce fait ils voulaient un nom commençant par e. Cette explication n'a pas convaincu tout le monde et surtout pas NetBeans qui a lancé une affiche publicitaire animée[11] incitant les développeurs à ne plus travailler dans le noir…

Eclipse est cependant bien plus qu'un EDI Java et plutôt que Sun, c'est Microsoft que le groupe cherche à éclipser en concurrençant sa suite Visual Studio (Taft, 2005).

---

8 par opposition au *copyright* traditionnel (cf. http://www.gnu.org/copyleft/copyleft.fr.html (consulté le 31 mai 2005), pour la distinction entre *copyleft* et *copyright :* « Les développeurs de logiciels propriétaires utilisent le *copyright* pour restreindre la liberté des utilisateurs; nous utilisons le *copyleft* pour la garantir. C'est pourquoi nous avons inversé le nom, en changeant '*copyright'* en '*copyleft'.* »).

9 *Guide to legal documentation for Eclipse-based content*, http://www.eclipse.org/legal/guidetolegaldoc.html (consulté le 4 juin 2005).

10 http://java.developpez.com/outils/edi/ (consulté le 2 juin 2005).

11 http://blogs.sun.com/roller/resources/alexismp/nb40anim.gif (consulté le 2 juin 2005).

1.3.1. Points forts

L'un des critères sur lequel Eclipse emporte tous les suffrages est naturellement sa gratuité. Mais cela n'aurait pas suffi à en faire, en si peu de temps (3 ans), l'un des EDI les plus utilisés en entreprise, d'autant qu'il n'est pas le seul EDI gratuit disponible sur le marché (c'est le cas aussi de NetBeans par exemple). Outre le fait qu'il soit *open source*, ce qui fait la force d'Eclipse par rapport aux autres EDI classiques est qu'aucun de ses modules n'est intégré de manière figée à son noyau (Gamma et Beck, 2004). Il n'existe pas d'outil central et monolithique auquel viendraient se greffer des outils complémentaires. Comme le montre la figure 1, l'ouverture de son noyau permet l'ajout de très nombreux *plug-ins*. Il est ainsi extensible à l'infini.

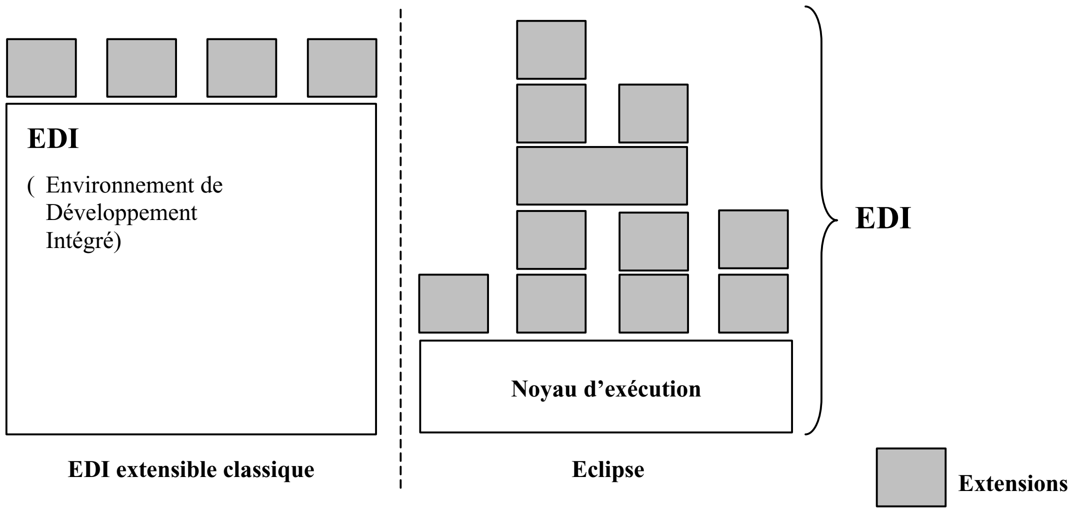

Figure 1 — Comparaison d'un EDI classique et d'Eclipse.
D'après Gamma et Beck (2004).

Eclipse présente également bien d'autres points forts (Doudoux, 2004) :

- il possède une ergonomie entièrement configurable, qui propose différentes « perspectives » selon les activités à réaliser ;
- il intègre toutes les fonctionnalités considérées comme indispensables pour un EDI (entre autres, création de projet, de modèles, *refactoring*, débogage) ;
- il contient le nécessaire pour développer de nouveaux *plug-ins ;*
- plusieurs versions d'un même *plug-in* peuvent cohabiter sur une même plate-forme ;
- le chargement des *plug-ins* est dynamique : les *plug-ins* ne sont chargés que lorsque cela est nécessaire, afin d'économiser la ressource processeur et l'espace mémoire ;
- son interface a été développée à partir de composants du kit SWT *(Standard Widgets Toolkit)* — ensemble d'outils développés par IBM — ce qui la rend à la fois plus rapide et plus conviviale qu'une interface élaborée à partir des classiques composants graphiques de AWT ou de Swing ;
- c'est un environnement de développement universel multilangages[12] (Java, C/C++, Cobol, C#, PHP, XML, UML et bien d'autres) et multiplateformes (entre autres, Windows, Linux, Mac OS X, Solaris 8) ;

[12] via l'utilisation de *plug-ins*

- la plate-forme est entièrement internationalisée : des *plug-ins* permettant d'utiliser Eclipse dans une dizaine de langues sont téléchargeables séparément ;
- ...

1.3.2. Points faibles

Comme le souligne Viseur (2002), si les qualités techniques des logiciels *open source* ont été fréquemment saluées, il n'en est pas de même de leurs interfaces. Celles-ci pêchent souvent d'un manque de finition propre aux logiciels commerciaux (Yamagata, 1997).

C'est certainement la raison pour laquelle l'amélioration de l'ergonomie a été l'un des objectifs majeurs pour les développeurs de la version 3.0 d'Eclipse[13]. Néanmoins, la courbe d'apprentissage reste plutôt abrupte. L'équipe de développement d'Eclipse en a bien conscience et travaille à son amélioration.

> *« Les utilisateurs novices d'un produit basé sur Eclipse peuvent vivre une première expérience désagréable, par surabondance d'informations. Cette première expérience pourrait être mieux vécue si le produit pouvait reconfigurer le plan de travail afin de ne proposer qu'un sous-ensemble de fonctionnalités réellement nécessaires et utilisables par un novice ; des pages de bienvenue pourraient être personnalisées pour des utilisateurs types ou des niveaux d'expérience particuliers. »* (Propos tenus par les membres du groupe de travail de la plate-forme Eclipse, rapportés par Holzner, 2004).

Pour réussir à ce que les utilisateurs puissent rapidement prendre en charge le développement d'un projet, il faut que l'environnement les incite à apprendre, en leur donnant des satisfactions dès qu'un effort est fourni, le degré de satisfaction devant naturellement être proportionnel à l'effort. Ce n'est pas encore complètement le cas actuellement avec Eclipse, mais c'est ce vers quoi l'on tend.

De plus, malgré le chargement dynamique des *plug-ins* et l'utilisation des composants SWT, Eclipse peut se révéler assez lent, que ce soit au moment du lancement de la plate-forme, ou ensuite lors de l'exécution de certains processus gros consommateurs de ressources (*build* pour Java, recherche, …). Cette lenteur peut être encore amplifiée en fonction du système d'exploitation sur lequel tourne la plate-forme : beaucoup de plaintes notamment sont issues des utilisateurs développant sous Linux. Des exemples de plaintes d'utilisateurs mécontents peuvent être consultés sur le site de NetBeans[14], qui n'est cependant peut-être pas l'entité la plus objective sur laquelle on peut s'appuyer pour dresser la liste des points faibles d'Eclipse…

Par ailleurs, on retrouve les problèmes classiques inhérents au développement *open source* : une documentation succincte qui est souvent décalée par rapport à la version courante et d'éventuels problèmes de compatibilité entre les différentes versions. Ce dernier problème est particulièrement ennuyeux dans le cas d'Eclipse où le passage à une nouvelle version peut conduire à des incompatibilités avec les versions de *plug-in* existantes. Il faut ensuite un laps de temps plus ou moins conséquent (généralement la communauté *open source* est plutôt réactive) pour que tous les *plug-ins* soient à leur tour adaptés.

Enfin, sur un autre plan, il y aurait fort à faire pour améliorer l'ergonomie du site web de la fondation Eclipse, notamment concernant les aspects navigation et recherche. Le projet Phœnix travaille sur ce point.

---

[13] Voir les thèmes d'amélioration présentés dans l'*Eclipse Project 3.0 Plan (Final)*, http://www.eclipse.org/eclipse/development/eclipse_project_plan_3_0.html#TargetOperatingEnvironments (consulté le 2 juin 2005).

[14] Rubrique « Histoires vraies de personnes qui ont migré vers l'EDI NetBeans. http://fr.netbeans.org/edi/migrer/temoignages.html (consulté le 2 juin 2005)

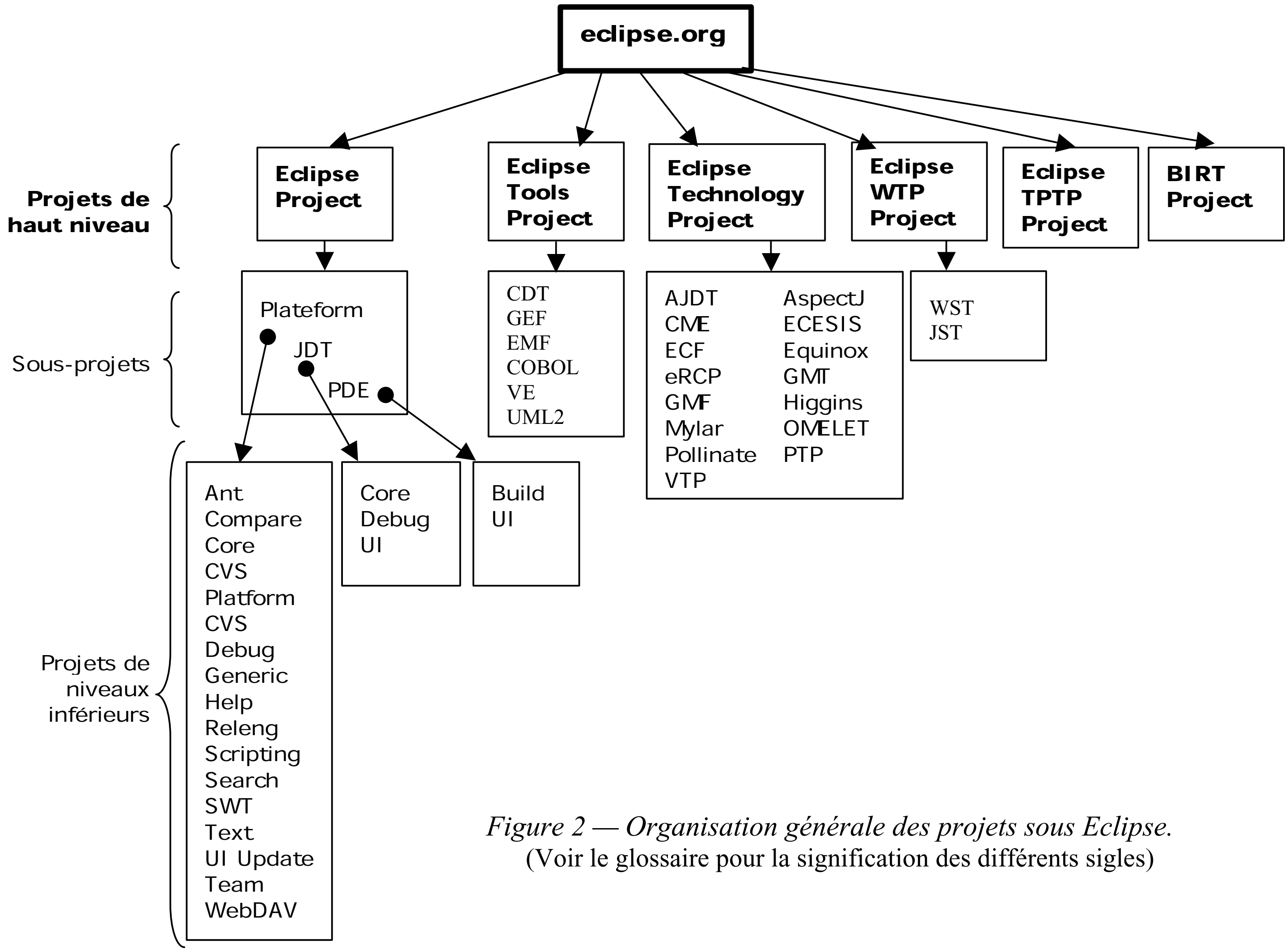

Figure 2 — Organisation générale des projets sous Eclipse.
(Voir le glossaire pour la signification des différents sigles)

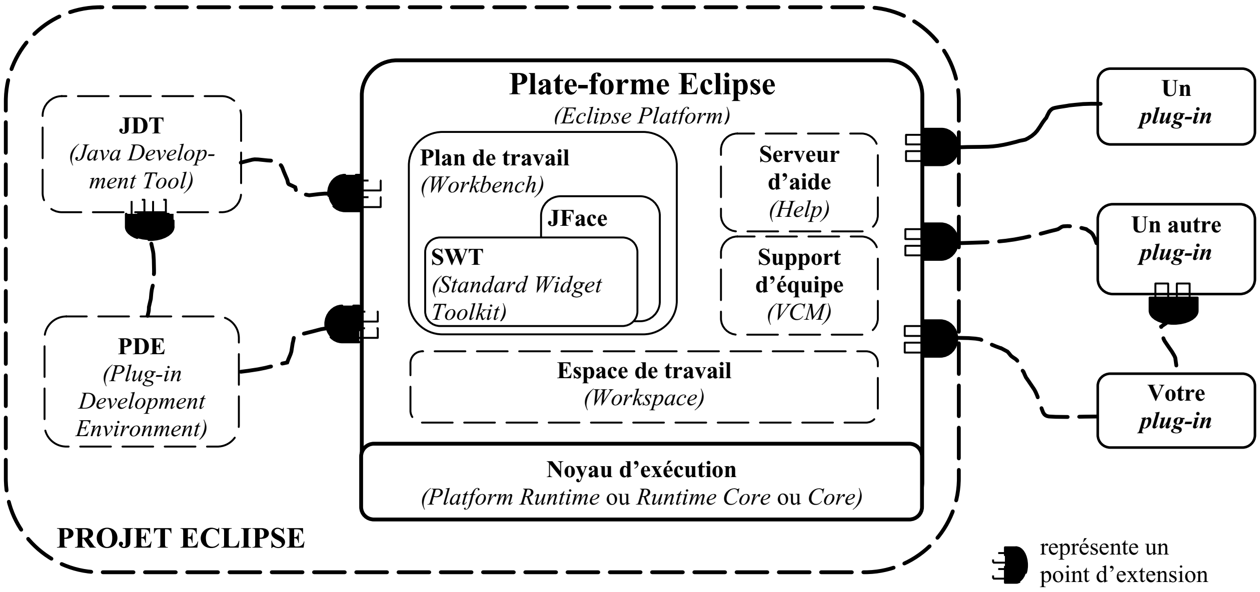

Figure 3 — Vue générale de l'architecture d'Eclipse.
D'après http://eclipse.org/eclipse/presentation/eclipse-slides_files/v3_document.htm
(transparent 5, documentation Eclipse, consultée le 4 juin 2005).
Avec VCM pour Versioning and Configuration Management.

2. Développer avec Eclipse

Cette section permet d'appréhender les différents concepts clés et connaissances indispensables sur le cœur d'Eclipse, pour pouvoir à la fois développer avec Eclipse et contribuer à son développement. Elle présente l'organisation générale des différents projets d'Eclipse et décrit plus précisément le *Eclipse Project* et l'architecture de la plate-forme Eclipse.

De plus, dans le cadre des objectifs définis en introduction, nous souhaitons pouvoir personnaliser l'interface utilisateur d'Eclipse (*workbench*) en fonction de nos besoins, il est donc nécessaire de savoir comment se présente cette interface.

2.1. Organisation générale

Le projet global Eclipse est constitué de projets hiérarchisés par niveau (voir figure 2), travaillant sur différents aspects de développement de la plate-forme Eclipse. Des informations plus détaillées sur ces projets sont disponibles en ligne[15].

Deux projets nous intéressent plus particulièrement dans le cadre de ce rapport, leurs objectifs sont définis comme suit :

- *Eclipse Project* : adapter et faire évoluer l'architecture et la structure de la plate-forme Eclipse en fonction des besoins de la communauté Eclipse et de ses utilisateurs.
- *Eclipse Tools Project* : fournir un point central pour la construction de divers outils afin de faciliter la création d'une variété d'outils la plus large possible pour la plate-forme Eclipse (illustration d'un projet de ce type en section 3.2.2).

L'ensemble du projet Eclipse (*Eclipse Project*) est constitué de trois sous-projets comprenant :

- la plate-forme Eclipse elle-même, qui constitue l'ossature de base de toute l'application ;
- les outils de développement Java (JDT — Java Development Tools) ;
- l'environnement de développement de *plug-ins* (PDE - Plug-ins Development Environment), qui permet à chaque développeur de programmer ses propres outils pour Eclipse.

Notons que JDT et PDE sont en fait des *plug-ins*, se greffant à la plate-forme Eclipse au même titre que les autres *plug-ins* (voir figure 3). Ils sont cependant livrés dans le package d'installation Eclipse de base (SDK, Standard Development Kit). Ceci est d'ailleurs compréhensible dans la mesure où l'évolution d'Eclipse est conditionnée par l'ajout de nouveaux *plug-ins* qui s'écrivent en Java et doivent être connectés de manière adéquate à la plate-forme Eclipse (ce que facilite l'usage du PDE).

2.2. Architecture de la plate-forme Eclipse

La plate-forme définit un ensemble de structures et services communs dont le rôle est d'assurer l'interopérabilité des outils qui s'y branchent (*plug-ins*). C'est l'équivalent du noyau pour un système d'exploitation.

[15] http://www.eclipse.org/projects/index.html (consulté le 31 mai 2005).

La plate-forme est constituée de cinq composants (voir figure 3) : le noyau d'exécution (*runtime core* ou *core* ou *runtime platform*), l'espace de travail (*workspace*), le plan de travail (*workbench*), le support d'équipe et le serveur d'aide. Elle comporte également des points d'extension permettant de venir connecter d'autres *plug-ins* (sur la figure 3, les points d'extension sont représentés par des prises/*plug*).

2.2.1. Noyau de la plate-forme

Le rôle du noyau est la gestion générale des *plug-ins* :

- il lance au bon moment les outils logiciels constituant Eclipse ;
- il charge dynamiquement les *plug-ins* dans l'espace de travail au moment où ils sont nécessaires
- il recherche les nouveaux *plug-ins*, initialise les *plug-ins* existant et maintien un registre d'information à leur propos.

En dehors du noyau, tout dans Eclipse est *plug-in*.

2.2.2. Plan de travail

L'interface utilisateur (IU) Eclipse est appelée *plan de travail* ou *workbench*. Le plan de travail est un élément central d'Eclipse qui sera présenté plus en détail en section 2.3.

Le plan de travail a été développé en utilisant les composants de la boite à outils SWT (*Standard Widgets Toolkit*) et la librairie JFace. SWT est une API de bas niveau proposant des objets (*widgets*) qui permettent la création d'interfaces graphiques, mais qui nécessitent aussi énormément de code. C'est pourquoi il est associé à JFace. Cette bibliothèque propose un certain nombre de classes encapsulant de nombreuses opérations de base et elle facilite ainsi le développement des interfaces graphiques reposant sur SWT.

Petite aparté sur SWT

La première API (Application Programming Interface) pour développer des interfaces graphiques portables d'un système à un autre en Java est AWT. Cette API repose sur les composants graphiques du système sous jacent, ce qui lui assure de bonnes performances. Malheureusement, ces composants ont des fonctionnalités limitées, car ils représentent le plus petit dénominateur commun aux différents systèmes concernés.

Pour pallier ce problème, Sun a proposé une nouvelle API, Swing. Cette API est presque exclusivement écrite en Java, ce qui assure sa portabilité. Bien que reconnue et éprouvée, Swing présente deux gros inconvénients : sa consommation en ressources machine et la lenteur d'exécution des applications qui l'utilisent.

Librairie développée par IBM, SWT propose une approche intermédiaire : utiliser autant que possible les composants natifs du système et implémenter les autres composants en Java. Les trois avantages de SWT sont la rapidité d'exécution, des ressources machines moins importantes lors de l'exécution et un rendu parfait des composants graphiques selon le système utilisé puisqu'il utilise des composants natifs (il n'a donc pas le « *look* » Java). SWT présente malgré tout lui aussi quelques inconvénients :

- Il dépend du système utilisé lors de l'exécution, puisque pour fournir l'apparence native d'une application, il fait appel aux bibliothèques logicielles du système d'exploitation sur lequel l'application est exécutée. Cette approche va à l'encontre des pratiques historiques liées aux développements d'application Java et a suscité à ce titre une forte controverse chez un grand nombre d'utilisateurs pour qui Eclipse se devait, tout comme Java, d'être totalement indépendant du système d'exploitation.

- Il n'utilise pas de purs objets java, ce qui rend impossible l'utilisation du ramasse-miettes Java pour libérer la mémoire des composants créés. Cette mémoire doit être libérée explicitement en invoquant la méthode *dispose()*.

2.2.3. Espace de travail

L'espace de travail Eclipse, ou *workspace*, peut être considéré comme une sorte de réservoir de toutes les ressources nécessaires au travail du développeur. Tout code écrit fait partie d'un *projet* auquel est attribué un dossier dans le répertoire de travail Eclipse. Dans la terminologie Eclipse, le terme *ressource* fait référence à ces différents éléments du noyau de l'espace de travail que sont les fichiers, les dossiers et les projets. Les fichiers et dossiers sont reliés logiquement au système de fichiers de la machine hôte.

Le développeur dispose d'un certain nombre d'outils lui permettant de gérer l'ensemble des ressources, comprenant les opérations classiques de gestion d'un système de fichiers, à savoir : lecture, création, modification et suppression de ressources.

Eclipse gère par ailleurs la sauvegarde des modifications effectuées dans chaque ressource, mémorisant au passage l'historique des changements (information utilisée notamment par le support d'équipe, voir la section suivante).

2.2.4. Support d'équipe

Lorsque des développeurs travaillent en équipe, il est indispensable que leurs différentes tâches soient bien coordonnées, que ce soit sur les aspects conception, réalisation ou planning de travail. Le support d'équipe, ou *Versioning and Configuration Management* (VCM), est à ce titre un *plug-in* Eclipse très intéressant, car il a en charge le contrôle version (*versioning*) du code du programme : au fur et à mesure du développement, ce dernier est enregistré dans une archive, ou extraite de celle-ci, afin de pouvoir reconstituer par la suite l'historique des modifications du code.

Ce composant fonctionne comme un client CVS (*Concurrent Version System*) interagissant avec le serveur CVS. Il assure ainsi un contrôle des différentes versions du code et encadre efficacement le développement d'applications en équipe et la gestion du travail de ses membres à l'aide de l'outil CVS.

Petit aparté sur CVS

CVS est un système de contrôle de version développé en *open source*. Il est supporté par la plupart des systèmes d'exploitation[16]. De nombreux autres types d'archivages sont disponibles, dont certains plus puissants que CVS, mais CVS est le plus répandu (peut-être parce qu'il est gratuit).

Le contrôle du code source s'appuie sur des outils de contrôle d'accès au code et de maintien d'un historique des modifications effectuées dans celui-ci. L'existence d'un historique du code source constitue un outil très puissant qui permet :

- de comparer une version récente d'un code à une version plus ancienne ;
- de revenir à une version antérieure si les modifications récentes ont débouché sur des catastrophes ingérables.

Le contrôle du code source permet par ailleurs de coordonner le développement simultané de différentes versions d'un logiciel (gestion de branches parallèles de développement). Par exemple, un développeur peut vouloir travailler simultanément sur la version diffusable de son code et sur une nouvelle version béta.

[16] voir www.cvshome.org (consulté le 31 mai 2005)

D'un point de vue plus pratique, les fichiers partagés sont stockés dans le *référentiel* CVS. On distingue en général deux modèles de référentiel :

- *Verrouillage pessimiste* — Un fichier donné ne peut être réservé que par un seul programmeur à la fois. Lorsqu'un fichier est réservé, celui-ci est verrouillé, au sens où les autres programmeurs ne peuvent récupérer que des copies en lecture seule du fichier. L'accès au fichier est séquentiel.
- *Verrouillage optimiste* — Les fichiers peuvent être réservés et modifiés librement par tous les programmeurs. Lorsque la modification d'un fichier est validée, le logiciel de gestion du référentiel fusionne les modifications effectuées automatiquement. S'il ne peut mener à bien cette opération, il notifie au programmeur qu'il lui faut faire ce travail lui-même. L'accès au fichier est libre.

Par défaut, la plupart des logiciels CVS suivent les règles du verrouillage optimiste. C'est également le cas d'Eclipse.

Remarques : si on travaille seul, l'installation d'un serveur CVS pour gérer les versions n'est pas indispensable. Il existe un historique local de taille paramétrable. Cet historique n'assure cependant pas la protection contre la perte de code en cas de grosse catastrophe suite à des modifications importantes.
D'autre part, il est tout à fait possible d'utiliser un autre serveur que CVS si on le souhaite, à condition toutefois qu'il soit compatible avec Eclipse.

2.2.5. Serveur d'aide

Comme son nom l'indique, le serveur d'aide apporte des informations venant aider le programmeur. L'aide est proposée à travers un navigateur web standard. C'est un système de documentation extensible : tout *plug-in* peut fournir une documentation HTML au format XML, qui indique comment naviguer dans l'aide. Des points d'extension permettent d'apporter des contributions concernant des livres entiers (ensemble de pages sur un sujet donné, il y a par exemple un guide sur PDE, un guide d'utilisation sur CDT, etc.), des sections (rubriques) de livres existants, ou encore, des fenêtres d'aide *pop-ups* accessibles à partir de la touche F1. Des mécanismes d'aide peuvent ainsi être disponibles pour tous les *plug-ins*.

2.3. Détail du plan de travail (Workbench)

Décrire dans le détail l'interface Eclipse, appelée *plan de travail* ou *workbench*, nécessiterait plus d'un livre, ce n'est donc pas envisageable dans le cadre de ce rapport. Trois concepts de base de l'interface utilisateur doivent néanmoins être compris : les éditeurs, les vues et les perspectives.

La plupart du temps, les notions de vue, d'éditeurs et de perspectives sont transparentes, voire triviales pour l'utilisateur manipulant quotidiennement Eclipse. Ce sont pourtant des notions importantes dans le monde d'Eclipse, surtout si l'on se place dans une perspective d'extension (voir section 3) et/ou si l'on souhaite personnaliser l'environnement de développement en fonction de la tâche que l'on est en train d'accomplir. L'objectif de cette section sera donc de présenter ces trois concepts importants.

La figure 4 illustre l'une des apparences visuelles que peut prendre le plan de travail Eclipse. Ce plan de travail comporte toutes sortes de barres d'outils et de menus destinés à faciliter le travail du développeur. C'est un environnement multifenêtres, constitué d'un certain nombre d'éléments graphiques, de vues, d'éditeurs et de perspectives.

2.3.1. Vues

Dans cet environnement, chaque fenêtre, appelée *vue*, présente l'état des développements selon un certain point de vue (Holzner, 2004). Sur la figure 4, la vue *Navigateur* (dans le coin supérieur gauche) montre tous les projets développés par le programmeur. Il offre ainsi une vue de l'espace de travail organisé selon la présentation classique d'un système de fichiers.

Les vues se complètent les unes les autres. Par exemple, la vue *Propriétés* décrit un élément sélectionné dans une autre vue.

Afin de gérer au mieux l'espace visuel à l'écran, Eclipse propose un système d'onglets permettant de basculer d'une vue à l'autre. La vue *Package*, par exemple, est empilée sous la vue *Navigateur* et présente un autre point de vue sur l'espace de travail, organisé cette fois en hiérarchie de *Packages* (paquets de classes Java).

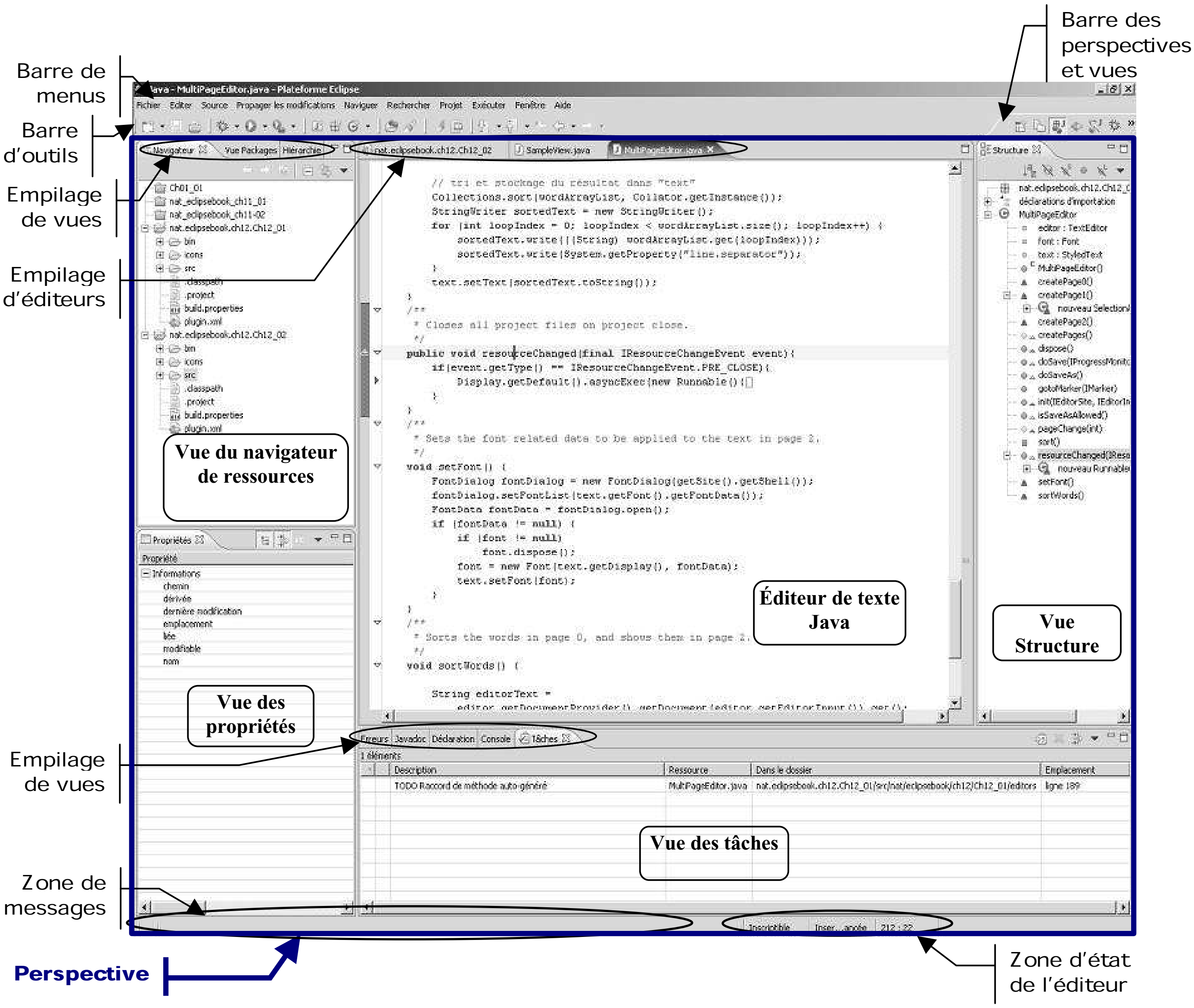

Figure 4 — Une illustration du plan de travail Eclipse.

2.3.2. Éditeurs

L'éditeur est une fenêtre spéciale qui apparaît en général au centre du plan de travail. On peut considérer l'éditeur comme une sorte de vue permettant de réaliser des actions plus complexes.

L'éditeur permet de créer et modifier les fichiers ressources. Il y a donc plusieurs types d'éditeur : l'éditeur affiché est fonction du type de fichier sur lequel on travaille. Par exemple, l'éditeur invoqué pour un fichier texte simple sera différent de celui invoqué pour un fichier Java. L'éditeur Java permettra de réaliser des actions telles que l'on peut attendre de tout éditeur commercial Java comme la mise en évidence de la syntaxe, l'indentation automatique, etc., ce que ne fera pas l'éditeur de fichier texte. La figure 4 montre un éditeur Java. On remarque par exemple que les mots-clés du langage sont colorés. La figure 5, section 3.3.1, illustre un éditeur plus complexe, multipages, pour les *plug-ins*.

Il est possible d'ouvrir simultanément plusieurs fichiers de type différents et donc d'afficher en même temps plusieurs éditeurs différents. Tout comme les vues, ils sont alors empilés et accessibles par l'intermédiaire d'onglets.

2.3.3. Perspectives

Les perspectives permettent de spécifier des ensembles pré-définis de vues et d'éditeurs, qui sont automatiquement appelés dans un contexte donné (Holzner, 2004).

En général, le programmeur ne décide pas de lui-même des vues et des éditeurs associés à un projet : le choix de ceux-ci est conditionné dans le cadre de perspectives. Une perspective est un groupe de choix liés à un type de développement. Le programmeur peut néanmoins créer ses propres perspectives et il peut passer rapidement d'une perspective à une autre par l'intermédiaire de la barre d'outils associée (voir figure 4).

3. Les extensions ou *plug-ins*

La philosophie sous-tendant le développement de plug-ins *peut être résumée par cette maxime : ne réinventez pas la roue !*

L'environnement Eclipse est vaste et basé, nous l'avons dit, sur l'ajout de composants par *plug-in*, permettant de faciliter la création, l'intégration et l'utilisation d'outils logiciels. De nombreux langages (Java, C++, etc.) et outils logiciels (compilateurs, *débogueurs*, vérificateurs de code, etc.) ont ainsi été intégrés à Eclipse, via des *plug-ins*.

Cette section définit et explique comment utiliser les *plug-ins* et décrit les différents concepts et principes régissant la construction de nouveaux *plug-ins.*

3.1. Définitions

Un *plug-in*, ou *extension*, est un module que l'on peut brancher sur un point d'entrée particulier d'une application. Eclipse peut ainsi être vu comme une collection d'emplacements destinés à recevoir des modules complémentaires : ces emplacements sont appelés *points d'extension* (voir figure 3). Cette notion est vraiment très importante pour le développement d'Eclipse qui repose sur les contributions que chaque développeur peut apporter — comme nous l'avons expliqué en section 1.3.1, c'est d'ailleurs là que réside l'originalité et la force de l'EDI Eclipse par rapport aux autres EDI.

Petit aparté sur le choix du terme *plug-in*

Arthorne et Laffra (2004) soulignent que, rétrospectivement, l'usage du terme « *plug-in* » n'était pas forcément le plus judicieux pour désigner les composants d'une application Eclipse. En effet, ce terme suppose l'existence d'une prise, d'une machine monolithique ou d'une grille, dans laquelle on se branche. Ce n'est pas le cas dans Eclipse où un *plug-in* se connecte avec un ensemble d'autres *plug-ins* pour former une application exécutable. Pour ces auteurs, une meilleure analogie serait de comparer le *plug-in* à un objet au sens de la programmation orientée objets : tout comme l'objet, le *plug-in* encapsule un comportement (actions) et/ou des données qui interagissent avec d'autres *plug-ins* pour former un programme exécutable.

Le *plug-in* est la plus petite unité de fonction qui peut être développée séparément. Il existe des *plug-ins* de différentes tailles : généralement un petit outil sera écrit en un seul *plug-in* (exemple d'un *plug-in* qui ne comporte que l'action de créer un fichier .zip), alors que les fonctionnalités d'un outil plus complexe seront réparties dans plusieurs *plug-ins* (exemple de l'éditeur HTML).

Petit aparté sur les fragments de *plug-ins*

En fait, on peut trouver une unité plus petite que le *plug-in*, il y a le fragment de *plug-in*. En effet, il existe un mécanisme qui permet à un *plug-in* d'être synthétisé à partir de plusieurs fragments séparés, localisés chacun dans leur propre répertoire ou URL. Les fragments sont généralement utilisés dans deux cas particuliers :

- isoler les dépendances des systèmes d'exploitation ;
- permettre l'internationalisation, chaque fragment contenant alors les traductions d'une langue naturelle donnée.

Le *plug-in* logique résultant regroupe le *plug-in* de base et les fragments.
La construction de fragments de *plug-in* est similaire à la création de *plug-in* classique (voir section 3.3) : ils sont localisés dans leur propre sous-dossier et contiennent quasiment les mêmes fichiers, mais ils sont instables séparément (ne peuvent être utilisés que conjointement avec le *plug-in* de base).

3.2. Utiliser des *plug-ins*

La plate-forme Eclipse fournit le support logiciel nécessaire au bon fonctionnement des *plug-ins* installés. L'installation d'un *plug-in* est très simple : il suffit de copier le répertoire le contenant dans le répertoire « *plugins* » et de redémarrer Eclipse (les *plug-ins* ne peuvent pas être ajoutés après le démarrage). On peut néanmoins s'interroger sur la performance d'Eclipse en fonction du nombre de *plug-ins* installés.

En effet, l'environnement Java et la base Eclipse constituent plus de 60 gros *plug-ins* et le kit pour la version française en rajoute plus d'une centaine. La moindre extension utilisée va à son tour augmenter le nombre de *plug-ins* présents au démarrage d'Eclipse. Le chargement de chacun de ces *plug-ins* consomme quelques millisecondes. De plus, chaque contribution peut contenir une quantité de code non négligeable et le chargement des exécutables Java correspondants (fichiers d'archives .jar) peut demander plusieurs secondes[17]. C'est pourquoi lors de chaque lancement, Eclipse vérifie les *plug-ins* présents dans le répertoire *plugins*, mais ceux-ci ne sont chargés que lorsque cela est nécessaire, afin d'économiser la ressource processeur et l'espace mémoire. Ce chargement dynamique des *plug-ins* est naturellement complètement transparent pour l'utilisateur, qui ne sait pas quand l'exécution d'un *plug-in* termine et quand l'exécution d'un autre commence. Dans tous les cas, à partir du moment où un *plug-in* est chargé, il le reste jusqu'à ce qu'on quitte la plate-forme Eclipse.

Exemple de chargement dynamique

On dispose d'une contribution dans laquelle un menu est ajouté dans la barre des menus.

- Lors du chargement d'Eclipse, la contribution est enregistrée dans le registre des *plug-ins* disponibles et traitée sans que le *plug-in* correspondant ne soit activé : le menu est construit à partir des informations contenues dans un fichier spécial (le fichier manifeste, voir section 3.3.2).
- Le *plug-in* n'est activé que lorsque cela est nécessaire : dans notre exemple, le *plug-in* ne sera activé que lorsque l'utilisateur sélectionnera un item du menu.

En général, les *plug-ins* portent un nom tel que *org.eclipse.swt.win32_2.1.1* ou *org.junit_3.8.1*, qui permet d'identifier le nom du *plug-in* et son numéro de version. Lorsque Eclipse charge un *plug-in* stocké dans plusieurs sous-répertoires du répertoire *plugins*, le numéro de version est utilisé par Eclipse pour que seule la version la plus récente soit chargée.

3.2.1. Plug-ins *existants*

La liste des *plug-ins* en cours d'utilisation par la plate-forme Eclipse est consultable dans la vue *plug-ins* de l'interface utilisateur. Si on souhaite utiliser un *plug-in* qui n'est pas dans cette liste il faut le télécharger et éventuellement le payer.

Le site *eclipse-plugins.info*[18] propose une liste des contributions (gratuites ou commerciales) disponibles pour Eclipse. Une contribution peut être constituée de un ou plusieurs *plug-ins.* Au 5 juin 2005, le nombre de contributions total indiqué était de 860 ; au 29 mai 2005, on en comptait 851, deux jours avant on en recensait 847, etc. Le nombre de contributions existantes est donc important et en constante évolution, il est de ce fait impossible d'en donner ici une liste exhaustive. Pour faciliter la recherche, ces contributions ont été classifiées et répertoriées dans différentes catégories (voir tableau 1).

[17] Concevoir des classes Java de telle manière que leur chargement soit accéléré suppose des connaissances pointues que peu de programmeurs ont le temps d'acquérir.

[18] *http://eclipse-plugins.info/eclipse/plugins.jsp* (consulté le 3 juin 2005)

Tableau 1 — Catégories de plug-ins *recensées sur* eclipse-plugins.info.
(D'après http://eclipse-plugins.info/eclipse/plugins.jsp)
Le nombre entre parenthèse correspond au nombre de contributions dans chaque catégorie.

All (860)	Mobile/PDA (13)	Source Code Analyzer (23)
Bug Tracker (6)	Modelling (19)	Team (7)
Code Generation (16)	Network (10)	Testing (31)
Code Generation/Modelling (5)	News (7)	Tomcat (5)
Code mngt (77)	Obsolete (1)	Tools (71)
Database (53)	Patterns (4)	Tutorial (37)
Deployment (18)	Profiling (16)	UI (33)
Documentation (27)	Project management (5)	UI components (11)
Editor (21)	Real-time (1)	UML (20)
Electronics (1)	Research (4)	Utility (37)
Entertainment (38)	Rich Client (8)	Web (39)
Graphics (4)	Safety-critical (2)	Web Service (23)
J2EE development platform (43)	SCM (22)	XML (41)
Languages (52)	Snipets (2)	
Middleware (6)	Snipets/Modelling (1)	

3.2.2. Regroupements de plug-ins *— Eclipse Tools Project*

Lorsque les *plug-ins* concernant une contribution donnée sont nombreux et issus de diverses sources, sur un sujet jugé d'intérêt collectif, il est possible de demander la création d'un projet de type *Eclipse Tools Project*.

Il y a six gros projets de cette nature actuellement en chantier (voir figure 2). Le projet CDT[19] par exemple (*C/C++ Development Tools*) s'attache à fournir un EDI complet fonctionnel pour les langages C et C++ sur la plate-forme Eclipse.

Les langages C et C++ font parti des langages de programmation les plus populaires et les plus largement utilisés dans le monde (Leszek, 2003). Malheureusement, l'ensemble des *plug-ins* disponibles pour C et C++ est beaucoup moins fourni que pour Java. La présentation du projet CDT sur le site de la fondation Eclipse comprend d'ailleurs un appel à contributions.

Les fonctionnalités proposées par le kit CDT pour assister le développement d'applications C/C++ comprennent actuellement :

- un éditeur (*editor*) : fonctionnalités basiques, mise en évidence de la syntaxe, complétion de code, etc. ;
- un débogueur (*debbuger*) ;
- un exécuteur (*launcher*) ;
- un compilateur (*parser*) ;
- un moteur de recherche (e*ngine search*) ;
- un fournisseur d'aide au contenu (*content assist provider*) ;
- un générateur de *makefile* (*makefile generator*).

[19] http://www.eclipse.org/cdt (consulté le 6 juin 2005).

3.3. Créer de nouveaux *plug-ins*

Petit aparté sur l'ergonomie de l'IU

Eclipse est une plate-forme très flexible et très extensible, mais cette flexibilité présente cependant un sérieux revers : il n'y a aucun moyen depuis le programme d'assurer la cohérence de l'interface utilisateur entre les composants enregistrés à l'intérieur de la plate-forme. C'est pourquoi Edgar *et al.* (2004) ont rédigé un document définissant les règles à respecter lors de la création de nouveaux composants pour maintenir la cohérence. Ce manuel peut être consulté utilement par tout contributeur potentiel à la plate-forme Eclipse.

La création de nouveaux *plug-ins* peut se faire de deux manières différentes : soit en utilisant un des assistants fournis par l'environnement de développement de *plug-ins* PDE ; soit en partant d'un projet vide et en créant quasiment tout « à la main ». Dans tous les cas, il peut être utile de connaître comment est structuré un *plug-in*.

3.3.1. *Architecture d'un* plug-in

L'architecture d'un *plug-in* est modulaire. En parcourant le répertoire *plugins* d'Eclipse, on constate que tout sous-répertoire de *plug-in peut* contenir les fichiers suivants :

- *about.html* : fichier affiché lorsque l'utilisateur souhaite obtenir des informations sur le *plug-in* ;
- **.jar* : fichier archive contenant le code Java du *plug-in* ;
- *icons* : répertoire qui contient des icônes (au format .gif par défaut) qui peuvent par exemple être utilisées pour agrémenter un bouton sur la barre d'outils ;
- *lib* : ce répertoire contient les fichiers .jar complémentaires ;
- *plugin.properties* : ce fichier contient les valeurs des variables chaînes de caractères utilisées par le fichier *plugin.xml* ;
- *plugin.xml* : ce fichier XML (eXtensible Markup Language) est encore appelé *fichier manifeste*, il contient la description globale du *plug-in*.

Seul ce dernier fichier *plugin.xml* est obligatoire. Tous les autres sont facultatifs. Certains *plug-ins* ne contiennent pas du tout de code, mais la plupart du temps cependant, la création d'un *plug-in* implique la création de plusieurs fichiers, en particulier des fichiers code. Ces derniers doivent être rédigés en Java qui est le langage de développement d'Eclipse.

Les fichiers énumérés ci-dessus sont ceux qui constituent le corps du *plug-in* une fois qu'il est déployé. Lorsque l'on crée un *plug-in*, la structure du répertoire de travail contient au minimum les éléments suivants :

- *bin* : répertoire contenant les fichiers Java compilés (.class) ;
- *src* : répertoire contenant les fichiers Java source (.java) ;
- *build.properties* : ce fichier indique à l'environnement de développement où se situe le fichier du code source Java servant à créer le fichier .jar (fichier archive Java) ;
- *plugin.xml* : voir définition ci-dessus.

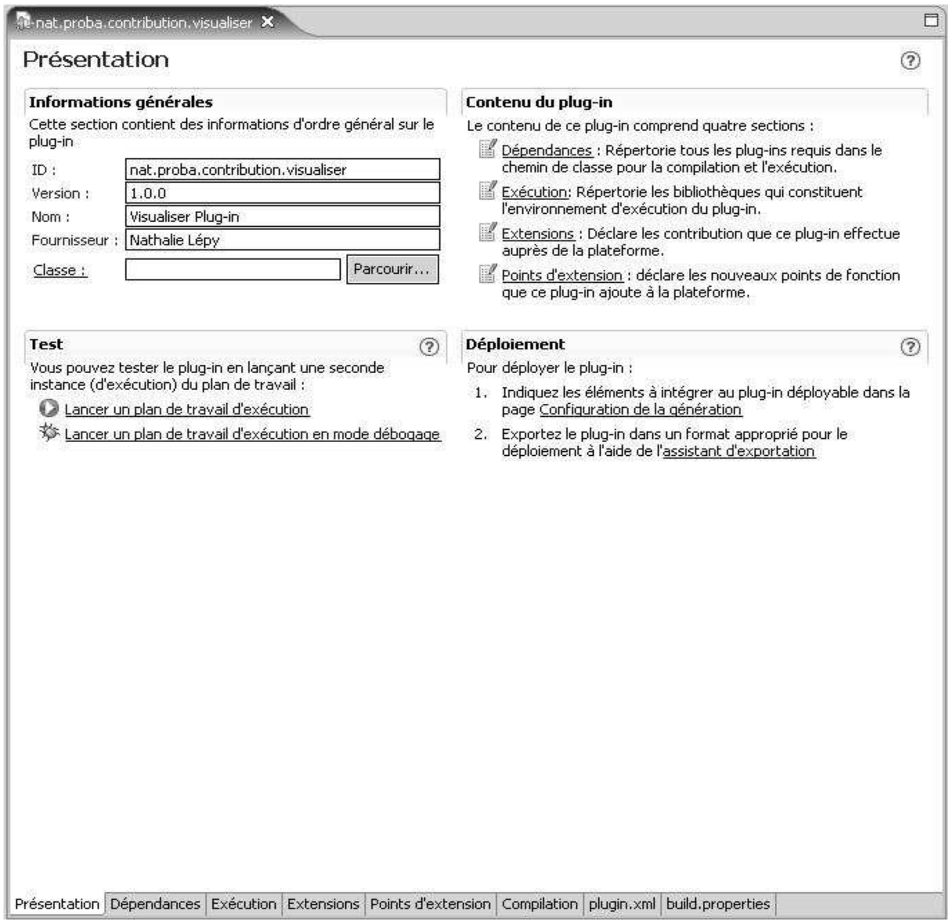

Figure 5 — *Illustration de l'éditeur de* plug-ins.

Si l'on est dans le cas de figure de création de *plug-in* sans assistant, que l'on crée un projet vide, les répertoires *src* et *bin* sont initialement vides. Le fichier manifeste, lui, contient quelques éléments. Pour visualiser le contenu de ce fichier, on fait appel à l'éditeur de *plug-ins* : il suffit de double-cliquer sur *plugin.xml*, pour qu'il s'ouvre automatiquement. La figure 5 présente cet éditeur particulier, qui comporte les onglets suivants :

- *Présentation* : contient une information synthétique sur le *plug-in* (nom, version, nom du fournisseur, etc.) ;
- *Dépendances* : indique les *plug-ins* requis pour le *plug-in*.
- *Exécution* : indique les bibliothèques requises pour exécuter le *plug-in* ;
- *Extensions* : indique les points d'extension utilisés par le *plug-in* ;
- *Points d'extensions* : indique les points d'extension fournis par le *plug-in* ;
- *Compilation* : permet de configurer la compilation (bibliothèques à utiliser et ordre de leur compilation, listes des fichiers à inclure dans les générations binaires et sources) ;
- *plugin.xml* : cette page propose un éditeur permettant d'éditer le code source de *plugin.xml* ;
- *build.properties* : affiche le contenu du fichier *build.properties.*

Toute modification réalisée dans l'un des onglets *Présentation, Dépendances, Exécution, Extensions, Points d'extensions,* provoque une mise à jour automatique de *plugin.xml*.

Notons enfin la présence du raccourci « *Lancer un plan de travail d'exécution* » dans la page *Présentation,* qui permet de lancer un deuxième exemplaire d'Eclipse, prenant en compte le *plug-in* en cours de création dans sa base des *plug-ins* disponibles. Cela permet de le tester.

3.3.2. *Fichier manifeste* plugin.xml

Le fichier manifeste définit l'apparence de la contribution ;
son comportement est déterminé par le code Java.
(Gamma et Beck, 2004)

Le fichier manifeste est le seul fichier indispensable. C'est lui qui va permettre la réalisation du chargement dynamique du *plug-in*. En effet, comme nous l'avons vu dans la section précédente, l'architecture modulaire du *plug-in* parvient à découpler la déclaration, contenue dans le fichier manifeste, de l'implémentation, contenue dans les fichiers Java. Si l'on emprunte l'analogie formulée par Gamma et Beck (2004), le fichier manifeste est la partie visible de l'iceberg et le code Java, sa partie immergée (voir figure 6). La lecture de ce fichier fournit donc à Eclipse les informations nécessaires pour mettre à jour son registre des *plug-ins* disponibles, sans nécessiter de charger le code. La figure 8, Annexe B, montre un exemple de fichier *plugins.xml*.

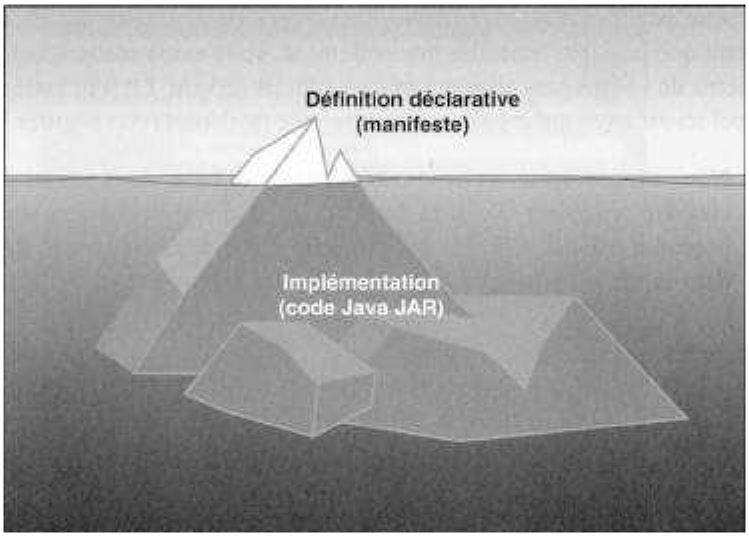

Figure 6 — Représentation imagée du découplage déclaration/implémentation du plug-in*.*
D'après Gamma et Beck (2004, figure 3.2).

Le fichier manifeste réalise également la déclaration des interconnections avec les autres *plug-ins.* Comme nous l'avons expliqué dans la section 3.1, un point d'extension réalise l'interface, l'articulation, entre un nouveau *plug-in* et un ancien *plug-in*. Chaque *plug-in* contient donc un ou plusieurs points d'extension qui peuvent apparaître de deux manières dans la définition du *plug-in* :

a) un *plug-in* peut contribuer à étendre les fonctionnalités d'un ou plusieurs *plug-in(s)* existant(s) via leurs points d'extension respectifs ;

b) un *plug-in* peut définir de nouveaux points d'extension, afin de permettre à d'autres *plug-ins* d'étendre à leur tour ses fonctionnalités.

La plus répandue est l'utilisation a). Dans leur livre, Gamma et Beck (2004) citent une remarque d'un de leur lecteur :

« Moi ce qui m'intéresse, c'est juste de savoir comment écrire des plug-ins. Je n'ai pas besoin de savoir comment déclarer de nouveaux points d'extension. Parmi plusieurs centaines de plug-ins déjà disponibles, presque aucun n'autorise à déclarer de nouveaux points d'extension ».

Pour que nous puissions tous capitaliser sur le travail existant, définir de nouveaux points d'extension est pourtant vital. Contribuer à Eclipse ne signifie pas qu'il faille ajouter des blocs fonctionnels monolithiques. Il s'agit plutôt de créer des opportunités d'extension, en apportant d'emblée certaines extensions. Cela dépasse néanmoins la perspective définie dans le cadre de ce probatoire, nous ne traiterons donc pas l'insertion de points d'extension dans ce rapport, mais nous les situons néanmoins dans l'architecture générale d'un *plug-in* (voir section 3.3.1) et dans la structure du fichier manifeste associé (voir figure 8, Annexe B).

Concrètement, un point d'extension permet de construire un *plug-in* à partir des exports d'un autre *plug-in.* Notons que les *plug-ins* ne peuvent utiliser que les classes exportées par d'autres *plug-ins*, les points d'extension sont donc à ce titre également importants. La figure 7 illustre les relations entre *plug-ins*, l'un contribuant à l'autre. Il est important de bien comprendre que lorsqu'on apporte une contribution, elle est ajoutée aux contributions existantes : **rien n'est prévu pour remplacer un bloc fonctionnel existant**.

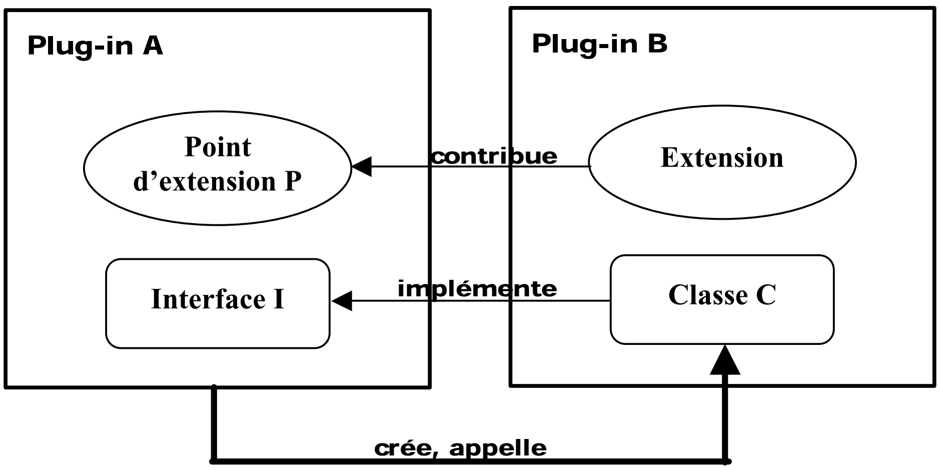

Figure 7 — Description des relations entre deux plug-ins.
D'après http://eclipse.org/eclipse/presentation/eclipse-slides_files/v3_document.htm
(transparent 14, documentation Eclipse, consultée le 4 juin 2005).

Dans notre exemple figure 7,

- le *plug-in* A déclare un point d'extension P et une Interface I, associée à P ;
- le *plug-in* B implémente l'interface I avec sa propre classe C, qui contribue au point d'extension P ;
- lors de l'exécution, le *plug-in* A instancie la classe C et appelle les méthodes de l'interface I définie dans la classe C.

3.3.3. *Utiliser le PDE*

L'environnement de développement de *plug-ins* (PDE — Plug-in Development Environment) permet de construire des *plug-ins* en utilisant les points d'extension standards disponibles. Il propose un ensemble d'assistants permettant de créer et initialiser divers types de projets de *plug-in*. Ces assistants créent une structure de répertoire de *plug-in* standard et ajoutent différents éléments selon l'assistant sélectionné. On peut ainsi obtenir des modèles pour construire les éléments suivants :

- de nouveaux boutons de barre d'outil et de nouveaux menus, ainsi que les jeux d'actions associés (celles qui seront déclenchées par un clic sur le bouton ou le menu) ;
- de nouveaux éditeurs simples ou multipages, comportant toutes les fonctions de base d'un éditeur, telles que les fonctions « couper », « copier », « coller », « rechercher », offrant la possibilité de mettre en évidence la syntaxe et prenant en charge le double-clic ;
- des menus en incrustation (menus qui ajoutent un sous-menu et une nouvelle action au menu en incrustation d'un objet cible, à définir) ;
- des pages de propriétés, à ajouter à des ressources ;
- des extensions à une perspective telles que des raccourcis de vue, un assistant, un jeu d'actions et des vues ;
- des vues de plan de travail, avec plusieurs options comprenant la prise en charge des menus en incrustation, la barre d'outils locale, le double clic, le tri et le filtrage.

Grâce au PDE, on peut très facilement créer le fichier manifeste *plugin.xml* du *plug-in*, indiquer les ressources nécessaires à l'exécution, définir les points d'extension, etc.

Dans le cadre de l'extension envisagée au cours de ce probatoire, nous souhaitions savoir si Eclipse permettait notamment de pouvoir intégrer l'utilisation d'un nouveau langage de programmation et de réaliser des opérations (effectuées par l'exécution d'un autre programme écrit en C) sur le résultat de l'exécution du code. Cette extension nécessiterait donc d'implémenter les fonctionnalités suivantes :

- un éditeur reconnaissant les fichiers avec notre nouvelle extension et permettant de saisir du code dans notre nouveau langage ;
- l'insertion d'un nouveau menu, comprenant deux items, dans la barre des menus et deux boutons dans la barre des tâches, éléments auxquels on souhaitait lier les actions suivantes :
  - 1er item et 1er bouton : compiler le programme édité, l'exécuter et générer un fichier *résultat* de l'exécution ;
  - 2ème item et 2ème bouton : faire afficher la représentation graphique des résultats contenus dans le fichier *résultat ;*
- la personnalisation ou la création d'une perspective qui permettrait l'affichage de ces nouveaux menus et boutons lorsque note nouvelle éditeur serait affiché.

En regardant les exemples de *plug-ins* développés pour d'autres langages (par exemple CDT), nous avons très rapidement obtenu la réponse à notre question : ces réalisations sont tout à fait envisageables. La question était alors d'essayer d'évaluer la complexité de la tâche. Au regard des divers assistants proposés par PDE, le développement de ces différentes fonctionnalités devrait être relativement aisé ; on peut même dire qu'Eclipse a été écrit pour permettre ce type de personnalisation. Nous disposons en effet des assistants nécessaires pour implémenter des éditeurs, des menus, des boutons et des perspectives. Il ne reste plus qu'à regrouper tous ces éléments au sein d'un même *plug-in* et à programmer le compilateur pour notre nouveau langage. Tout ceci ne pose pas de difficultés majeures. Les deux difficultés que l'on peut en revanche envisager concernent la prise en main du logiciel Eclipse (voir section 1.3.2, les points faibles) et, éventuellement, le temps nécessaire à l'apprentissage de la programmation Java.

Conclusion

Fruit du travail d'un consortium de grandes entreprises (Borland, IBM, …), Eclipse est un EDI qui est devenu, en peu de temps, extrêmement performant et populaire et qui se place en forte compétition avec NetBeans (SunOne Studio), JBuilder (Borland), et IntelliJ IDEA (JetBrains). La communauté des programmeurs Java en est effectivement rapidement venue à utiliser Eclipse, appréciant sa gratuité et le fait qu'il soit développé en *open source*, qu'il soit d'excellente qualité et totalement personnalisable.

Mais même s'il a été développé en Java, Eclipse est bien plus qu'un EDI Java : c'est une plate-forme de développement universelle permettant d'intégrer toutes sortes d'outils de développement. Nous nous sommes d'ailleurs attachée à présenter surtout les aspects génériques d'Eclipse, indépendamment de tout langage de programmation. Autrement dit, nous ne nous sommes pas placés dans la perspective des utilisateurs de la plate-forme, mais dans celle des contributeurs. Notre problématique, outre celle d'obtenir une vue d'ensemble sur le logiciel Eclipse, consistait à découvrir comment réaliser une extension permettant de faire d'Eclipse la plate-forme de développement pour un nouveau langage de programmation.

La grande force d'Eclipse réside dans l'ouverture de son noyau qui permet l'ajout de très nombreux *plug-ins*. Du point de vue de la plate-forme Eclipse, les *plug-ins* peuvent être vus comme des enseignants qui « apprennent » à Eclipse comment travailler avec des éléments aussi variés que des fichiers Java, du contenu web, des graphiques, de la vidéo, etc. Eclipse nous permet de développer indépendamment des outils qui s'intègrent aux outils d'autres personnes de manière si progressive qu'il est impossible de savoir où finit un outil et où commence un autre. La notion d'outil telle qu'on la pratiquait jusqu'alors, et telle qu'on l'entend dans tout EDI traditionnel, n'est plus adéquate.

Eclipse tire également son originalité du choix des composants graphiques utilisés lors de son développement : Eclipse ne contient pas d'AWT, ni de Swing, mais uniquement des composants SWT/JFace (composants développés par IBM). L'interface gagne ainsi en rapidité et convivialité.

D'un point de vue plus technique, cette étude nous a également permis d'évaluer la faisabilité du développement envisagé et d'en estimer la complexité. L'utilisation de l'environnement de développement de *plug-ins* PDE paraît être une bonne méthode pour pouvoir obtenir assez rapidement et facilement un outil reconnaissant et pouvant effectuer des traitements sur un nouveau langage de programmation. Avec PDE il est également possible de personnaliser l'environnement de développement associé à ce nouveau langage via la modification ou la création d'une perspective.

Références bibliographiques

Arthorne J., Laffra C. (2004). *Official Eclipse 3.0 FAQs*. Gamma E., Nackman L., Wiegand J. Eds. The Eclipse Series, Addison-Wesley.
☞ Extrait consultable sur : *http://eclipsefaq.org/chris/faq/faq-list.html*. Date de la dernière consultation : 29 mai 2005.

Bolour A. (2003). Notes on the Eclipse plug-in architecture. *http ://www.eclipse.org/Article-Plug-in-architecture/plugin_architecture.html*. Date de dernière consultation : 02 mai 2005.

Doudoux J.-M. (2004). Développons en Java avec Eclipse — Version O.50.1. *http://perso.wanadoo.fr/jm.doudoux /java/dejae/* ou *http://perso.wanadoo.fr/jm.doudoux /java/dejae/dejae_0_50.pdf*. Date de la dernière consultation : 24 mai 2005.

Eclipse community (2003). Eclipse platform technical overview. Object Technology International publication. *http ://www.eclipse.org/whitepapers/eclipse-overview.pdf*. Date de dernière consultation : 29 mai 2005.

Edgar N., Haaland K., Li J., Peter K. (2004). Eclipse User Interface Guidelines — Version 2.1. IBM Corporation. *http://www.eclipse.org/articles/Article-UI-Guidelines/Contents.html*. Date de dernière consultation : 6 juin 2005.

Gamma E., Beck K. (2004). *Eclipse: principes, patterns, and plug-ins*. CampusPress.

Girard D. (2001). Eclipse/WebSphere Studio Application Developer : Retour d'Expérience. *http://www.application-servers.com/articles/pdf/a19s.wsad-en.pdf*. Date de la dernière consultation : 29 mai 2005.

Girard D., Rafal O. (2005). Interview de Mike Milinkovich, directeur exécutif de la Fondation Eclipse. *Le Monde Informatique*, édition du 20 mai 2005, ou *http://www.application-servers.com/stories.do?reqCode=wholeStory&sid=2005-05-19-21:35:54*. Date de la dernière consultation : 2 juin 2005.

Holzner S. (2004). *Eclipse*. Paris : Éditions O'Reilly.

OSI (2005). The Open Source Definition — Version 1.9. *http://opensource.org/docs/definition.php*. Date de la dernière consultation : 30 mai 2005.

Perens B. (1999). The Open Source Definition. In *DiBona C., Ockman S., Stone M. (Eds), Open Sources: Voices from the Open Source Revolution*. O'Reilly.

Poulet (2005). Installation d'un IDE intégrant MinGW, Eclipse, CDT et wxWidgets sous Windows. *http://www.firecortex.com/doc/*. Date de la dernière consultation : 29 mai 2005.

Rajorshi B. (2004a). The Eclipse IDE: Part 1. *Linux For You*, August 2004. http://freehosting.hostrave.com/p/rajorshi/article_eclipse1.htm. Date de la dernière consultation : 4 juin 2005.

Rajorshi B. (2004b). The Eclipse IDE: Part 2. *Linux For You*, August 2004. http://freehosting.hostrave.com/p/rajorshi/article_eclipse2.htm. Date de la dernière consultation : 4 juin 2005.

Stallman R. M. (2002). Why "Free Software" is Better Than "Open Source". In *Free Software, Free Society: Selected Essays of Richard M. Stallman*, Joshua Gay Editor.

Traduction française sur http://www.gnu.org/philosophy/free-software-for-freedom.fr.html. Date de la dernière consultation : 2 juin 2005.

Taft D. K. (2005). Eclipse: Behind the name. *eWeek*, May 20, 2005. http://www.eweek.com/article2/0,1759,1818233,00.asp. Date de la dernière consultation : 4 juin 2005.

Viseur (2002). Fiche 132 : La dynamique open source. In *Elaboration de la stratégie commerciale de la société Capflow*. Travail de fin d'études en management de l'innovation. *http://www.ecocentric.be/res/fiche132.pdf*. Date de la dernière consultation : 1er juin 2005.

Yamagata H. (1997). Le pragmatisme du logiciel libre : entretien avec Linus Torvalds Linux France, *http://www.linux-france.org/article/these/interview/torvalds/pragmatist-fr.html*. Date de la dernière consultation : 1er juin 2005.

Annexes

Annexe A. Bibliographie et webographie thématique

Livres écrits sur Eclipse

Arthone et Laffra (2004) ont recensé un certain nombre de livres anglais sur Eclipse. Nous avons ajouté à cette liste quelques livres parus depuis ou à paraître.

- Arthorne J., Laffra C. (2004). *Official Eclipse 3.0 FAQs*. Gamma E., Nackman L., Wiegand J. Eds. The Eclipse Series, Addison-Wesley.
  ☞ Extraits consultables sur : http://eclipsefaq.org/chris/faq/faq-list.html
- Budinsky F., Steinberg D., Ellersick R. (2004). *Eclipse modeling framework*. Gamma E., Nackman L., Wiegand J. Eds. The Eclipse Series, Addison-Wesley.
- Burd B. (2004). *Eclipse for dummies*. Wiley Publishing.
- Carlson D. (2005). *Eclipse distilled*. Gamma E., Nackman L., Wiegand J. Eds. The Eclipse Series, Addison-Wesley.
- Colyer A., Clement A., Harley G., Webster M. (2005). *Eclipse AspectJ: Aspect-oriented programming with AspectJ and the Eclipse AspectJ development tools.* Gamma E., Nackman L., Wiegand J. Eds. The Eclipse Series, Addison-Wesley.
- Chaber E. (2004). *Les JSP – Avec Struts, Eclipse et Tomcat*. Dunod.
- Clayberg E., Dan Rubel (2004). *Eclipse: Building commercial-quality plug-ins*. Gamma E., Nackman L., Wiegand J. Eds. The Eclipse Series, Addison-Wesley.
- D'Anjou J., Fairbrother S., Kehn D., Kellerman J., McCarthy P. (2004). *Java™ Developer's Guide to Eclipse*. Addison-Wesley. 2nd Edition.
- Daum B. (2003). *Eclipse 2 for Java developers.* Wiley.
- Daum B. (2004). *Professional Eclipse 3 for Java developers.* Wiley.
- Daum B., Raimond c. (2005). *Eclipse 3 pour les développeurs Java : Développez des plug-in et des applications.* Dunod.
- Djaafar K., Salvatori O. (2005). *Eclipse et JBoss : Développement d'applications J2EE professionnelles, de la conception au déploiement*. Eyrolles.
- Djaafar K. (2003). *Développement J2EE avec Eclipse et WSAD*. Eyrolles.
- Dudney B. (2003). *Eclipse live.* SourceBeat.
- Gallardo D., Burnette E., McGovern R. (2003). *Eclipse in action: A Guide for the Java developer*. Manning.
- Gamma E., Beck K. (2003). *Contributing to Eclipse: Principles, patterns, and plug-ins*. Gamma E., Nackman L., Wiegand J. Eds. The Eclipse Series, Addison-Wesley.
- Gamma E., Beck K. (2004). *Eclipse: principes, patterns et plug-ins*. CampusPress.
- Holzner S. (2004). *Eclipse: A Java developer's guide.* O'Reilly.
- Holzner S. (2004). *Eclipse Cookbook*. O'Reilly.
- Holzner S. (2004). *Eclipse*. Paris : Éditions O'Reilly.
- Judd C., Shittu H. (2005). *Pro Eclipse Jst: Plug-ins For J2ee development*. Apress.

- Li Guojie J. (2005). *Professional Java Native Interfaces with SWT/JFace (Programmer to Programmer)*. Wrox.
- Li J., Edgar N., Haaland K., Peter K. (2006). *Eclipse User Interface Guidelines*. Addison-Wesley.
- McCarthy P. (2005). *The Java Developer's Guide to Eclipse*. US original AW, Addison-Wesley.
- Moore W., Dean D., Gerber A., Wagenknecht G., Vanderheyden P. (2004). *Eclipse development using the Graphical Editing Framework and the Eclipse Modeling Framework.* IBM RedBooks, http://www.redbooks.ibm.com/abstracts/sg246302.html?Open.
- Ng S., Holder S., Scarpino M. (2003). *JFace/SWT in action.* Manning.
- Northover S, Wilson M. (2004). *SWT: The Standard Widget Toolkit*. Gamma E., Nackman L., Wiegand J. Eds. The Eclipse Series, Addison-Wesley.
- Pluta J. (2003). *Eclipse: Step by step.* MC Press. 2003.
- Scarpino M., Holder S., Ng S. (2004). *SWT/JFace in Action*: *GUI design with Eclipse 3.0.* Manning Publications.
- Shavor S., D'Anjou J., Fairbrother S., Kehn D., Kellerman J., McCarthy P. (2003). *The Java developer's guide to Eclipse*. Addison-Wesley.
- Valcarcel C. (2004). *Eclipse kick start*. Sams Publishing.
- Warner R., Harris R. (2004). *The definitive guide to SWT and JFace*. Apress.
- Zeller A., Krinke J. (2005). *Essential open source toolset: Programming with Eclipse, JUnit, CVS, Bugzilla, Ant, Tcl/Tk and more*.

Notons que pour la plupart, ces livres sont fortement orientés développement Java (même ceux pour lesquels le mot « Java » n'apparaît pas explicitement dans le titre).

Sites Internet

L'ensemble des liens Internet proposés dans la liste ci-dessous était valide au 29 mai 2005.

Sites dédiés

http://www.eclipse.org/
: Le site Internet de la fondation Eclipse

http://www.eclipsecon.org/

http://www.improve-technologies.com/pages/Java/IDE/Eclipse/

http://www.eclipsetotale.com/
: Un site francophone consacré au projet Eclipse et aux outils Websphere Studio d'IBM News. Actualité, articles, tutoriaux...

http://eclipsewiki.editme.com/
: Ce site "wiki" sur Eclipse a pour vocation d'être enrichi par ses visiteurs. Les thèmes actuellement abordés sont : plate-forme Eclipse, JDT, SWT/JFace, CDT, GEF, News, PDE, ANT, problèmes repertoriés, JUnit, CVS, ...

http://www.eclipse-workbench.com/
: Un site anglophone sur Eclipse, maintenu par une SSII hollandaise.

http://mmoebius.gmxhome.de/eclipse/eclipse.htm
: Liste des Howto, patchs, liens et FAQ sur Eclipse et sa communauté

http://perso.wanadoo.fr/jm.doudoux/java/dejae

http://www.mobilefish.com/developer/eclipse

http://www.cs.umanitoba.ca/~eclipse

Généralités sur Eclipse

Eclipse platform technical overview, by OTI members:
http://www.eclipse.org/whitepapers/eclipse-overview.pdf

Eclipse project FAQ: http://www.eclipse.org/eclipse/faq/eclipse-faq.html

Official Eclipse 3.0 FAQs, by John Arthorne and Chris Laffra (IBM):
http://eclipsefaq.org/chris/faq/faq-list.html

Création de l'Eclipse Foundation, par Pierre Tramo :
http://linuxfr.org/2004/02/05/15356.html

Eclipse/WebSphere Studio Application Developer : Retour d'Expérience, par Didier Gérard (Improve Technologies) :
http://www.application-servers.com/articles/pdf/a19s.wsad.pdf

Développons en Java avec Eclipse par Jean-Michel Doudoux :
http://perso.wanadoo.fr/jm.doudoux/java/dejae/
ou http://perso.wanadoo.fr/jm.doudoux/java/dejae/dejae_0_50.pdf

Tutoriaux pour Eclipse 3.0 :
http://www.developpez.net/forums/viewtopic.php?t=219712&sid=2e764df3ab2968e824127f2e03fba44d

Cours et Tutoriaux Java : http://java.developpez.com/cours/#eclipse

What is Eclipse and how do I use it?, by Marc R. Erikson and Angus McIntyre (IBM):
http://www-106.ibm.com/developerworks/opensource/library/os-eclipse.html

Rappel utilisation d'eclipse : http://wiki.essi.fr/pub/Trash/WebHome/eclipse.html

Tout savoir sur Eclipse : http://www.smile.fr/content/smile/technologie/eclipse.htm

Par ailleurs, un certain nombre d'articles techniques sur Eclipse sont disponibles sur :
http://www.eclipse.org/articles/index.html

Développement de plug-ins

Eclipse plugins exposed, part 1: Simple GUI Elements, by Emmanuel Proulx:
http://www.onjava.com/pub/a/onjava/2005/03/30/eclipse.html?page=1
et http://www.onjava.com/pub/a/onjava/2005/03/30/eclipse.html?page=2

Eclipse plugins exposed, part 2: A first glimpse, by Emmanuel Proulx:
http://www.onjava.com/pub/a/onjava/2005/02/09/eclipse.html
et http://www.onjava.com/pub/a/onjava/2005/02/09/eclipse.html?page=2

Developping Eclipse plug-ins — How to create, debug, and install your plug-in, by David Gallardo: http://www-128.ibm.com/developerworks/opensource/library/os-ecplug/index.html

PDE does plug-ins, by Wassim Melhem and Dejn Glozic (IBM):
http://www.eclipse.org/articles/ArticlePDE-does-plugins/PDE-intro.html

Développez vos propres plug-ins *pour Eclipse* par Molière J. :
http://jmoliere.developpez.com/tutoriel/java/eclipse/plugin/

Quelques pointeurs pratiques

Site de téléchargements divers : http://www.eclipse.org/downloads

Sites de téléchargement de *plug-ins* :
http://www.eclipse-plugins.info,
http://www.yoxos.com
http://www.eclipse.lu/jsp/resources.jsp
http://www.crionics.com/products/opensource/eclipse/eclipse.jsp

Règles de création d'interfaces utilisateur d'Eclipse : http://www.eclipse.org/articles/Articles-UI-Guidelines/index.html

Installer Eclipse : http://www.eclipsetotale.com/articles/installation.html

Getting started with the Eclipse platform, by David Gallardo (independent software consultant): http://www-106.ibm.com/developerworks/java/library/os-ecov/?t=egrL296&p=starteclipseplat

Creating a text-based editor for Eclipse, by Elwin Ho (HP): http://devresource.hp.com/drc/technical_white_papers/eclipeditor/EclipseEditor.pdf

Installing Eclipse: http://www.mobilefish.com/developer/eclipse/eclipse.html

Connecting Eclipse to a CVS Repository: http://www.mobilefish.com/developer/eclipse/eclipse_quickguide_cvsrepository.html

Projet CDT

CDT — C/C++ Development Tools: http://www.eclipse.org/cdt/

CDT FAQ, sur Eclipse : http://dev.eclipse.org/viewcvs/index.cgi/%7Echeckout%7E/cdt-home/user/faq.html?cvsroot=Tools_Project (FAQ officielle)

CDT Faq, sur Eclipsewiki : http://eclipsewiki.editme.com/CDTFaq

Eclipse accueil désormais les projets C/C++, par Sébastien Marineau (QNX) et Doug Schaefer (IBM) : http://www.electronique.biz/editorial/245168/environnement-de-developpement/eclipse-accueille-desormais-les-projets-en-c/c++/

Eclipse project CDT (C/C++) plugin tutorial, by Brian Lee (University of Manitoba): http://www.cs.umanitoba.ca/~eclipse/7-EclipseCDT.pdf

C/C++ development with the Eclipse Platform — How to use the C/C++ Development Toolkit (CDT), by Pawel Leszel (independent software consultant): http://www-106.ibm.com/developerworks/opensource/library/os-ecc/

Installation d'un IDE intégrant MinGW, Eclipse, CDT et wxWidgets sous Windows, par Frédéric Poulet (Firecortex) : http://www.firecortex.com/doc/

Installation d'un IDE intégrant Eclipse, CDT et wxWidgets sous Linux, par Frédéric Poulet (Firecortex) : http://www.firecortex.com/doc/

Installing CDT, by Mike Ludé: http://eclipsewiki.editme.com/InstallingCDT

Installing (CDT) C/C++ plugin: http://www.mobilefish.com/developer/eclipse/eclipse_quickguide_cdtplugin.html

Setting up Eclipse CDT on Windows, Linux/Unix, Mac OS X, by Max Berger: http://met.dnsalias.net:1111/teaching/cdt/cdt.pdf ou http://met.dnsalias.net:1111/teaching/cdt/index.jsp

Taking Eclipse to C/C++ — It's not just for Java any more, by Alan Zeichick: http://www.devx.com/DevDemos/Article/27641/2046

Refactoring

Refactoring with Eclipse using the new open source IDE for improving working code, by Daniel H. Steinberg (Java Offering): http://www-106.ibm.com/developerworks/library/l-eclipse.html

Annexe B. Exemple de fichier manifeste

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.0"?>
<plugin
   id="nat.proba.contribution.visualiser"
   name="Plug-in Visualiser"
   version="1.0.0"
   provider-name="Nathalie Lépy">

   <requires>
       <import plugin="org.eclipse.ui"/>
   </requires>

   <runtime>
      <library name="visualiser.jar">
         <export name="*"/>
      </library>
   </runtime>

     <extension point="org.eclipse.ui.actionSets">
       <actionSet
         id="nat.proba.contribution.visualiser.actionSet"
         label="actionSet visualiser">
       <action
          id="nat.proba.contribution.visualiser.ActionVisualiser"
          label="action visualiser"
          icon="icons/bel_oeil_rétréci.jpg"
         toolbarPath="GroupeVisualiser"
          class="nat.proba.contribution.visualiser.
                 visualiser.ActionVisualiser">
         </action>
         <action
          id="nat.proba.contribution.visualiser.ActionTraiter"
          label="action traiter"
          icon="icons/avatar_rétréci.gif"
          toolbarPath="GroupeVisualiser"
          class="nat.proba.contribution.visualiser.
                 traiter.ActionTraiter">
         </action>
       </actionSet>
     </extension>

     <extension-point
       name="Parou Entrer"
       id="nat.proba.contribution.visualiser.parouEntrer"/>

</plugin>
```

Identification du *plug-in*

Les autres *plug-in* nécessaires

Localisation du code du *plug-in*

Déclaration des contributions réalisées par ce *plug-in*

Déclaration d'un nouveau point d'extension ouvert aux contributions d'autres *plug-ins*

Figure 8 — Exemple de fichier manifeste d'un plug-in (plugin.xml).

Cet exemple contient les déclarations d'un *plug-in* qui ajoute deux boutons à la barre d'outil et leur associe à chacun une action.

La partie `<extension-point ... />` est donnée ici pour les besoins de l'illustration mais est complètement inutile dans le cadre de ce développement.

Glossaire de termes et acronymes

Si deux personnes utilisent deux termes différents pour désigner une même chose ou si elles utilisent le même terme pour faire référence à deux choses différentes, cela peut prêter à confusion et conduire à l'incompréhension. Il est donc important d'utiliser le terme correct pour faire référence aux différents éléments utiles dans les commentaires, le code et la documentation (Eclipse Foundation[20]). Pour ce faire, nous proposons ci-dessous une définition des termes et une traduction des sigles utilisés dans ce rapport et/ou importants pour la compréhension de la documentation, en français ou en anglais, écrite sur Eclipse.

AJDT	*AspectJ Development Tools Project*
API	*Application Programming Interface*
AspectJ	Langage orienté aspect, extension de Java™
AWT	*Abstract Windowing Toolkit* Première occurrence de boite à outils Java pour les GUI (*Graphical User Interfaces*)
BIRT	*Business Intelligence and Reporting Tools project* Système *open source* de production de rapport basé sur Eclipse qui est intégré à l'application pour produire des rapports contraints, à la fois pour le web et au format pdf
CDT	*C/C++ Development Tool*
CME	*Concern Manipulation Environment*
CPL	*Common Public Licence*
CVS	*Concurrent Versions System* Outil libre de gestion des versions et du partage de fichiers.
ECECIS	*Eclipse Community Education project*
ECF	*Eclipse Communications Framework*
Eclipse	Eclipse est une plate-forme de développement unvierselle
Eclipse platform	Structures et services centraux sur lesquels l'extension de *plug-ins* est créée. Elle est composée du noyau de la plate-forme (*core* ou *platform core*) et de l'interface utilisateur (le *workbench)*
EDI	Environnement de développement intégré L'EDI est une interface qui permet de développer, compiler et exécuter un programme dans un langage donné.
EMF	*Eclipse Modeling Framework*
EPL	*Eclipse Public Licence*
eRCP	*Embedded Rich Client Platform*
Exécutable	Voir *runtime*
GEF	*Graphical Editing Framework*
GMF	*Graphical Modeling Framework*
GMT	*Generative Model Transformer*
GUI	*Graphical User Interface*
IDE	*Integrated Development Environment* (voir EDI)
JDT	*Java Development Tool*
JFace	Bibliothèque de classes permettant de piloter beaucoup des tâches de programmation de l'interface utilisateur

20 http://www.eclipse.org/glossary.html

JST	*J2EE Standard Tool*
HTML	*HyperText Markup Language*
MinGW	*Minimalist GNU for Windows*
OMELET	*Open Modeling Environment with Links for Extensions and Transformations*
Open source	Ce dit d'un logiciel dont le code est mis à disposition pour d'éventuelles modifications/évolutions selon les besoins des programmeurs l'utilisant et qui peut être redistribué. Mais *Open Source* implique plus que la simple diffusion du code source, pour une définition plus précise des critères, voir : Perens (1999) et http ://opensource.org/docs/definition.php
OSI	*Open Source Initiative* Organisation, à but non lucratif, qui gère la campagne de l'*open source* et sa marque de certification.
OTI	*Object Technology International :* Filiale d'IBM qui initialement a développé Eclipse
PDE	*Plug-in Development Environment*
Plug-in	Extension ou module que l'on peut brancher sur un point d'entrée particulier d'une application, ici l'application Eclipse
PTP	*Parralel Tools Plateform*
RCP	*Rich Client Platform*
Refactoring	Propre au JDT, le *refactoring* regroupe l'ensemble des opérations permettant de transformer du code sans modifier sa logique
Runtime	Composant qui gère l'exécution des outils logiciels constituant Eclipse et qui charge les *plug-ins* nécessaires au bon moment
Swing	Bibliothèque Java, complémentaire du kit AWT, permettant de réaliser une implémentation non native de contrôles sophistiqués tels que des arborescences, des tables et du texte.
SDK	*Standard Development Kit* Contient la plate-forme Eclipse, le JDT et le PDE
SWT	*Standard Widget Toolkit* Boite à outils d'objets élémentaires (*widgets*) pour le développeur Java, composée d'un API portable et d'une implémentation native.
TPTP	*Eclipse Test & Performance Tools Platform Project* *Plate-forme de développement ouverte fournissant des structures et services pour des outils de test et de performance*
VCM	*Versioning and Configuration Management*
VE	*Visual Editor*
VTP	*Voice Tools Platform*
Workbench	Plan de travail constituant l'interface utilisateur d'Eclipse
Workspace	Espace de travail, contenant l'ensemble des fichiers utiles pour le projet Eclipse en cours de développement
WSAD	*WebSphere Studio Application Developer* WSAD est un IDE, développé par IBM, qui permet de concevoir, construire, tester et déployer des services Web, des portails et des applications J2EE.
WST	*Web Standard Tool*
WTP	*Eclipse WebTool Plateform project*
UI	*User Interface*
URL	*Uniform Resource Locator*
XML	*eXtensible Markup Language*

Index

RÉSUMÉ

Étude de l'environnement de développement intégré ouvert Eclipse dans l'optique d'une extension

Ce rapport présente l'Environnement de Développement Intégré (EDI) Eclipse, plate-forme universelle de développement ouvert permettant d'intégrer toutes sortes d'outils de développement. Nous décrivons la philosophie Eclipse et l'architecture particulière de cette plate-forme de développement, nous expliquons notamment comment Eclipse évolue grâce aux contributions des développeurs de *plug-ins*.

Dans cette étude, le point de vue adopté est celui d'un développeur d'outils logiciels souhaitant intégrer à Eclipse un nouveau langage et des outils associés. Nous montrons que Eclipse s'avère être un environnement performant, permettant d'obtenir assez rapidement et facilement un outil reconnaissant un nouveau langage de programmation et pouvant compiler, exécuter et effectuer des traitements sur ce langage.

Mots-clés : Eclipse, EDI, *open source, plug-ins,* extension.

ABSTRACT

Study of the open Eclipse integrated development environment to perform an extension

In this report, the Integrated Development Environment (IDE) Eclipse is presented. Eclipse is a universal open development platform integrating many kinds of development tools. We describe the Eclipse philosophy and the specific architecture of this development platform. In particular, we explain how this platform can upgrade thanks to the contributions of plug-ins developers.

In this study, the point of view of a software developer was adopted, aiming at integrating a new language and its associated tools into Eclipse. We show that Eclipse is an efficient environment allowing to perform its task quite rapidly and easily: it provides a tool which is able to recognize a new programming language and to parse, run and carry out several processing tasks on this language.

Keywords: Eclipse, IDE, open source, plug-ins*,* extension.