

M Alexis CAUVE

Informatique Signaux et Systèmes de Sophia Antipolis
Conservatoire Nationale des Arts et Métiers
Diplôme d'Etudes Supérieures Techniques

JavWebInspector

Extension de JavInspector en application Web

Encadrants : Pierre CRESCENZO
Michel GAUTERO
Philippe LAHIRE

Intervenants : Richard GRIN

SOMMAIRE

Sommaire 2

Introduction 3

I Présentation du projet	4
II Analyse comparative des technologies Internet	6
III Présentation de la technologie J2EE	9
IV Conception architecturale client serveur de JavWebInspector	12
V Conception finale et refactoring de la couche présentation	13
VI Procédure d'installation du serveur J2EE et de déploiement	15
VII A l'attention de mes successeurs	16
VIII Glossaire	17
IX Bibliographie et ressources Web	18

Introduction

A la réception et à l'acceptation de ce projet, je ne m'attendais pas à effectuer un tel parcours de connaissances, lequel m'amena à étudier de nombreux sujets, avec en premier lieu J2EE, les Applets, RMI, la Sécurité, les sockets, les Servlets et leurs moteurs, JSP, EJB, les serveurs d'applications et (Apache, TomCat), mais aussi des comparaisons avec d'autres technologies, ASP, PHP, XML, les scripts CGI...

Ce chemin m'a également mené sur la recherche d'un pattern spécialisé pour l'initialisation et à l'exécution des applications. Je n'ai pas vraiment trouvé d'écrits intéressants à ce sujet ; bien que le design pattern, l'architecture logiciel et le design up donne lieu à des articles et des livres, il semble que les bonnes pratiques de conception logicielle restent de l'ordre du secret industriel. J'ai cependant eu le plaisir de constater un travail particulier sur ce thème effectué par l'organisation Jakarta Project (<http://jakarta.apache.org>) avec le framework MVC de Struts et le framework Avalon. Jakarta est leader dans de nombreux projet « Open Source ».

Un autre chemin particulier, de ce projet m'a initié au travail du « refactoring », qui nécessite de découvrir une méthodologie appropriée. Le « refactoring » me rappelle des parties d'échecs, stimulantes et frustrantes par alternance.

Ce parcours a donné lieu à une connaissance livresque importante que je ne peux retranscrire dans ce rapport. L'étude de faisabilité et le refactoring de la couche présentation ont été des phases importantes par rapport au peu de code nécessaire pour aboutir. Je regrette de ne pas avoir réellement réussi à traverser toutes les étapes du cycle de vie du projet ; j'entends ainsi, que la phase de test unitaire à été escamotée, bien que l'application fonctionne « en apparence ».

Cette application, encore juvénile, présente le cœur d'une application au potentiel beaucoup plus vaste, et par laquelle nous pouvons aisément imaginer un outil d'analyse de sémantique, refactoring, d'identification de pattern...

J'ai essayé, sans succès (faute de temps), d'offrir une architecture J2EE à cette application (EJB, connections JDBC...), afin qu'autrui puisse librement libérer ce potentiel.

C'est avec plaisir que j'apprends aujourd'hui que cette application est confiée à d'autres étudiants.

Bonne chance à vous les gars.

Je tiens à remercier, pour son soutien et ses conseils, dans le domaine de la sécurité, M Richard GRIN.

I Présentation du projet

« Le projet JavWebInspector consiste en l'extension d'une application autonome, JavInspector, vers un mode de déploiement Internet. Au final, cette nouvelle version doit permettre à tout utilisateur de l'utiliser sur son poste, sans avoir à se soucier du navigateur Web utilisé, ou de l'installation du JDK, ni du « Plug-in » JRE, ou de tout autre problème de configuration. Le projet consistera en l'étude des possibilités existantes (langage, matériel, serveur...), donc de la faisabilité, puis en la réalisation de la nouvelle version, si possible.... Les outils utilisés doivent être gratuits, ou open source » ; voilà en quelques lignes « l'abstract » de ce projet, extrait d'une discussion.

JavInspector est une application JAVA autonome, développée par quatre stagiaires en Maîtrise à l'Université de Nice (<http://www.crescenzo.nom.fr/logiciels.html>) permettant d'effectuer une analyse de fichiers « bytecode », c'est à dire portant l'extension « .class » en s'appuyant sur le mécanisme de « reflection » (introspection en français) et les API associées, préexistantes dans le SDK du JDK 1.3 (Standard Development Kit du Java Development Kit). JavInspector fournit, des informations sur la structure de l'arbre d'héritage des classes et interfaces, les méthodes et les attributs.... et donne un rendu de ces informations par une IHM complexe. Cette application permet également aux utilisateurs de créer puis ajouter des « plug-in » destinés à une analyse plus personnelle du code pour les fichiers « .java ». Cette fonctionnalité n'ayant pas été complètement achevée pour la version JavInspector elle se retrouve escamotée pour la version JavWebInspector.

Afin de répondre à cette demande, une part importante de l'effort de réalisation a été dévolue à un état de l'art sur les techniques de déploiement Web, les architectures et les langages. Suite à cette recherche et conjointement à la prise en compte du langage par lequel fut développé JavInspector, une prise de décision a donné raison à l'utilisation de J2EE.

Une deuxième phase de l'effort est consacrée à l'analyse du code, pour une prise en main, et à un consistant travail de « refactoring » afin d'obtenir une architecture en couche du logiciel, et plus particulièrement pour l'IHM, nous habilitant à développer une couche présentation déployée sous forme d'applet dans un navigateur Web, aussi bien que en mode application dans une fenêtre classique.

Enfin l'implémentation de l'Applet, son déploiement et la gestion des droits de sécurité avant de passer à l'étape finale, le déploiement à partir d'un serveur Apache (fourni avec le framework J2EE SDK), formeront la phase finale.

L'application résultante JavWebInspector est implémentée en JAVA et conserve les fonctionnalités de JavInspector. Elle apporte la nouveauté d'être empaquetée dans un fichier WAR spécifique de J2EE, ce qui permet d'accéder aux fonctionnalités de distribution sur Internet par un déploiement automatique (autodeploy system) par le biais du serveur Apache J2EE. La couche présentation sur Internet consiste en une

Applet exécutée dans un Navigateur Web, Internet Explorer ou Netscape Navigator qui ont été les seuls navigateurs à être testés.

L'architecture J2EE n'est pas multi tiers, le client est un client lourd qui se retrouve porteur de l'ensemble du logiciel. Même si cela n'était ni l'origine des spécifications du projet ni nécessaire au fonctionnement du logiciel, j'aurais souhaité avoir le temps de développer la couche logique métier (dans notre cas l'analyseur de « classifieurs »), afin de préparer l'architecture à la réception de nouvelles fonctionnalités éventuellement produites à l'avenir par des étudiants.

L'application est également empaquetée dans un fichier JAR destiné à un fonctionnement local aussi bien sur une fenêtre classique (lancement par un fichier bat) que sur une Applet dans un navigateur Web (lancement par une page HTML).

II Analyse comparative des technologies Internet

Le déploiement de l'application sur Internet, comme sa transformation en mode client serveur relève les questions :

- Quelles parties de l'application doivent se trouver constamment présentes sur le client et/ou sur le serveur ?
- Quelles informations vont transiter du client au serveur ?

Le développement de « plug-ins » par l'utilisateur est-il possible ? Et si oui, ces « plug-ins » doivent-ils être rapatriés par le serveur et stockés dans un but quelconque ? Quelles sont les responsabilités, et pour chacune, qui du client ou du serveur en est le porteur ? Quelle sera l'évolution du logiciel ?

Bien sûr, ces questions et leurs réponses sont intimement liées à la technologie utilisée dans le projet, puisque cette dernière doit être en accord avec les spécifications prévues à ce jour et prévisibles à moyen terme.

1/ Analyse comparative des technologies

Au regard de l'importance que peut adopter l'extension des fonctionnalités du JavWebInspector, tel que l'élaboration de diagrammes UML, le développement réparti en groupes coordonnés par le serveur, l'analyse fine du code..., il sera préférable d'établir l'architecture la plus souple afin d'offrir aux informaticiens l'accès à un tel travail sans soucis de restructuration du présent projet.

Parmi les technologies Internet nous avons principalement :

- Le langage ASP ;
- Le langage .net et la plateforme;
- le langage PHP ;
- le langage JAVA associé au « framework » J2EE (Java 2 Enterprise Edition)
- les script CGI : Perl, Python, C et JavaScript

ASP et « .net » :

Ces technologies ne seront pas étudiées, car relevant de la propriété, elles se trouvent éliminées de ce choix.

PHP :

- Origine : Rasmus Lerdorf en 1994 développait PHP Personal Home Page à des fins personnelles pour garder des traces des visiteurs qui consultait son CV, sur son site Web. Ensuite utilisé dans un objectif professionnel, le langage s'est répandu auprès de ses collègues, puis dans le monde informatique. Aujourd'hui ce sigle désigne Hypertexte Preprocesseur.
- Nature : c'est un langage de script, en non pas un vrai langage de programmation, puisque le script n'est invoqué que par déclenchement d'un événement ; il n'appartient pas à la norme CGI. PHP réunit à la fois le langage et un logiciel libre interpréteur.
- Portabilité : on trouve différentes versions en fonction des plates-formes. De plus elle est réduite du fait des extensions optionnelles et de ses nombreux paramètres de génération.

- Liberté : « open source »
- Version : la dernière version est la numéro 4.
- Exécution : ne fonctionne pas en mode local, il nécessite un serveur (contrairement à Java Script)
- Normalisation : on ne trouve pas de norme pour ce langage
- Potentialité : traitement de fichier et bases de données sur le serveur ou tourne le script ; manipulation et génération d'images, accès aux variables d'environnement du serveur, réalisation de pages Web dynamiques, gestion de sessions.

JavaScript à ne pas confondre avec JScript de Microsoft:

- Portabilité : disponible sur tous les navigateurs modernes, avec peu de variantes, portabilité satisfaisante.
- Liberté : gratuit
- Execution : s'exécute chez le client, il existe cependant une version coté serveur
- Normalisation : norme ECMA
- Potentialité : très bien adapté pour réagir en temps réel aux manipulations de l'utilisateur, et aux évènements de la souris et du clavier (IHM) ; exploitation de page DHTML, gestion de fenêtres. Elle ne possède que 200 fonctions, et est de ce fait handicapée pour la création de page Web dynamiques

JAVA :

- Origine : inventé en 1990 par James Gosling, ingénieur chez Sun, afin de produire des programmes destinés à la domotique, plus souples que ceux existant en C++. L'objectif initial de ce langage, alors nommé « Oak », ne sera pas atteint, car le marché se tourna alors vers le World Wide Web, et non vers la domotique. Cependant les avantages offerts par ce langage lui permirent de se placer sur le marché Internet, et de devenir JAVA avec une évolution fortuite : l'exécution de programme en toute sécurité dans des pages Web.
- Nature : langage de programmation objets ;
- Portabilité : avec la Java Virtual Machine, la portabilité est totale (sauf exception) ;
- Liberté : le langage et les API sont gratuits, de nombreux Open Sources existent ;
- Version : 1.4.2
- Exécution : fonctionne en mode local, et sur les navigateurs Web, à la condition d'avoir préalablement installé les plug-in pour les Navigateur et la JRE en mode local.
- Potentiel : la réputation de JAVA n'est plus à faire, et son évolution vers la robotique et la médiatisation suit de près les marchés industriels.

REM : Un des grands intérêts de JAVA réside dans le package J2EE qui est une plateforme pour la programmation en entreprise, ce qui inclut, les applications distribuées, locales, client serveur réseau ou Internet....J2EE est à la fois un ensemble d'API pour le développement mais comporte également les outils de déploiement, les serveurs appropriés et une architecture normalisée.

Au regard des critères et du potentiel de ces technologies, et du fait que l'application JavInspector est développée en JAVA, il m'apparaît nettement que l'utilisation du « « framework » J2EE reste le plus approprié, bien que, pas forcément le plus simple à mettre en œuvre, et surtout à découvrir.

Cette technologie propose des API de développement Web, selon un mode multi-couches, ainsi que des outils, mais surtout une large souplesse dans l'adaptation de l'architecture du logiciel. Ce choix architectural, suite à l'étude de J2EE constitue la prochaine phase de notre projet.

III Présentation de la technologie J2EE

La conception d'un logiciel en mode client-serveur nécessite l'adoption d'une architecture adaptée. On retrouve dans cette architecture une analogie par l'architecture MVC usuelle des applications autonomes, comme nous allons le découvrir par la présentation des architectures multicouches qui suivent. Ces architectures ne sont pas des spécificités de la technologie J2EE, mais nous les explorons dans ce chapitre afin de mieux comprendre l'apport de J2EE, et l'implication dans nos choix.

1 / Les architectures Web

L'architecture à deux niveaux : dans cette architecture la charge de traitement, c'est à dire l'application, est exécutée par le client, et le serveur se contente de contrôler le trafic entre l'application et la base de donnée dont il est l'hôte.

Ce découpage en deux couches, présente de nombreux inconvénients dont l'augmentation du trafic sur le réseau, car l'application, qui est alors limitée en ressource, effectue de nombreuses requêtes. La maintenance est difficile à appliquer, car tout changement peut entraîner la modification de la base. Au final différentes versions de l'application peuvent fonctionner sur des clients différents.

L'architecture à trois niveaux, dite trois tiers : elle remédie à l'architecture deux niveaux en définissant trois niveaux logiques, qui communiquent par des interfaces normalisées.

Ces niveaux sont :

- le niveau présentation : qui contient l'IHM, reçoit des données du niveau métier et les formate avant de les afficher ; cette couche n'est pas limité au Web, mais peut aussi définir un IHM sur une application de bureau (autonome) ou encore sur Wap, Palm etc..
- le niveau métier : qui couvre la logique de l'application et qui est appelé par l'utilisateur pour extraire les données.
- le niveau données : qui contient les données nécessaires à l'application

Cette séparation de l'IHM et de la logique application permet de modifier l'interface graphique sans modification des couches supérieurs. Cette architecture est en analogie avec l'architecture MVC.

L'architecture multiniveaux : elle décompose l'architecture précédente afin d'augmenter la souplesse de l'application, pour la maintenance et le déploiement.

On trouve les couches :

- IHM : comme ci dessus, mais apte à présenter et à supporter différentes technologies tel que le Web, l'application de bureau, le Wap.. grâce à la couche logique de présentation, qui est un intermédiaire de contrôle.
- Présentation : elle prend en charge l'affichage sur l'IHM en fonction de la technologie porteuse de cette dernière. En effet la complexité de l'information

présentée dépendra de l'aptitude de la technologie de l'IHM ; c'est une couche de formatage et filtrage des données.

- Logique métier : modélise la logique de l'application.
- Service d'infrastructure : est porteur de fonctionnalités supplémentaires pour l'application ; c'est du plug-in dans un certain sens, vers des composants métiers développée en dehors de notre application.
- Niveau données : c'est la base de données, ou encore des fichiers plats..., tout ce qui a fonction de mémorisation.

Ce modèle est en faite une extension du modèle MVC avec les analogues :

- Vue → IHM (la couche présentation y est intégré dans les applications autonomes)
- Contrôleur → Logique métier
- Modèle → les données

Architecture d'entreprise : spécifique de J2EE, cette architecture explicite la couche logique métier, de façon à ce qu'elle accueille des objets intermédiaires, nommés objets applicatifs ou composants métier. L'application est alors décomposée en objets capable de communiquer par des interfaces définis. L'application devient un « patchwork » de modules réutilisables (le concept n'est pas nouveau, mais toujours pas maîtrisé en réalité!). J2EE fournit pour ces couches une architecture normalisée avec les Java Beans Entreprise, qu'il ne faut pas confondre avec le simple Java Bean.

2/ L'apport de J2EE

J2EE fournit une gamme d'outils et d'API afin de concevoir « facilement » les différentes couches.

Pour la couche présentation et logique de présentation:

- Java Servlet 2.2 fournit une couche d'abstraction orientée objet dans le cadre de l'élaboration d'applications Web Dynamique ; les servlets sont des programmes héritant de la classe Servlet, laquelle fournit les méthodes pour recevoir les requêtes http, les traiter et émettre un message, qui est généralement une nouvelle page HTML, par le cycle des JSP.
- Java Server Page (JSP) : cette extension permet de valoriser davantage les applications web J2EE en permettant le développement d'applications web basées sur des modèles ; les JSP permettent grâce au moteur de servlet de produire facilement des pages HTML.

Pour la couche logique métier et objets intermédiaires :

- Les EJB : sont des composants Java pour des applications distribuées multi niveaux. Cette extension fournit un moyen standard pour définir les composants coté serveur et définit une vaste infrastructure d'exécution pour l'hébergement des composants coté serveur.
- Le protocole RMI-IIOP : Remote Method Invocation over the Internet Inter ORB, permet la mise en l'oeuvre de l'API RMI Java classique sur IIOP. De plus, il permet aux applications RMI et CORBA de se rejoindre.

- JMV : Java Message Service, fournit une API Java pour accéder à un service de messages et permet de publier et de souscrire à divers types de services middleware orientés message.
- JNDI : Java Naming and Directory Interface ; normalise l'accès aux différents services de nommage et d'annuaire disponibles actuellement. Cette API a été conçue de façon à être indépendante de toute exécution de service de nommage et d'annuaire. J2EE définit également une interface pour fournisseur de service, le Service Provider Interface JNDI, afin que les fournisseurs de services de nommage et d'annuaire puissent la mettre en œuvre.
- API Java Transaction : elle est chargée de la mise en œuvre d'applications distribuées transactionnelles.
- Java MAIL : fournit un cadre indépendant des plates-formes et des protocoles dans le but de concevoir des applications de courrier électroniques basés sur JAVA.

Pour la couche de données :

- l'extension de Java DataBase Connectivity (JDBC) : également appelée paquetage facultatif JDBC 2.0, contribue à améliorer l'API JDBC Standard en facilitant l'obtention de connexions, le pooling de connexions, les transactions distribuées etc..

Bien sûr, seules quelques API, parmi cette liste exhaustive, dont les explications sont issues du livre (Programmation avec J2EE, ed Eyrolles, S.Allamaraju...), seront utilisées dans notre projet. Cette liste offre cependant un aperçu du pouvoir d'action offert par JAVA J2EE.

IV Conception architectural client-serveur de JavWebInspector

Couche présentation :

Selon le « cahier des charges » du projet, l'application doit être déployée sur le Web, et l'IHM portée par un navigateur. Etant donné, la complexité de l'IHM, la génération d'une IHM sous forme de pages HTML dynamiques et la mise en place d'une correspondance avec le serveur par les API Servlet et JSP reste exclue. Il est en effet plus logique de conserver l'IHM en l'état et de la porter sous forme d'Applet dans la fenêtre du navigateur.

Couche logique métier :

La couche logique métier est ici l'API développée par les précédents auteurs du logiciel, et qui consiste à produire une analyse des attributs, méthodes, et de l'arbre d'héritage d'un « classifieur ». Dans l'idéal cette API aurait dû rester sur le serveur et être transformée en EJB.

Couche de données :

Dans notre cas les données conservées pourrait être des plug-in d'analyse. Nous pourrions également envisager le développement d'une base de données conservant des informations sur les analyses effectuées.

Les délais de ce travail étant de 3 mois/homme, il n'a malheureusement pas été possible d'établir la couche logique métier et la couche de données. De ce fait seul la couche présentation a été fournie, selon les spécifications du projet.

V Refactoring de JavInspector et mise en place de la couche présentation

Le principal objectif de « refactoring » a été de préparer l'IHM pour être portable simultanément en mode Applet (classe JApplet de Java) ou en mode Frame (classe JFrame), par un mécanisme de « switch » selon que l'application est lancée depuis une fenêtre HTML, ou depuis un fichier BAT.

Tout simplement, lorsqu'une page HTML exécute l'Applet le processus débute avec la méthode *Init* d'une JApplet, et de même lorsque l'utilisateur lance l'application par un fichier BAT (ou double clique sur un JAR exécutable), c'est la méthode statique **main()** d'une JFrame qui est exécutée. Cette méthode appelle l'exécution d'une classe responsable de l'initialisation de l'application, par la méthode statique **initialize()** de cette dernière. Lorsque cette classe a terminé son initialisation, le processus a produit l'IHM (et initialisé d'autres composants), Menu, ToolBar, JTabbedPane... dont l'ensemble de la composition graphique a pour racine un objet JRootPane.

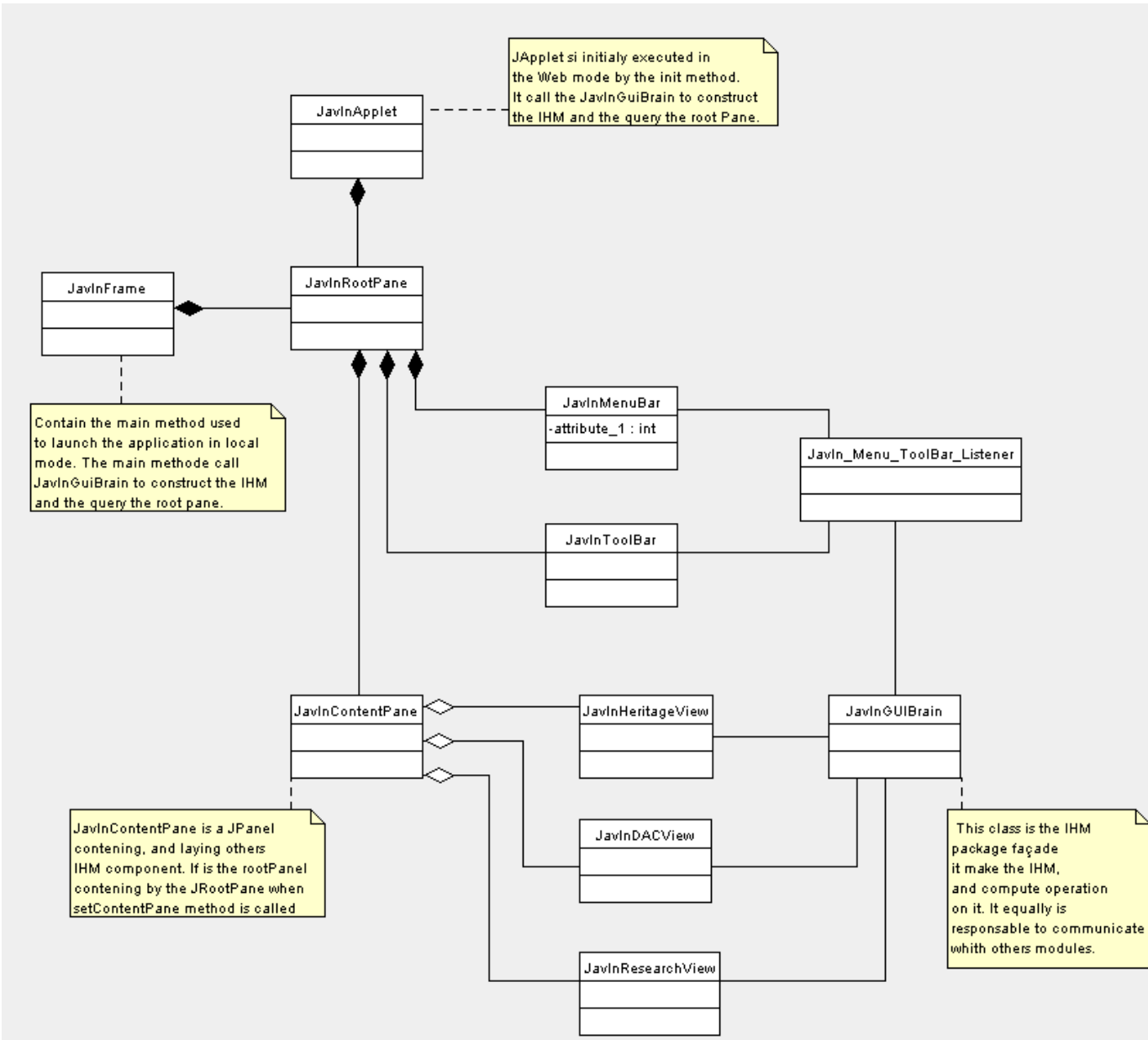
Ce JRootPane et son contenu est instancié (j'ai admis ce verbe cf. glossaire) par le processus mais non encore visible. Il ne reste alors plus qu'à l'objet JFrame ou JApplet à remplacer son JRootPane natif par celui offert par le logiciel.

Dans notre logiciel la JApplet porte le nom de JavInApplet, la JFrame est JavInFrame tandis que la classe d'initialisation qui est aussi le médiateur et le cerveau de l'IHM est la classe JavInGuiBrain.

Dans un objectif de clarté et de simplification la classe d'initialisation de l'application, JavInGuiBrain, détient une responsabilité particulière ; en effet elle construit l'IHM, c'est à dire qu'elle instancie puis assemble des composants entre eux. Ainsi au lieu d'avoir une classe JRootPane qui instancie les composants JMenu, JToolBar, et les différents panels associés aux différentes perspectives d'analyse avant de les ajouter à son LayoutManager, nous avons la classe JavInGuiBrain qui se charge de cette affaire en instanciant le JRootPane, puis le JMenu... avant d'inviter le JRootPane à associer le JMenu à son LayoutManager.

Le JavInGuiBrain se retrouve alors comme un « mécano » qui assemble les morceaux pour produire l'IHM. Ceci offre l'avantage de centraliser le procédé d'assemblage et ainsi de faciliter la procédure de maintenance. En effet les nouveaux composants graphiques produits, par exemple un menu élaboré, donneront lieu à une instanciation et une composition en cet endroit.

Diagramme UML de Classe de l'IHM :



VI Procédure d'installation du serveur J2EE et de déploiement

La version JavWebInspector se comporte au final comme une application exécutée à partir d'une Applet chez le client. Ce dernier est donc un client lourd, et le serveur devient un serveur d'application Web.

6.1 Fichier War pour J2EE

J'ai utilisé pour le déploiement l'ensemble d'outils fournis avec J2EE, bien que ce ne soit pas nécessaire, puisqu'un simple serveur Web Apache faisait l'affaire; cependant par soucis didactique et évolutif pour mes successeurs j'ai préféré fournir un fichier WAR, qu'il suffit de soumettre au système d'auto déploiement.

- Télécharger le package J2EE1.4 sur le site <http://java.sun.com/j2ee>, c'est un exécutable qui se charge de l'installation sur la machine ;
- Dans le dossier J2EEHome/docs vous trouverez la page QuickStart.html qui fournit toutes les explications de base sur la mise en route du serveur. L'exemple fourni montre comment servir un fichier WAR, une application Web, avec Hello.war ; remplacez cet exemple par l'application javain.war-deployed.
- Pour résumer : copiez l'application javain.war-deployed dans le dossier J2EE_HOME/domains/domain1/server/autodeployed.
- Lancez le serveur, ouvrez votre navigateur et entrez votre adresse locale, si en locale : <http://localhost:1024/javain/JavInApp.html>; le chemin /javain/ est un chemin contextuel ;

6.2 Fichier JAR avec un switch entre Applet et JFrame

Le fichier sJavIn.jar (s pour signé) permet de lancer l'application à partir d'un fichier « .bat », en invoquant la classe principale (fonction main) `ter.GUI.mainGUI.JavInFrame`, ou à partir d'une page html, en appelant l'applet `ter.GUI.mainGUI.JavInApplet` ;

VII A l'attention de mes successeurs

Je vous transmets par ce chapitre des remarques et réflexions personnelles sur mes écueils, et sur ce qui est à envisager lors de la reprise de ce logiciel, sachant qu'il n'est pas facile de reprendre le travail d'un autre. Ces conseils et remarques amicales m'ont également été transmises par des professionnels en architecture du logiciel et développement Java, et relèvent de mon expérience avec ces derniers :

- Le code doit subir un « cleaning », notamment pour la séparation des attributs de classes, d'instances, et locales aux méthodes ; la partie me concernant a été effectuée, au mieux.
- Le packaging doit être doté de plus de cohérence, notamment la syntaxe et la sémantique de dénomination.
- Certaines classes doivent être scindées pour départager les responsabilités (mieux vaut plusieurs petites classes même si ça semble ridicule).
- Une batterie de tests avec JUnit serait la bien venue;
- Des diagrammes UML font défaut, notamment un Use Case, et des diagrammes de séquences débutant par une action de l'utilisateur sur l'IHM ; par exemple lorsque l'utilisateur clique sur le bouton effectuer une analyse.....
- L'API analyseur de « classifieur » peut être transformé en EJB, ce qui permettra par la suite de greffer des fonctionnalités relevant d'une base de données ou d'un entrepôt de « plug-in » d'analyse.
- Chaque couche, package ou module, doit comporter une façade (Design Pattern) à laquelle font appel les autres couches du logiciel
- Lors de la composition d'un IHM : ex JTabbedPane dans JPanel, la modification des attributs de ces JComponent doit être effectuée juste après l'instanciation, afin de regrouper en un paragraphe toutes les instructions (si possible). Ceci facilite la lecture du code.

VIII Glossaire :

API : Application Programming Interface

CGI : Common Gateway Interface

Classifieur : abus de langage pour représenter un élément classificateur tel que une interface ou une classe. Je conserve ce terme auparavant utilisé dans les rapports sur JavInspector.

FrameWork : espace de travail destiné pour un objectif particulier et qui comprend des logiciels, des API, de la documentation... ce terme est galvaudé.

IHM : Interface Homme Machine ;

Instancier : le verbe n'existe pas en français, c'est un anglicisme de « instance » qui veut dire un cas, un exemple c'est à dire un exemplaire. Instancier une classe se serait fournir un exemplaire de cette classe ; en français : instance vient du instantia qui est une sollicitation pressante.

EAR : Enterprise Archive, dossier compressé,

JAR : Java Archive, dossier compressé contenant le bytecode java, permettant une exécution et un déploiement rapide.

MVC : Modèle Vue Contrôleur

WAR : Web Archive, c'est l'équivalent d'une fichier JAR

IX Bibliographie et ressources Web

« Programmation avec J2EE, Conteneurs J2EE, Servlets, JSP et EJB », de S.Allamaraju and All. Edition : Eyrolles ; Collection : Solutions développeurs.

« Guide du développeur Java Servlets et JSP », de Phil Hanna. Edition : First Interactive.

« The Design Patterns Java Companion » de Addison-Wesley ; libre sur Internet.

« Design Patternss par la pratique » de A. Shalloway et J.R. Trott ; Edition : Eyrolles ; Collection : Technologies objet.

« Design Patterns EJB Patterns avancés, processus et idiomes » de Floyd Marinescu. Edition : Vuibert.

Source Web d'informations :

Site de SUN : <http://java.sun.com>

Site d'entraide des développeurs francophones : <http://www.developpez.com>

Site Apache Software Foundation : <http://www.apache.org>