RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE

MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITÉ IBN-KHALDOUN DE TIARET

FACULTÉ DES SCIENCES APPLIQUEES

DÉPARTEMENT DE GENIE ELECTRIQUE

MEMOIRE DE FIN D'ETUDES

Pour l'obtention du diplôme de Master

Domaine : Sciences et Technologie

Filière : Génie Electrique

Spécialité : Informatique Industrielle

THÈME

RÉALISATION D'UNE INTERFACE GRAPHIQUE EN JAVA INTÉGRANT DES PROGRAMMES FORTRAN DU LGEP SIMULANT LES ONDES ÉMANANT DE LA FOUDRE

Préparé par :

M. Cissé Ousmane

M. Coulibaly Drissa Souleymane

Devant le Jury :

Nom et prénoms	Grade	Qualité
M. Adda Benatia	MAA	Président
M. Gouasmi Med	MAA	Examinateur
M. Maaskri Mustapha	MAA	Examinateur
M. Benabid Houari	MAA	Encadreur
M. MIMOUNI Abdenbi	PROF	Co-Encadreur

PROMOTION 2016 /2017

DEDICACE

Je dédie ce travail :

- *A mes parents qui ont cru en moi et se sont sacrifiés pour que je puisse avoir ce qu'ils n'ont pas eu. Qu'ALLAH fasse que votre sacrifice ne soit pas vain.*

- *A mes frères et mes sœurs, pour m'avoir servi d'exemples à suivre et m'avoir donné leur soutien inconditionnel tout au long de ma vie.*

- *A mes frères proches de la communauté des étudiants étrangère de Tiaret, pour leur soutien moral et leur solidarité.*

- *A ma famille Algérienne, la famille Dacharaoui pour son soutien et ces cinq ans inoubliables passés ensemble.*

- *A mon binôme pour tous ces moments d'insouciance et aussi ces moments de peine qu'on a eu à passer ensemble.*

- *A toute personne ayant cru en ma modeste personne et ayant œuvré pour que j'en sois l à aujourd'hui.*

A tous, je vous remercie du fond du cœur.

Cissé Ousmane

DEDICACE

Je dédie ce travail à :

Mes Parents :

Mon père, qui peut être fier et trouver ici le résultat de longues années de sacrifices et de privations pour m'aider à avancer dans la vie. Puisse Dieu faire en sorte que ce travail porte son fruit ; Merci pour les valeurs nobles, l'éducation et le soutien permanent venu de toi.

Ma Mère, qui a œuvré pour ma réussite, de par son amour, son soutien, tous les sacrifices consentis et ses précieux conseils, pour toute son assistance et sa présence dans ma vie, reçois mes sentiments et mon éternelle gratitude.

Mes grands-parents, Bolô Kouma, mes frères et sœurs ainsi que toute la famille Coulibaly.

M. Bamba Abdoulaye, professeur, recevez ici ma gratitude

L'ensemble du personnel de «BramsCenter» et de «WeDem'zB».

Ce travail est également dédié à tous mes professeurs qui doivent voir dans ce travail la fierté d'un savoir bien acquis.

Coulibaly Drissa Souleymane

REMERCIEMENTS

Nous rendons grâce à Dieu, le tout Miséricordieux, le très Miséricordieux de nous avoir donné le courage, maintenu en bonne santé, et de nous avoir permis de mener à bien ce modeste travail malgré les difficultés rencontrées.

Nous remercions le gouvernement Algérien de nous avoir permis de poursuivre nos études supérieures en Algérie.

En suite nos remerciements vont à l'endroit du corps professoral du département de ST et du génie électrique de l'université IBN KHALDOUN de Tiaret, qui nous ont accordé leurs temps et leurs savoirs précieux durant ces 5 ans. En particulier le personnel du laboratoire LGEP (M. Mimouni Abdenbi ; M. Benabid Houari ; M. Mokhtari Abdelkader ; M. Messlem Youcef ; Mm Lakhdar Asmaa ; M.Habri ; M. Omari Mohamed) pour la réalisation de ce projet.

A l'ensemble de nos promotionnels de l'année 2012 pour leurs aides et soutiens durant ces 5 dernières années en Particulier à Dacharaoui Waafa et a toute sa famille.

Nous remercions également les étudiants de la communauté étrangère de Tiaret pour ces 5 ans d'entraides et de solidarité.

Cissé Ousmane et Coulibaly Drissa Souleymane

Sommaire

Table des matières

Liste des figures

Liste des tableaux

Liste des abréviations :

EM : Electromagnétique

CEM : Compatibilité Electromagnétique

GUI : Graphical User Interface

JVM : Java Virtual Machine

JDK : Java Developpement Kit

SDK : Software Developpement Kit

API : Application programming interface

JNA : Java Native Access

JNI : Java Native Interface

IHM : Interface Homme Machine

OS : Operating System

UML : Unified Modeling Language

BD : Base de Donnée

LGEP : Laboratorie de Génie Electrique et de Plasma

INTRODUCTION GÉNÉRALE

Ce projet porte sur la réalisation d'une interface graphique Java intégrant des programmes Fortran.

En effet, le laboratoire de génie électrique et le plasma de l'université IBN Khaldoun à développer au fil des ans, des programmes (une application en langage Fortran) ayant pour objectif de simuler les ondes électromagnétiques émanant de la foudre dans un environnement électromagnétique.

Cette simulation est basée sur la résolution d'équations différentielles électromagnétiques dépendantes du temps modélisant les ondes (équations de Maxwell…). La résolution de ces équations nécessite l'utilisation des méthodes numériques, dans le cas présent celle dite FDTD (Finite Difference Time Domaine) qui génère un nombre important de calcul scientifique (calculs itératives) d'où le choix de fortran pour programmer l'algorithme de résolution, en plus les premiers concepteurs l'avaient déjà utilisé.

Le fait de simuler ses ondes se révèle être d'une importance primordiale dans l'étude de la compatibilité électromagnétique (CEM), car les perturbations électromagnétiques produites par la foudre représentent un véritable danger permanent pour tout système électrique ou électronique environnant. L'actuelle situation d'envahissement de notre environnement par tant d'appareillage électrique et électronique de type divers, exige, en matière de CEM une caractérisation de plus en plus précisée des diverses sources de nuisance en termes de pollution. Les ondes générés par la foudre font partie de ces source de pollution, en somme l'objectif étant de mieux connaitre ces ondes dans l'optique de pouvoir protéger nos systèmes.

En terme de précision, quoi de mieux qu'une solution numérique réalisée par un langage scientifique tel que le fortran.

Mais l'inconvénient majeur de ce langage de programmation réside au niveau de l'interface utilisateur qu'il fournit ; qui est l'une des plus basique(en général, une interface conversationnelle avec le terminal du système d'exploitation ou par fichier) et dont l'utilisation requière une maitriser de l'outil de programmation et d'avoir aussi un minimal de connaissance sur la structure interne du programme et son fonctionnement.

En plus, la représentation des résultats sous formes graphique n'est pas chose évidente avec fortran.

Comme exemple, l'utilisation de l'application actuelle s'effectue comme suite, il faut :

- Créer et nommer de manière spécifique des fichiers dans lesquels nous définissons les conditions de la simulation (c'est à dire les paramètres du courant, du domaine, du milieu) selon un ordre préétablie dans le programme tout en étant attentif à ne pas faire d'erreur dans les types des variables au risque de voir le logiciel se planter.

- Regrouper ces fichiers dans un répertoire précis contenant un unique exécutable de l'application Fortran afin de pouvoir lancer une simulation quelconque.

- Une fois l'exécutable lancé, le programme sollicite l'utilisateur à travers l'invité de commande à lui fournir d'autres informations complémentaires sur les paramètres de la simulation (le choix de stratification, le model de la foudre, la présence d'une tour etc…).

- Dans le cas où toutes ces étapes sont franchies correctement, la simulation est lancée et il faudra attendre la fin de tous les calculs générés (pouvant aller à une dizaine d'heures). Au terme de ceux-ci, on récupère des fichiers de sortie contenant les résultats de la simulation dans le même répertoire que l'exécutable Fortran et les fichiers d'entrés préalablement crées.

- Les résultats obtenus seront transmis manuellement par l'utilisateur à un autre logiciel pouvant tracer des graphiques (dans le cas présent <<ORIGIN>>), qui sera chargé de les représentés graphiquement sous forme de courbes afin de permettre une interprétation. Ce point implique de la part de l'utilisateur d'avoir et de maîtriser une autre application de traçage (en général, payant).

Cette interaction actuelle de l'application fortran avec l'utilisateur, présente plusieurs points de nuances :

- L'impossibilité de lancer plus d'une instance de l'application Fortran par exécutable. Ce qui, dans le souci d'un gain de temps complique une exécution en parallèle de plusieurs cas de simulation, car il faudrait avoir autant d'exécutable que de cas de simulation qu'on voudrait faire en parallèle.
- Dans le souci de garder le code fortran secret, le fait de devoir donner à l'utilisateur tant de renseignement sur la structure interne du programme n'est pas souhaitable sans oublier le fait qu'un exécutable fortran n'est pas portable d'un système d'exploitation à un autre. Ce qui nous obligerait d'avoir autant d'exécutables différents que de systèmes d'exploitation susceptible d'être utilisés par les utilisateurs à défaut de ne pas leur remettre le code fortran.
- L'utilisateur aura du mal à bien sauvegarder ses simulations, étant donné qu'il faudrait déplacer les résultats obtenus pour ne pas les écraser par une autre simulation.

Au vu de la complexité de l'exploitation et des nuances du logiciel, il parait évident d'apporter des améliorations à ce logiciel par la réalisation d'une interface utilisateur graphique(GUI).L'objectif principale de cette interface sera de pallier à tous ces désagréments sinon a défaut apporter des compris pour rendre plus confortable, simple, intuitive l'exploitation de cette application.

CHAPITRE I
RECHERCHE BIBLIOGRAPHIQUE

I.1. Introduction

L'objectif de ce projet est la réalisation d'une interface graphique utilisateur GUI (Graphical User Interface) intuitive et conviviale programmé en Java permettant d'interagir avec une application programmé en Fortran (code natif) traitant de la simulation des effets de la foudre sur l'environnement électromagnétique. Cette simulation nécessitant un nombre important d'équations générant des milliers de calcul dont le nombre augmente avec la complexité de la géométrie du problème ; en plus ces calculs doivent être précis pour refléter la réalité et s'effectuer le plus rapidement possible pour minimiser le temps de simulation d'où l'irrévocabilité du choix de fortran comme langage de programmation. Au vu de l'utilisation actuelle du l'application fortran décrite dans l'introduction générale, notre GUI doit s'atteler à quelques principales tâches :

- faire une saisie avec vérification des conditions de la simulation au travers d'une interface avant de les transmettre au programme Fortran.
- reprendre et présenter le résultat de la simulation.
- une solution de sauvegarde pour les paramètres, les résultats, et les représentations graphiques
- la possibilité de réafficher une simulation déjà effectué.

Notre recherche s'articulera autour des mots clés :

Réalisation, Interface Java, GUI, Intégrer, codes fortran, code natif, JNI, JNA, compilateur, systèmes d'exploitation.

Dans ce chapitre, on a étudié les différentes technologies proposées par java pour nous permettre d'entre en contact avec du code FORTRAN.

I.2. Justification du choix de Fortran par les initiateurs

Inventé en 1954, Fortran (FORmula TRANslator) est le plus ancien langage de programmation de haut niveau utilisé principalement pour le calcul scientifique, suivi notamment par Lisp (1958), Algol (1958) et COBOL (1959).

En effet, depuis sa création, le Fortran n'a cessé d'évoluer au cours des décennies ; ceci s'illustre par les publications de 16 versions du langage.

De Fortran I (1956) aux versions 77 et 90(les plus couramment utilisés publiées respectivement en 1978 et 1990) jusqu'à la nouvelle version connue sous le nom de << Fortran 2015>> dont la

publication est prévue vers le mi-2018 ; nous constatons une augmentation considérable du nombre de bibliothèques scientifique écrites en fortran et des efforts continus consacrés aux compilateurs pour exploiter toujours de nouvelles possibilités des calculateurs (vectorisation, coprocesseurs, parallélisme).

C'est ainsi qu'aujourd'hui encore (années 2017), le langage Fortran reste très utilisé pour plusieurs raisons :

- La présence de très nombreuses bibliothèques de fonctions, mises au point et améliorées durant de nombreuses années ;
- L'existence de logiciels en Fortran ayant demandé des ressources très importantes pour leur développement, et dont le passage à un autre langage est jugé trop couteux d'où la logique de la continuité ;
- L'existence de compilateurs performants qui produisent des exécutables très rapides. [Wikipedia]

I.3. Justification du choix de Java comme langage de programmation [1]

Pour la création d'interface graphique, il existe plusieurs langages et environnement de développement, le choix s'effectue selon les besoins. Les interfaces graphiques (GUI) sont non seulement difficiles à concevoir (du fait de la multiplicité des aspects humains et logiciel qu'il faut prendre en compte) mais aussi à implémenter [2].

De nombreuses plateforme de développement ont vus le jour dans l'optique de facilite le plus possible cette tâche ardue. Le choix dans le cadre de notre projet, c'est porté sur le langage Java sous la plateforme Eclipse à cause de ses nombreux avantages.

Mis au point par ***James Gosing*** et ***Patrick Naughton*** employés de l'entreprise **Sun Microsystems** avec le soutien de ***Bill Joy*** (Cofondateur de Sun Microsysteme) en 1995, par la suite racheté par **Oracle**, Java est un langage de programmation de type orienté objet et une plate-forme informatique utilisée dans de nombreux domaines. Il est fortement inspiré des langages C et C++ et a pour slogan WORA « Write Once Run Anywhere » c'est à dire « Ecrire une Fois, et Exécuter Partout » faisant référence à la portabilité de ce langage. Faisant partie de la « grande famille » des langages orientés objets, Java répond donc aux trois principes fondamentaux de l'approche orientée objet (POO) : l'encapsulation, le polymorphisme et l'héritage. Java intègre les concepts les plus intéressants des technologies informatiques récentes dans une plateforme de développement riche et homogène. Comme dans tous les langages de programmation, le code source est censé passer par

une ou plusieurs étapes avant d'être exécuté sur un ordinateur, ainsi la programmation avec java utilise la technique suivante :

On a tendance à dire que Java n'est un interpréteur. Ceci n'est pas entièrement exact puisque un programme source est d'abord compilé. Il n'est pourtant pas traduit en langage machine mais en byte code, celui-ci étant ensuite exécuté par une machine virtuelle Java, que l'on peut considérer comme un interpréteur.

Les avantages procurent par cette technique, représentée par l'image, sont :

- Le programme compilé, il est interprété plus rapidement
- Une fois traduit en byte Code, le programme est protégé contre des contrefaçons
- La portabilité du programme, traduit en byte code, sur une autre plateforme est assurée. Java, de ce point de vue-là, est l'un des rares langages à présenter cet avantage

L'interpréteur veille à tout moment à ce qu'il n'y ait pas d'accès non autorisé à la mémoire. Ceci renforce l'aspect sécurité, étant donné que souvent les programmes Java sont exécutés à partir de l'internet.

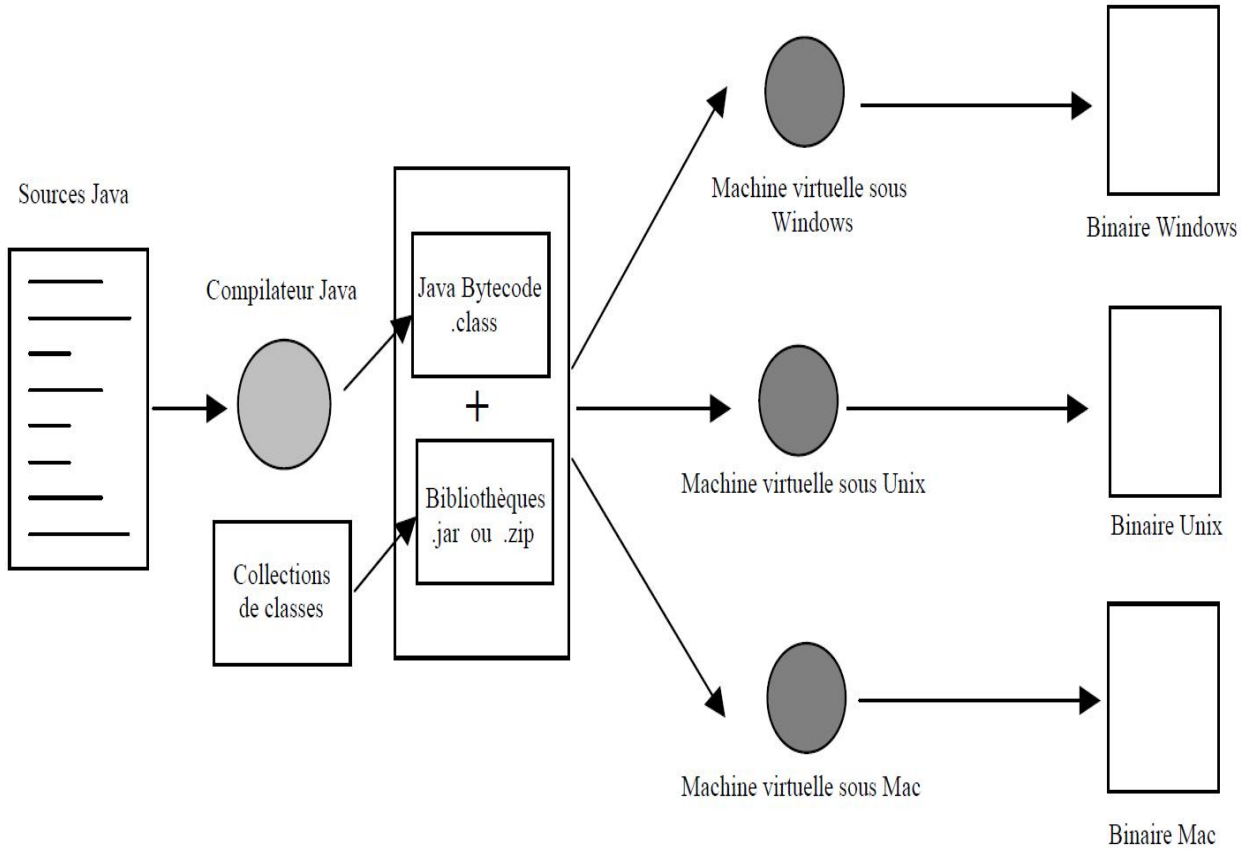


Figure 1 : Schéma du fonctionnement du langage JAVA

En résumé, le succès de java est dû à un ensemble de caractéristiques dont voici un aperçu :

- Langage de programmation objet et fortement typé : contraignants pendant le développement, l'approche objet et le fort typage du langage Java rendent plus robuste les programmes Java dès leur conception mais bien plus **structuré** et **maintenable** par la suite.
- Le programme Java **n'est pas aussi lent** que l'on pourrait le croire. Cela dépend surtout du type d'application que l'on souhaite faire et en général, les performances de java suffisent.
- Syntaxe proche du C et C++ : en reprenant une grande partie de la syntaxe de ces deux langages, Java facilite la formation initiale des programmeurs qui les connaissent déjà.
- Java est **parfaite** pour un **débutant**. C'est un langage de «***haut niveau***», c'est à dire que l'on ne se préoccupe pas de comment gérer la mémoire comme en C par exemple. Plus de pointeur. Tout est fait **automatiquement**, tout nous est caché ou presque.

La gestion de la mémoire est simplifie : le ramasse-miettes (garbage collector en anglais) intégré à Java détecte automatiquement les objets inutilisés pour libérer la mémoire qu'ils occupent. On peut cependant en fonction de la situation, configurer le Garbage Collector pour une meilleure gestion.

- Gestion des exceptions : Java l'intègre autant, pour faciliter la mise au point des programmes (détection et localisation des bugs) que pour les rendre plus robuste.
- Il est possible de compiler certaine partie du code en **code natif** pour une **meilleure performance**.
- Multitâche : grâce aux threads, Java permet de programmer l'exécution simultanée de plusieurs traitements et la synchronisation des traitements qui partagent les informations.
- Système de sécurité : Java protège l'information sensible s de l'utilisateur et le système d'exploitation de sa machine en empêchant l'exécution des programmes conçus de façon malintentionnée (contre un virus par exemple).
- Bibliothèque très riche : la bibliothèque fournie en standard avec Java couvre de nombreux domaines (gestion de collections, accès aux bases de données, interface utilisateur graphique, accès aux fichiers et aux réseaux, utilisation d'objets distribués, XML…, sans compter toutes les extensions qui s'intègrent sans difficulté à Java !).
- La **communauté** (les programmeurs) **Java** est **énorme**. Une question, un problème, il y aura toujours quelqu'un pour vous répondre.
- Les **Api Java (**Application Programming Interface**),** interface de programmation Applicative disposent d'une **bonne documentation** et **d'exemples.**
- La **rapidité** de conception d'un programme, console, IHM, bref c'est simple et rapide. Le fait que des tas de classes, API existent déjà vous faites gagner un temps considérable.

- Les environnements de développements intégrés (IDE) JAVA sont très bons comme Netbeans et Eclipse.
- Exécutable portable : comme l'exprime l'accroche Write Once Run Anywhere, un programme java, une fois écrit et compilé, peut être exécuté sans modification sur tout système qui prend en charge Java (Windows, UNIX ou Mac OS).
- Gratuit : le développement, les outils utilisés ainsi que l'exécution des programmes sont gratuits.

Java est en c**onstante évolution** du fait de sa masse importante d'utilisateur.

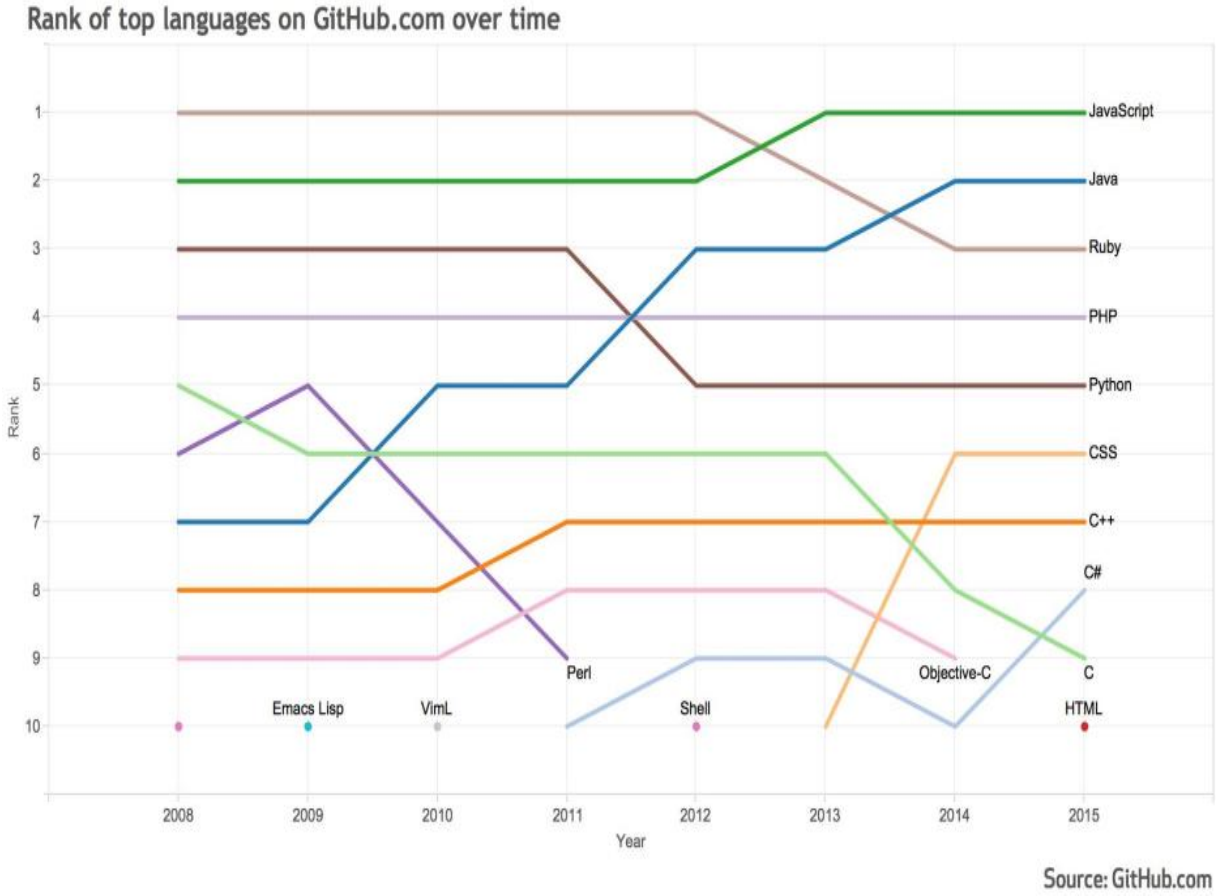


Figure 2 : Graphique de classement des langages de programmation entre 2008-2015

Le graphique des langages les plus utilisés de 2008 à 2015 suivant illustre bien nos propos :

Les langages de programmations ont évolué pour permettre aux programmeurs d'utiliser des concepts de plus en plus proches de la réalité et du langage naturel. La programmation en assembleur a remplacé le codage en binaire des données par un codage hexadécimal, et les instructions codées en binaire du microprocesseur par les instructions symboliques. Java intègre les concepts les plus intéressants des technologies informatiques récentes dans une plate-forme de développement riche et homogène. L'approche objet de ce langage, mais aussi sa portabilité et sa gratuite en font un des outils de programmation idéaux pour notre projet.

I.4. Présentation des matériels

I.4.1. JDK version 1.8_0.111 (java Développement Kit)

Cette application est téléchargeable gratuitement sur le lien :
(http://www.oracle.com/technetwork/java/javase/downloads/index-jsp-138363.html)

I.4.2. FORTRAN PowerStation 4.0

Il est aussi téléchargeable gratuitement sur liens et nos permettra de compiler le programme fortran (http://microsoft-fortran.software.informer.com/4.0/).
Fortran (FORmula TRANslator) est un langage de programmation utilisé principalement pour le calcul scientifique. Inventé en 1954, c'est le plus ancien langage de programmation de haut niveau. Le nombre de bibliothèques scientifiques écrites en Fortran, et les efforts continus consacrés aux compilateurs pour exploiter au fil des décennies les nouvelles possibilités des calculateurs (vectorisation, coprocesseurs, parallélisme) ont maintenu l'usage de ce langage, non sans d'importantes évolutions. [Wikipédia]

I.4.3 Eclipse Luna. Java 4.4 et Eclipse Neon C/C++ version 4.6

Ils sont tous téléchargeable gratuitement sur le lien (https://eclipse.org/downloads/)
Eclipse est un projet, décliné et organisé en un ensemble de sous-projets de développements logiciels, de la fondation Eclipse visant à développer un environnement de production de logiciels libre qui soit extensible, universel et polyvalent, en s'appuyant principalement sur Java. Bien qu'Eclipse ait d'abord été conçu uniquement pour produire des environnements de développement, les utilisateurs et contributeurs se sont rapidement mis à réutiliser ses briques logicielles pour des applications clientes classiques. Cela a conduit à une extension du périmètre initial d'Eclipse à toute production de logiciel : c'est l'apparition du Framework Eclipse RCP en 2004. [Wikipédia]
Ces deux logiciels nous servirons à programmer l'interface graphique et à compiler du code C/C++.

I.4.4 MingW (www.mingw.org)

Le compilateur C/C++ pour la plateforme Windows (version 5.1.3).
MinGW est l'acronyme de Minimalist Gcc for Windows. Ce projet apporte une collection d'outils permettant de produire du code natif pour la plateforme Windows. Il s'agit au fait d'une adaptation des outils de développement du GNU du monde Linux à la plateforme Windows. Concrètement,

MinGW va nous fournir l'outil GCC pour effectuer nos compilations et nos liaisons de la DLL construite pour la partie JNI.

I.4.5 Bibliothèque JNAJNA (version 3.0)

Les bibliothèques nécessaires à l'utiliser la technologie JNA. Elles sont téléchargeable sur le lien « maven.java.net/content/repositories/releases/net/java/dev/na/jna/ »

I.5. Présentation des différentes techniques exploitables dans le cadre de notre projet

Etant donné que Java est portatif mais plus lent dû au fait que le programme n'est pas compiler en code natif du processeur (binaire) ce qui représente le cas contraire du fortran ; donc pour tirer le meilleure des deux, on pourra faire appel au fortran à partir d'une GUI java en utilisant sans avoir à convertir en java l'intégralité des programmes fortran existants.

Nos recherches nous ont conduits à trois méthodes proposées par le java pour atteindre cet objectif.

I.5.1 Runtime

Avec cette technique il s'agira de lancer une application externe depuis un programme Java.

i. Description

L'exécution d'une application externe se fait grâce aux méthodes exec() de la classe Runtime. Chaque application Java possède une instance unique de la classe Runtime qui lui permet de s'interfacer avec son environnement(OS). Cette instance se récupère avec la méthode statique getRuntime() .

Pour lancer notre application, il nous suffira d'appeler l'une des six méthodes exec() de la classe Runtime et dont voici les déclarations :

```
public Process exec(String command); // Permet d'exécuter une ligne de commande dans un processus séparé.

public Process exec(String[] cmdarray);// Permet d'exécuter une commande avec ses arguments dans un processus séparé.

public Process exec(String[] cmdarray, String[] envp);// Permet d'exécuter une commande avec ses arguments dans un processus séparé en spécifiant des variables d'environnement.

public Process exec(String[] cmdarray, String[] envp, File dir);// Permet d'exécuter une commande avec ses arguments dans un processus séparé en spécifiant des variables d'environnement et le repertoire de travail.

public Process exec(String command, String[] envp);// Permet d'exécuter une ligne de commande dans un processus séparé en spécifiant des variables d'environnement.

public Process exec(String command, String[] envp, File dir);// Permet d'exécuter une ligne de commande dans un processus séparé en spécifiant des variables d'environnement et le répertoire de travail.
```

Les variables d'environnement spécifiées doivent l'être selon le format *nom=valeur*.

Il est important, si on doit lancer l'application tout en lui passant un certain nombre de paramètres de faire appel à l'une des méthodes de exec() attendant un tableau de String que de faire appel à une méthode exec() ne prenant qu'un simple String.

```
Runtime runtime = Runtime.getRuntime();
runtime.exec("monappli param1 param2");
```

Cette remarque est aussi valable si la commande elle-même contient des espaces.

```
Runtime runtime = Runtime.getRuntime();
runtime.exec(new String[]{ "C:\\Program Files\\MonAppli\\monappli.exe" });
```

Les différentes méthodes exec() renvoie un objet de type Process. Cette classe représente le processus de l'application externe et va permettre une interagir avec lui. La classe Process est une classe abstraite définissant 6 méthodes :

- la méthode **destroy**() qui permet de tuer le processus de l'application externe,
- la méthode **exitValue**() qui permet de récupérer la valeur de retour du processus de l'application externe,
- la méthode **getErrorStream**() qui permet de récupérer le flux d'erreur du processus de l'application externe,
- la méthode **getInputStream**() qui permet de récupérer le flux de sortie du processus de l'application externe,

- la méthode **getOutputStream**() qui permet de récupérer le flux d'entrée du processus de l'application externe,
- la méthode **waitFor**() qui met le thread courant en attente que le processus de l'application externe se termine.

La communication avec l'application nécessite la compréhension du fonctionnement des flux d'entrée/sortie en Java (le package java.io).

La récupération des flux se fait à travers 3 méthodes de la class Process :

- la méthode **getErrorStream**() permet de récupérer un *InputStream* représentant le flux d'erreur de l'application externe.
- la méthode **getInputStream**() permet de récupérer un *InputStream* représentant le flux de sortie de l'application externe.
- la méthode **getOutputStream**() permet de récupérer un *OutputStream* représentant le flux d'entrée de l'application externe.

Remarque :

- Il parait étrange de récupère un InputStream pour le flux de sortie standard. Il faut se placer au niveau de l'application JAVA, en effet il s'agit de la sortie standard pour l'application externe, l'application Java va lire ce flux qui est donc de son point de vue un flux d'entrée. De même pour l'entrée standard de l'application externe, du point de vue de l'application Java il s'agit d'un flux de sortie puisqu'elle y écrite (d'où l'OutputStream).
- Ambiguïté sur le rôle de Runtime :

Il faut savoir que les différentes méthodes de la class Runtime permettent de lancer une application et non d'interpréter une ligne de commande. C'est-à-dire que le programme appelé doit correspondre à un fichier exécutable et que chacun des paramètres lui seront passés tel quel sans modification.

- Problème fréquent sur le chemin d'accès de l'application, en effet si le chemin d'accès de l'application contient des espacements ou des caractères de séparations ; le lancement n'aura pas lieu car une partie du chemin sera considéré comme faisant partie des paramètres d'appel. Donc souvent, il faudra faire appel à un fichier « bat » pour ne pas avoir ce genre de problème.

ii. *Avantages*

- la simplicité apparente du code d'appel.
- exigera peu de modification sur l'application actuelle.
- pas besoin de joindre un package supplémentaire dans le JDK de Java.

iii. *Inconvénients*

- Consommation des flux :

Le problème majoritairement rencontre avec ce type d'appel est le fait que l'application semble se bloquer et cela est souvent du a une mauvaise gestion des flux. Les redirections d'entrée/sortie utilisent des buffers de taille limite (et dépendant du système hôte). Si les flux d'E/S ne sont pas traites par le programme appelant, le processus peut se trouver bloque ou pire encore, on peut facilement se retrouver dans un cas d'inter-blocage (le processus attend que le programme Java vide le buffer du flux afin de pouvoir continuer son exécution, alors que le programme Java attend que le processus fils se termine pour continuer son exécution, et donc les deux applications s'attendent mutuellement). Et toujours, dans l'optique d'éviter les inter-blocages, les différents flux doivent être traites depuis des threads différents, ce qui complique le tout.

- Une perte totale de la portabilité de l'application finale qui regrouperait les deux applications.
- La transformation des programmes en setup instable sur diverses systèmes hôtes s'avérera être très complexe.
- L'interface Java va rester figer tout au long de l'exécution de l'application fortran pour éviter des problèmes d'inter-blocage.
- Une gestion complexe d'échange en temps réel entre deux applications surtout si on voudrait afficher les résultats de la simulation au fur et à mesure que les calculs sont effectués.
- Cette technique va contraindre toute future évolution du programme fortran en étant pas capable de cumuler les exécutables.
- Une augmentation considérable des risques de Buck du programme final, surtout si le l'application fortran n'a été éprouvé.

iv. *Exemples d'utilisation*

Ici on utilise cette technique pour lancer une partie exécutable de l'application fortran à travers le Shell de Windows. Cette partie n'admet pas de paramètres d'appel et elle n'interagit pas avec le programme Java au cours de son exécution. Donc l'application Java lance le programme Fortran tout simplement.

```
public class Runt {
     public static void main(String[] args) throws IOException,
InterruptedException { File g = new
File("C:/Users/oc/Documents/Programme/programmationFortran/progFinal");
          System.out.println("Debut du programme");
          Process p = Runtime.getRuntime().exec("cmd /c start
C:/Users/oc/Documents/Programme/programmationFortran/progFinal/Debug/prog
Final.exe",args,g);
          p.waitFor();
          System.out.println("Fin du programme");
     }
}
```

I.5.2. JNI « Java Native Interface »

[www.math.ucla.edu/~anderson/JAVAclass/JavaInterface/JavaInterface,html],[3]

JNI est l'acronyme de Java Native Interface. C'est une technologie qui permet d'utiliser du code natif, notamment du C, dans une classe Java. Avec cette méthode, la communication entre nos différents programmes devra suivre le protocole suivant :

- ✓ De l'interface aux codes fortran

GUI --> Java Code --> JNI --> C code --> FORTRAN code

- ✓ Du fortran à l'interface

FORTRAN code --> C code --> JNI --> Java Code --> GUI

JNI est une passerelle qui permet de faire un appel à une fonction native. Toutefois, l'appel n'est pas direct. Il est ainsi obligatoire de définir une méthode native qui respecte un prototype précis. De ce fait, il devient obligatoire de passer par une méthode intermédiaire qui englobera cet appel. Appeler le FORTRAN à partir de C est assez facile. Les fonctions et les sous-programmes peuvent être appelés comme des fonctions C, à condition que les paramètres soient donnes par référence, c'est-à-dire par pointeurs, et non par valeur.

Dans notre cas, la technologie JNI est une utilisée pour relier le Java a C, puis on se sert du C pour interfacer le fortran, elle est fournie par défaut avec le JDK et nécessite de manipuler le langage natif pour effectuer les appels aux fonctions natives ; Toutefois, au moment de la création de projets JNI, des fichiers header (identification de la JVM par exemple) devront être liés lors de la phase de liaison. Les fichiers header sont disponibles à la racine du répertoire JDK dans les répertoires :

- JAVA_HOME%\include
- JAVA_HOME%\include\win32

suivant le type de système d'exploitation utilisé.

La mise en œuvre de JNI nécessite plusieurs étapes :

- la déclaration et l'utilisation de la ou des méthodes natives dans la classe Java
- la compilation de la classe Java
- la génération du fichier d'en-tête avec l'outil javah

- l'écriture du code natif en utilisant entre autres les fichiers d'en-tête fournis par le JDK et celui généré précédemment
- la compilation du code natif sous la forme d'une bibliothèque (dans le cas présent le code C associer au code Fortran)
- La bibliothèque est donc dépendante du système d'exploitation pour lequel elle est développée : .dll pour les systèmes de type Windows, .so pour les systèmes de type Unix, ...

i. *Description*

✓ **L'écriture et la compilation du Code Java**

Le code java doit contenir la déclaration et l'utilisation des méthodes natives. Cette déclaration est assez simple puisqu'il suffit de déclarer la signature des méthodes avec le modificateur native, ce qui permet au compilateur de savoir que ces méthodes sont contenues dans des bibliothèques natives. Les méthodes natives doivent avoir les mêmes noms que les méthodes du code C.

Pour pouvoir utiliser une méthode native, il faut tout d'abord charger la bibliothèque. Pour réaliser ce chargement, il faut utiliser la méthode statique loadLibrary() de la classe system et obligatoirement s'assurer que la bibliothèque est chargée avant le premier appel de la méthode native.

Le plus simple pour assurer ce chargement est de le demander dans un morceau de code d'initialisation statique de la classe.

Le nom de la bibliothèque fournie en paramètre doit être indépendant de la plate-forme utilisée : il faut préciser le nom de la bibliothèque sans son extension. Le nom sera automatiquement adapté selon le système d'exploitation sur lequel le code Java est exécuté. L'utilisation de la méthode native dans le code Java se fait de la même façon qu'une méthode classique.

Apres l'écriture du code, il faut la compiler avec la commande javac ou à travers l'IDE utilise.

✓ **La génération du Fichier d'en-tête**

L'outil javah fourni avec le JDK permet de générer un fichier d'en-tête qui va contenir la définition dans le langage C des fonctions correspondant aux méthodes déclarées natives dans le code source Java.

Javah utilise le byte code pour générer le fichier .h. Il faut donc que la classe Java soit préalablement compilée.

```
javah -jni nom_fichier_sans_extension
```

La syntaxe est donc :

```
Java_nomPleinementQualifieDeLaClasse_NomDeLaMethode
```

Le nom de chaque fonction native respecte le format suivant :

Ce fichier doit être utilisé dans l'implémentation du code de la fonction.

Même si la méthode native est déclarée sans paramètre, il y a toujours deux paramètres passés à la fonction native :

- un pointeur vers une structure JniEnv : cette structure permet d'invoquer certaines fonctionnalités natives de JNI grâce à un tableau de pointeurs de fonctions initialisé par la JVM
- jobject qui est l'objet lui-même : c'est l'équivalent du mot clé this dans le code Java

✓ **l'écriture et la compilation du programme Fortran**

Prendre le programme fortran existant et transformer le programme principal en fonction ou sous-programme appelable à partir du C. Toutes les données sont passées entre le C et Fortran par paramètre.

La compilation se fait avec la commande gfortran d'un compilateur fortran ou à l'aide d'un IDE prenant en charge le fortran.

✓ **L'écriture du code natif en C**

La bibliothèque contenant la ou les fonctions qui seront appelées doit être écrite dans un langage (c ou c++) et compilée.

L'écriture en C est facilitée par la génération du fichier.h, il est nécessaire en plus des includes liées au code des fonctions d'inclure deux fichiers d'en-tête :

- jni.h qui est fourni avec le JDK
- le fichier .h généré par la commande javah

On doit nommer les fonctions du code C en fonction de la déclaration leur prototype dans le fichier d'entête. Cela remplace main () si un code existant est utilisé.

Le code C doit contenir une déclaration du prototype du code Fortran sous forme de fonction avec le mot clé extern juste après les includes.

Il faut aussi insérer la déclaration de pointeurs spéciaux au début du code C et les libérer à la fin du code.

Il faut compiler ce fichier source sous la forme d'un fichier objet .o en utilisant la commande Gcc d'un compilateur prenant en charge le langage C ou à travers un IDE.

- ✓ **La création d'un fichier de type .def**

Il faut ensuite définir un fichier .def qui contient la définition des fonctions exportées par la bibliothèque.

- ✓ **La création du dll**

On doit générer un fichier dll qui regroupe les codes fortran et C. Cette opération peut s'effectuer avec l'une des commandes gcc ou gfortran d'un compilateur pouvant les supporte ou à l'aide d'un IDE.

- ✓ **L'exécution du programme final**

Il ne reste plus qu'à exécuter le code Java dans une machine virtuelle.

Il est intéressant de noter que tant que la signature de la méthode native ne change pas, il est inutile de recompiler la classe Java si la fonction dans la bibliothèque est modifiée et recompilée.

ii. *Avantages*

- Cette technique incorpore le code fortran dans le code Java, ce qui permet une intégration facile de composants fortran existant déjà éprouve.
- L'évolution du code fortran pourra se faire par module sans avoir à modifier quoique soit dans le programme actuel, puisqu'il suffira d'intègre chaque module en tant qu'une fonction dans l'interface.
- La possibilité d'améliore plus facilement la portabilité du programme fini à travers sa transformation en Setup, qui va incorporer un Makefile et les différents compilateurs nécessaires à son installation.
- La possibilité de tirer le meilleur des deux langages à savoir :
  - La portabilité de java.
  - La vitesse d'exécution et les performances de Fortran.
- Une gestion simplifiée de la communication entre les différents langages entrant en jeux lors de l'exécution du programme fini.

- L'accessibilité en temps réel des résultats des calculs fortran, ce qui pourra nous permettre de faire l'affichage et le trace au fur et à mesure que le fortran effectuera les calculs.
- Une minimisation des risques de plantage de l'application finale.
- L'utilisateur n'aura pas conscience de l'exécution du code fortran.

iii. Inconvénients

- La complexité de la réalisation de la technique.
- La nécessite d'au moins deux compilateurs ou de deux IDE.
- La création d'un makefile pour facilite les essais lors du développement.
- La restructuration du programme initiale.
- Une exigence de compatibilité entre le dll, le système d'exploitation et les outils de développements utilisés.
- Le développeur devra se conformer à la correspondance entre les types de variables entre Java-C et entre C-Fortran.

iv. *Exemple d'utilisation*

Dans cet exemple, le programme Java (**JavaCode.java**) crée un tableau d'entier de dix éléments, dont les éléments seront initialise dans l'ordre croissant de 0 à 9 puis afficher à l'écran. Apres le programme java fait appel à la fonction **sumsquaredc** (programmée en C dans un fichier nomme **Ccode.c** et retournant un entier) en lui passant en paramètre une copie du tableau d'entier. La fonction **sumsquaredc** signale à l'écran que le programme se trouve au niveau du langage C et affiche le contenu du tableau dont elle a reçu une copie lors de son appel. Par la suite, la fonction **sumsquaredc** fait appel à la fonction **SUMSQUAREDF** en lui passant par paramètre un entier qui correspond à la taille du tableau et ce tableau. La fonction **SUMSQUAREDF** est programmée en fortran, elle retourne un entier et prend deux paramètres (un entier et un tableau). Cette dernière fonction va calculer et remplacer chaque élément du tableau par son carrée avant de calculer la somme de ses carrées qu'elle va retourner à son appelant, qui correspond à la fonction **sumsquaredc**. Cette dernière va signaler à l'écran le retour du programme en langage C et afficher le contenu de son tableau et la somme reçus du programme Fortran. A partir de cet instant, le programme signale le retour en Java et affiche le contenu et la somme reçu de la fonction C avant de signaler la fin du programme.

Comme indiquer dans la description de la technologie JNI, la réalisation de ce programme se passe comme suite :

- ✓ **L'écriture et la compilation du code Java**

```
class JavaCode
{   final static int MAXSIZE = 10;
   private native int sumsquaredc(int arr[]);
  public static void main(String args[]) {
   System.out.println("Nous sommes dans le programme java");
   JavaCode c = new JavaCode();
   int arr[] = new int[MAXSIZE];
   System.out.println("Initialisation du Tableau");
   for(int i=0; i< MAXSIZE; i++) {
      arr[i]=i;
      System.out.println(i+"   "+arr[i]);}
    System.out.println("Appel du Code C");
    int sum = c.sumsquaredc(arr);
    System.out.println(" Nous sommes de retour ans java");
    System.out.println(" Le contenu du tableau arr[]");
    for(int i=0; i< MAXSIZE; i++)
     System.out.println(i+"    "+arr[i]);
    System.out.println("Somme des elements du tableau ="+ sum);
    System.out.println("Exit Java");
   }
 static {
      System.loadLibrary("codageC");
   }
}
```

La compilation se fait avec : `javac JavaCode.java`

✓ **La génération du Fichier d'en-tête**

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <C:\Program Files\Java\jdk1.8.0_121\include\jni.h>
/* Header for class JavaCode */

#ifndef _Included_JavaCode
#define _Included_JavaCode
#ifdef __cplusplus
extern "C" {
#endif
#undef JavaCode_MAXSIZE
#define JavaCode_MAXSIZE 10L
/*
 * Class:     JavaCode
 * Method:    sumsquaredc
 * Signature: ([I)I
 */
JNIEXPORT jint JNICALL Java_JavaCode_sumsquaredc
  (JNIEnv *, jobject, jintArray);

#ifdef __cplusplus
}
#endif
#endif
```

Elle se fait par la commande :

```
javah –jni JavaCode
```

✓ **L'écriture et la compilation du programme Fortran**

```
INTEGER FUNCTION SUMSQUAREDF(N,A)
INTEGER A(*)
DO I=1,N
ENDDO
ISUM=0
DO I=1,N
  A(I)=A(I)*A(I)
  ISUM=ISUM+A(I)
  WRITE(6,'(3I5)')I,A(I),ISUM
ENDDO
SUMSQUAREDF=ISUM
END
```

La compilation se fait avec la commande :

```
gfortran -c -m64 FortranCode.f
```

- ✓ **L'écriture du code natif en C**

```
#include <stdio.h>
#include "C:\Users\oc\Documents\Programme\programmationMixte\ESSAJNI\JavaCode.h"
extern  int sumsquaredf_(int* , jint []);
JNIEXPORT jint JNICALL Java_JavaCode_sumsquaredc(JNIEnv *env,jobject obj, jintArray
ja)
{
int n = (*env)->GetArrayLength(env, ja);
 jint *a = (*env)->GetIntArrayElements(env, ja, 0);
 int i,result=0;
    printf("--Nous nous trouvons dans le programme C--\n");
    printf("--Affichage du contenu du tableau copie depuis le java--\n");
    for(i=0 ; i< n ; i++)
          {
           printf("%2d %5d\n",i,a[i]);
          }
    printf("Appel a FortranCode\n");
    result = sumsquaredf_( &n , a);
    printf("--Nous sommes de retour dans le programme C --\n");
    printf("Le contenu du tableau \n");
           for(i=0;i<n;i++)
           {
            printf("%2d %5d \n",i,a[i]);
            }
    printf("Somme des elements du tableau a[] = %d\n",result);
     (*env)->ReleaseIntArrayElements(env, ja, a, 0);
      return result;
}
```

La compilation se fait avec la commande :

```
gcc -c -m64 Ccode.c
```

- ✓ La création d'un fichier def

```
EXPORTS
Java_JavaCode_sumsquaredc
```

- ✓ La création du dll

On peut le faire au travers de l'IDE DEV ou par la commande :

```
gcc -shared -m64 -o codageC.dll FortranCode.o Ccode.o txt.def
```

- ✓ L'exécution du programme final

```
java JavaCode
```

Les résultats à l'affichage (invité de commande)

```
Nous sommes dans le programme java
Initialisation du Tableau
0   0
1   1
2   2
3   3
4   4
5   5
6   6
7   7
8   8
9   9
Appel du Code C
--Nous nous trouvons dans le programme C--
--Affichage du contenu du tableau copie depuis  java--
 0    0
 1    1
 2    2
 3    3
 4    4
 5    5
 6    6
 7    7
 8    8
 9    9
Appel a FortranCode
--Nous sommes de retour dans le programme C --
Le contenu du tableau
 0    0
 1    1
 2    4
 3    9
 4   16
 5   25
 6   36
 7   49
 8   64
 9   81
Somme des elements du tableau a[] = 285
 Nous sommes de retour ans java
 Le contenu du tableau arr[]
0   0
1   1
2   4
3   9
4   16
5   25
6   36
7   49
8   64
9   81
Somme des elements du tableau =285
Exit Java
```

I.5.3. JNA « Java Native Access » [3]

JNA (Java Native Access) est une API permettant d'accéder à du code natif sans faire appel explicitement à la couche de programmation JNI. Le développement nécessite une interface Java pour décrire le prototype les fonctions et les structures contenues dans le code natif à appeler.

i. Description

En effet Java Native Access est une extension à java ; il s'agit de la bibliothèque disponible pour java depuis sa version 1.4 (JNA 3.5.2) qui s'occupe du chargement des bibliothèques dynamiques, de l'appel des fonctions, des DLL sous Windows ,de la définition des structures et de la conversion des types,… si bien que toutes les étapes fastidieuses liées à la manipulation de code C/C++ pour réaliser la passerelle entre Java et le code natif sont rendues très simples.

La bibliothèque JNA utilise une petite bibliothèque native appelée bibliothèque d'interface de fonction étrangère (library called foreign function interface library (libffi)) pour appeler dynamiquement le code natif.

JNA se présente comme un choix beaucoup plus simple d'accès, en permettant d'accéder dynamiquement à n'importe quelle bibliothèque partagée du système sans utiliser JNI. Ainsi Pour utiliser des fonctions C, il suffit d'inclure le fichier qui les définit et de déclarer l'en-tête de ces fonctions dans une interface.

Le développement nécessite une interface Java pour décrire le prototype, les fonctions et les structures contenues dans le code natif à appeler. Cette interface doit obligatoirement étendre l'interface **com.sun.jna.Library** qui se trouve dans le fichier jna.jar à ajouter au projet java. Contrairement à JNI, l'utilisation de JNA nécessite le téléchargement d'une bibliothèque spécifique, qu'on peut trouver sur le site la bibliothèque JNA

• Site de JNA : github.com/twall/jna.

•Bibliothèque JNA : maven.java.net/content/repositories/releases/net/java/dev/na/jna/.[3]

À l'exécution une instance valide de cette interface pourra être récupérée. Cette instance sera automatiquement liée à la bibliothèque native et elle sera utilisée pour réaliser directement les appels aux méthodes natives.

Pour réaliser un simple appel de méthode via JNA, nous nous retrouvons à suivre un protocole en deux étapes :

• déclarer les méthodes natives dans une interface ;

• instancier dynamiquement l'interface

ii. *Avantages*

- Utilisation est plus simple que la méthode JNI ;
- pas besoin d'installer un compilateur C/C++ ;
- pas besoin de disposer d'un environnement de développement pour le langage C/C++ ;
- pas besoin de retoucher ton code natif, il faut juste le fichier JNA.jar qu'il faut ajouter à ton projet
- JNA est développé et testé sur Mac OS X, Microsoft Windows, FreeBSD / OpenBSD, Solaris, Linux, AIX, Windows Mobile et Android.

iii. *Inconvénients*

On paye la simplicité de JNA par rapport à JNI par une perte de vitesse.

La bibliothèque spécifique JNA (JNA.jar) à importer est une base de données avec des millions d'informations, nous comprenons alors que toute la bibliothèque sera parcourue afin de trouver notre code natif désiré. Pour des programmes simples vous ne vous rendrez pas compte de cet aspect mais lorsqu'il vous faut faire des opérations temps réel de haute précision, cela pourrait poser des problèmes. JNA est considéré comme étant 3 à 5 fois plus lentes que JNI.

iv. *Exemple*

Dans cet exemple assez simple nous allons utiliser la fonction ***puts*** du langage **C**, qui est fournie par le fichier *msvcrt.dll* sous Windows pour afficher un texte « Hello World ! ».

Pour ce faire nous allons créer un nouveau projet java dans l'environnement d'Eclipse nommé « hello.java » ensuite ajouter à celui-ci la bibliothèque JNA (fichier jna.jar) téléchargée au préalable.

Créer l'interface suivante :

```
package CInterface;

import com.sun.jna.Library;

public interface CInterface extends Library
{
      public int puts(String str);
}
```

Nous avons déclaré l'interface *CInterface* qui est une sous-classe de *jna*. Dans cette interface, les fonctions C sont déclarées comme méthodes. Pour utiliser la méthode *puts*, il suffit de créer une instance de *CInterface* dans notre projet.

```
CInterface malibC = (CInterface) Native.loadLibrary(libName, CInterface.class);

malibC.puts("Hello World!");
```

Pour que cela fonctionne, les imports suivants sont nécessaires :

```
import com.sun.jna.Library;

 import com.sun.jna.Native;
```

Le fichier hello.java et l'inerface CInterface se présentent comme suit :

```
package CInterface;
  import com.sun.jna.Native;
   public class JNATEST{
         public static void main(String[ ] args){
 CInterface malibC = (CInterface) Native.loadLibrary("c", CInterface.class);

 malibC.puts("Hello World!");
         }}
```

```
package CInterface;

import com.sun.jna.Library;

public interface CInterface extends Library
{
     public int puts(String str);
}
```

Remarque : Dans cet exemple le fichier hello.java et l'interface CInterface sont tous les deux situés dans un même package CInterface qui est spécifier en début de programme.

Nous pourrions ajouter à l'interface toutes les fonctions dont nous avons besoin si elles sont présentes dans le même fichier DLL et créer une interface pour chaque fichier DLL à inclure.

Java Native Access est connu par ces nombreux domaines d'application :

- ***Armed Bear Common Lisp(le langage de programmation Lisp)***
- ***JRuby utilise JNA pour la fonctionnalité POSIX ;***
- ***Freedom for Media in Java (FMJ)***
- ***IntelliJ IDEA IDE par la compagnie JetBrains***
- ***OpenSearchServer un moteur de recherche open source***
- ***Videolan vlcj du language Java.***
- ***Cyberduck FTP, SFTP, WebDAV, fichier Cloud et Amazon S3 Browser pour Mac OS X.***
- ***Webdriver***
- ***Cassandra, une Base de données open source proposée par l'entreprise Apache Software Foundation*** *[Wikipédia]*

I.5.4. Discussion et Conclusion

Apres avoir étudié ces technologies permettant de faire appel à du code natif à partir du java, la solution basée sur la JNI fournie par la JDK et l'autre basée sur la bibliothèque JNA sort du lot. On remarque que la solution JNI est un peu plus compliquée par rapport à la solution JNA car il faut manipuler le code C/C++ et aussi de l'installation d'outils supplémentaires. Alors que la solution JNA par sa bibliothèque faisant office de passerelle entre la bibliothèque dynamique et le code java permet de s'affranchir de la couche de programmation entre le Java et le C, mais le code fortran devra tout de même être lié au code java lors de la définition des méthodes natives et la communication inverse (le fortran a l'interface) n'est pas évidente. [3]

A l'heure du choix entre JNI et JNA, nous ne pouvons pas être catégoriques. Le choix doit s'opérer selon les cas rencontre. Des besoins spécifiques amèneront à utiliser obligatoirement une solution JNI, tels sera probablement notre cas, car l'application Fortran devra être maintenu en vue d'autres améliorations. Mais il est indéniable que JNA permet de simplifier largement les appels aux fonctions natives, d'ailleurs Eclipse a un projet d'intégration de JNA dans le JDK en incubation et pourrait être disponible dans la prochaine version de Java1.9 (qui sera disponible à partir du mois dejuillet2017).[Blocdeveloppez.com]

Tableau 1: TABLEAU RECAPITULATIF DE LA RECHERCHE BIBLIOGRAPHIQUE

Technologie	Avantages	Inconvénients
Runtime	•la simplicité apparente du code d'appel. •exigera peu de modification sur l'application actuelle. •pas besoin de joindre un package supplémentaire dans le JDK de Java.	•Une gestion assez complexe de la consommation des flux •Une perte totale de la portabilité de l'application finale qui regrouperait les deux applications. •La transformation des programmes en setup instable sur diverses systèmes hôtes s'avérera être très complexe. •L'interface Java va rester figer tout au long de l'exécution de l'application fortran pour éviter des problèmes d'inter-blocage. •Une gestion complexe voire impossible d'échange en temps réel entre deux applications surtout si on voudrait afficher les résultats de la simulation au fur et à mesure que les calculs sont effectués. •Cette technique va contraindre toute future évolution du programme fortran en étant pas capable de cumuler les exécutables. •Une augmentation considérable des risques de Buck du programme final, surtout si le l'application fortran n'a été éprouvé.
JNI « Java Native Interface »	•Incorporation du code fortran dans le code Java, ce qui permet une intégration facile de composants fortran existant déjà éprouve. •L'évolution du code fortran pourra se faire par module sans avoir à modifier quoique soit dans le programme actuel, puisqu'il suffira d'intègre chaque module en tant qu'une fonction dans l'interface. •La possibilité d'améliore plus facilement la portabilité du programme fini à travers sa transformation en Setup, qui va incorporer un Makefile et les différents compilateurs nécessaires à son installation. •La possibilité de tirer le meilleur des deux langages à savoir : -La portabilité de java. -La vitesse d'exécution et les performances de Fortran. •Une gestion simplifiée de la communication entre les différents langages entrant en jeux lors de l'exécution du programme fini. •L'accessibilité en temps réel des résultats des calculs fortran, ce qui pourra nous permettre de faire l'affichage et le trace au fur et à mesure que le fortran effectuera les calculs. •Une minimisation des risques de plantage de l'application finale. •L'utilisateur n'aura pas conscience de l'exécution du code fortran.	•La complexité de la réalisation de la technique. •La nécessite d'au moins deux compilateurs ou de deux IDE. •La création d'un makefile pour facilite les essais lors du développement. •La restructuration du programme initiale. •Une exigence de compatibilité entre le dll, le système d'exploitation et les outils de développements utilisés. •Le développeur devra se conformer à la correspondance entre les types de variables entre Java-C et entre C-Fortran.
JNA «Java Natif Acces »	• Utilisation est plus simple que la méthode JNI ; • pas besoin d'installer un compilateur C/C++ ; • pas besoin de disposer d'un environnement de développement pour le langage C/C++ ; • pas besoin de retoucher ton code natif, il faut juste le fichier JNA.jar qu'il faut ajouter à ton projet • JNA est développé et testé sur Mac OS X, Microsoft Windows, FreeBSD / OpenBSD, Sola-ris, Linux, AIX, Windows Mobile et Android.	On paye la simplicité de JNA par rapport à JNI par une perte de vitesse.

CHAPITRE II
CONCEPTION DE L'INTERFACE

II.1. Introduction

Ce chapitre est consacré à la fois à l'esthétique(Design) qu'au côté organisationnel et fonctionnel de l'interface. Nous avons développé cette phase après de nombreuses explications fournies par le personnel du laboratoire travaillant actuellement sur le sujet (M. Mimouni, M. Mokhtari, M. Omari Mohamed) sur le fonctionnement et l'utilisation des programmes FORTRAN. Ce développement s'est effectué sous les directives, les recommandations et les conseils de monsieur Mokhtari membre du laboratoire lgep.

Apres ces discutions avec les futurs concepteurs et utilisateurs de l'application FORTRAN, nous avons pu élaborer cette organisation à partir du cahier des charges fourni dont les étapes sont les suivantes :

A. Géométrie du problème
   - ✓ Domaine rectangulaire (Lx * Ly)
   - ✓ Maillage : fixer Δx et Δy (calculer Nx et Ny) ou fixer Nx et Ny (calculer Δx et Δy)
   - ✓ Paramètres physique (ε, μ, σ)
   - ✓ Choisir le type des BC (Boundary Conditions :Conditions aux limites)
     - PEC
     - 1er Ordre
     - 2^{em} Ordre
     - PML
     - UPML
     - CPML …
   - ✓ La source
   - ✓ Fixer le temps de simulation Tsim (Δt et Condition de stabilité(Test))

     Optionnel * Existence d'obstacles

   - ✓ Choisir le type de sauvegarde
     - Choisir la composante à garder
     - Choisir les points d'observation
     - Choisir une cartographie

B. .Vérifier graphiquement la configuration

C. Création d'un fichier « data » (création d'un répertoire qui contient configuration et résultats fortran)

D. Lancer le calcul (System cmd, DOS)

E. Exploitation des Résultats (Traçage , Traçage en 3D, Animation)

Ainsi, nous en sommes arrivés à organiser l'interface en une pile d'onglets :

- ✓ L'onglet définissant le domaine de simulation
- ✓ L'onglet définissant les conditions aux limites
- ✓ L'onglet définissant le model du courant de la foudre
- ✓ L'onglet définissant les paramètres électromagnétiques
- ✓ L'onglet représentant l'ensemble des paramètres de la simulation
- ✓ L'onglet montrant les résultats de la simulation sous forme de tableaux
- ✓ L'onglet affichant les résultats sous forme de courbes

II.2. L'onglet domaine de simulation

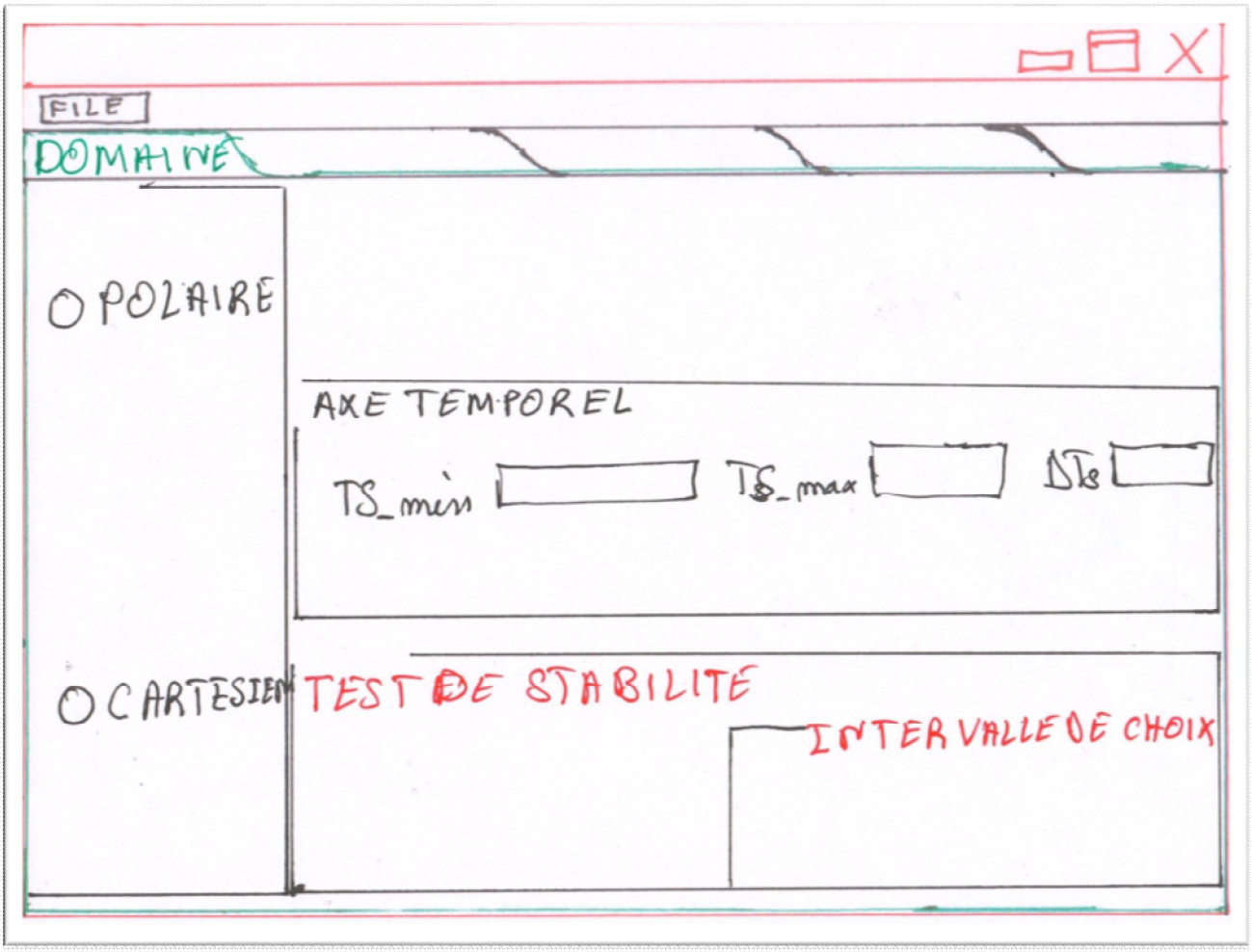


Figure 3 : Illustration conceptuelle de l'onglet Domaine de Simulation

Ce premier Onglet a pour objectif de permettre à l'utilisateur d'introduire les paramètres nécessaires à la définition du domaine dans lequel la simulation aura lieu ; en d'autres termes définir la géométrie du problème. Il se présentera comme suite :

Dans ce premier onglet nous avons les panels suivant : ***choix***, ***Domaine de la simulation***, ***AXE TEMPOREL***, ***TEST DE STABILITE, INTERVALLE DE CHOIX***.

- ***panel choix :***

Situé à l'EST ce panel nous permet de choisir le système de coordonnées désirées pour notre domaine de simulation. Il contient deux boutons radio (JRadioButon) nommés POLAIRE et CARTESIEN. L'utilisateur ne pourra cliquer que sur l'un des deux boutons à la fois, et en fonction du bouton choisi les champs de saisis spécifiques du panel ***Domaine de la simulation*** feront leurs apparitions automatiquement ainsi les coordonnées introduites seront soient polaires ou cartésiens.

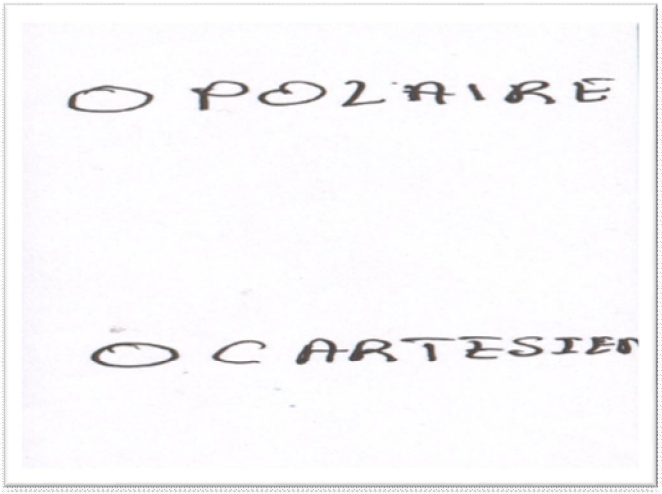


Figure 4 : Illustration conceptuelle du panel Choix de l'onglet Domaine de Simulation

- ***panel Domaine de la simulation* :**

Ce panel se situe au nord de notre onglet. Il contient les différents champs de saisie (JTextfield) qui permettront à l'utilisateur introduire les coordonnées (Abscisses, Ordonnées, Hauteurs, radiales..) du domaine. Selon le système de coordonnées sélectionné nous avons les champs de saisie appropriés qui apparaissent dans ce panel comme suite :

✓ **Système de coordonnées Cartésien :**

X_min, Y_min, Z_min : les valeurs minimales suivant respectivement les axes(X, Y, Z).

X_max, Y_max, Z_max : les valeurs maximales suivant respectivement les axes(X, Y, Z).

ΔX, ΔY, ΔZ : les pas suivant respectivement les axes(X, Y, Z).

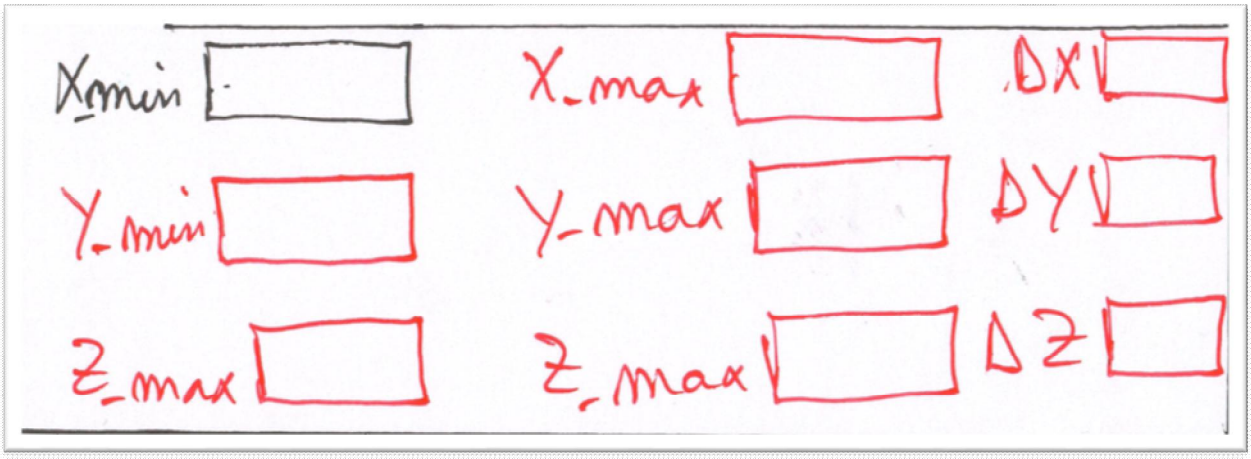


Figure 5 : Illustration conceptuelle du panel domaine 1 de simulation de l'onglet Domaine de Simulation

✓ **Système de coordonnées Polaire:**

ρ_min, Z_min : les valeurs minimales l'axe rho (coordonnées radiales).

ρ_max , Z_max : les valeurs maximales suivant l'axe Z et rho (coordonnées radiales).

Δρ et ΔZ sont les pas respectifs.

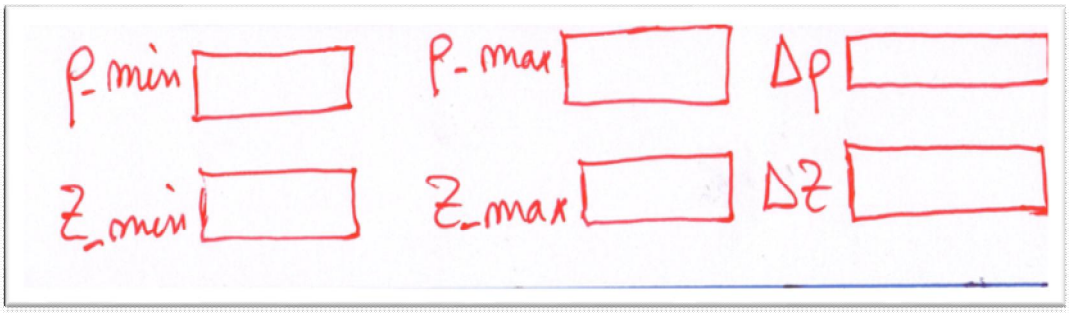


Figure 6 : Illustration conceptuelle du panel domaine 2 de simulation de l'onglet Domaine de Simulation

✓ ***panel espace temporel***

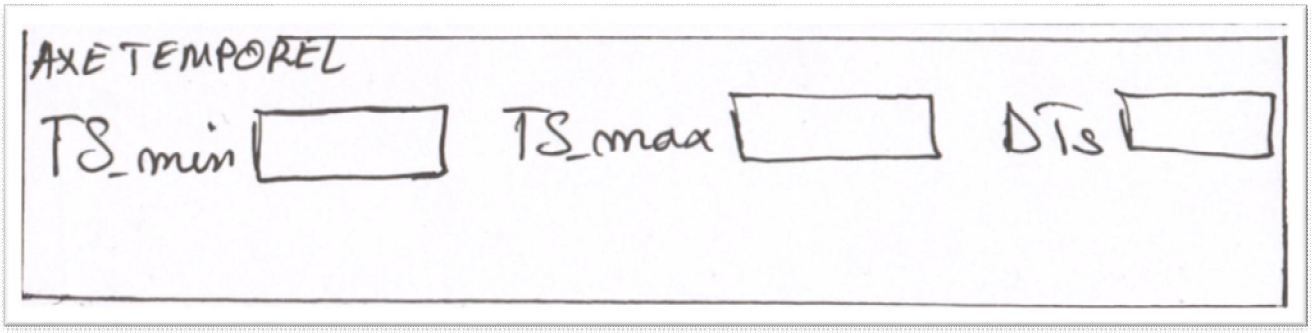


Figure **7 : Illustration conceptuelle du panel Espace Temporel de l'onglet Domaine de Simulation**

Ce troisième panel l'onglet se situe au centre. Il appelle l'utilisateur à saisir les paramètres du temps de la simulation et se présente comme suite : Avec Ts_min l'origine de l'axe du temps, Ts_max la valeur maximale du temps de la simulation et ΔTs le pas (espacement temporel).

➢ panel *test de stabilite*

Ce panel devra recevoir dans une amélioration future de l'interface, les paramètres pour effectuer le Test de stabilité.

➢ *panel intervalle de choix*

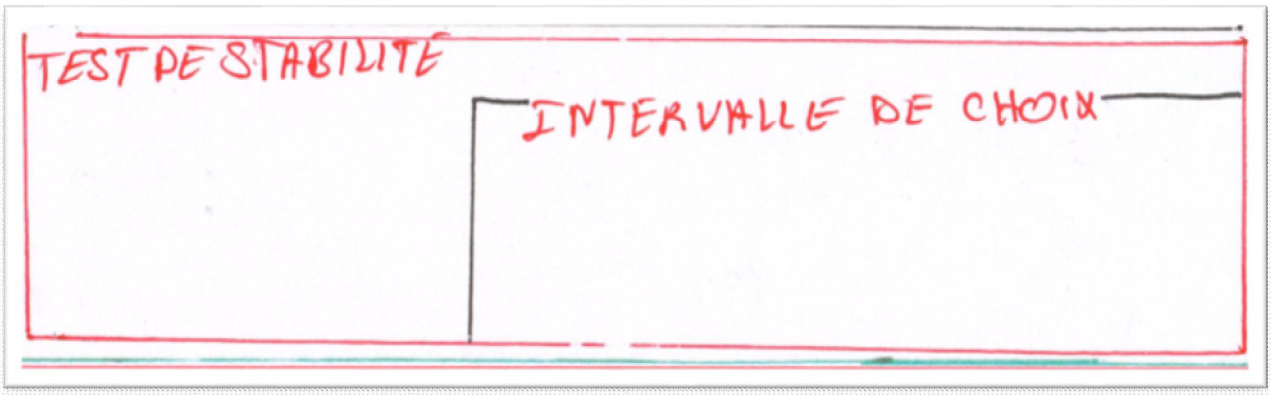


Figure 8 : Illustration conceptuelle du panel Test de Stabilité de l'onglet Domaine de Simulation

Tout comme le précèdent panel, ce panel devra être compléter avec les futures améliorations qui seront apportés

II.3. L'onglet définissant les Conditions aux limites

En effet, lorsque les équations du champ électromagnétique sont résolues dans le domaine temporel en utilisant des méthodes aux différences finies dans un espace non borné, il doit y avoir une méthode limitant le domaine dans lequel le champ est calculé. Ceci est réalisé en employant des conditions aux limites absorbantes (Absorbing Boundary Conditions : ABC) aux frontières artificielles du domaine pour simuler l'espace non borné. [6]

Le programme Fortran utilise les conditions aux limites absorbantes au **premier ordre** développées par ***Mur***. Mais c'est dans une perspective d'amélioration du programme fortran que le choix de l'utilisateur sera étendu dans l'interface Java.

L'utilisateur pourra alors donc, dans le panel ***Conditions aux limites*** contenu dans ce second onglet de notre interface java, sélectionner l'un des trois boutons correspondant aux conditions : **Mur d'ordre 1**, **Mur d'ordre 2** et **PML.**

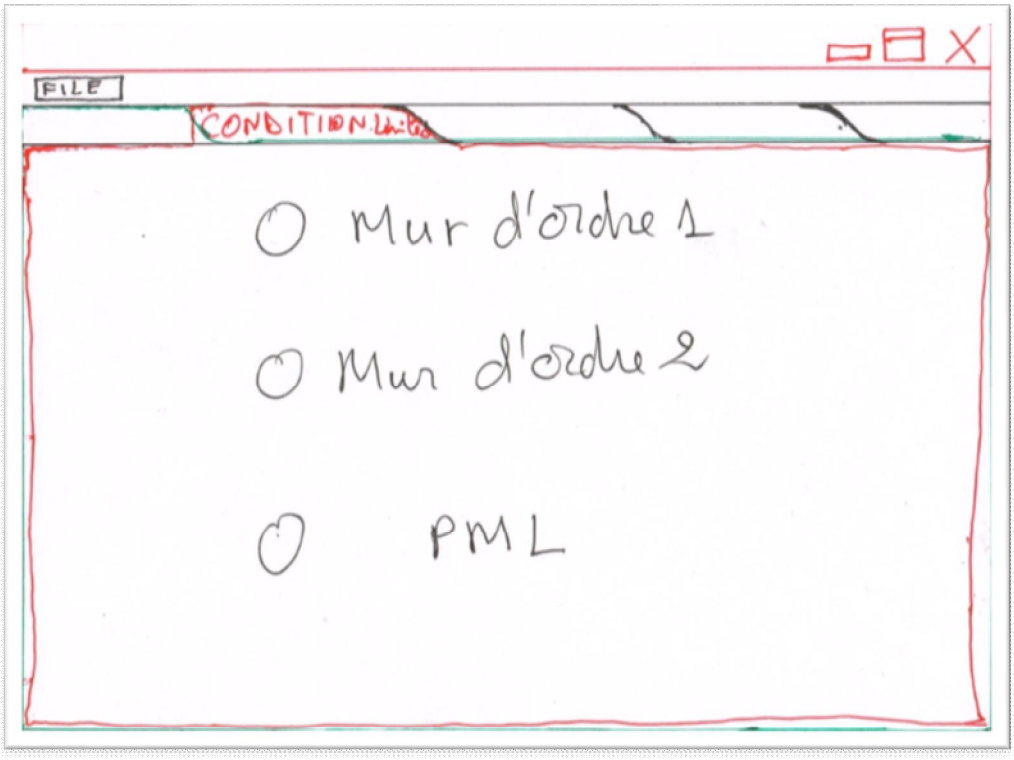


Figure 9 : Illustration conceptuelle de l'onglet Conditions aux Limites

II.4. L'onglet définissant le model du courant de la foudre

.

Les modèles utilisées pour modéliser les rayonnements électromagnétiques dans le programme fortran étant les *modèles de l'ingénieur [6] ;* ce troisième onglet de l'interface qui possède deux panels, propose donc à l'utilisateur de choisir un parmi les différents modèles de l'ingénieur pour la simulation.

A cet effet, dans le premier panel à travers des boutons l'utilisateur ferra un choix du model de courant de la foutre parmi les modèles suivants :

- Modèle de Bruce et Golde BG
- Modèle de la ligne de transmission TL
- Modèle de la ligne de transmission modifié avec décroissance exponentielle MTLE
- Modèle de la ligne de transmission modifié avec décroissance linéaire MTLL
- Modèle de la source de courant mobile TCS

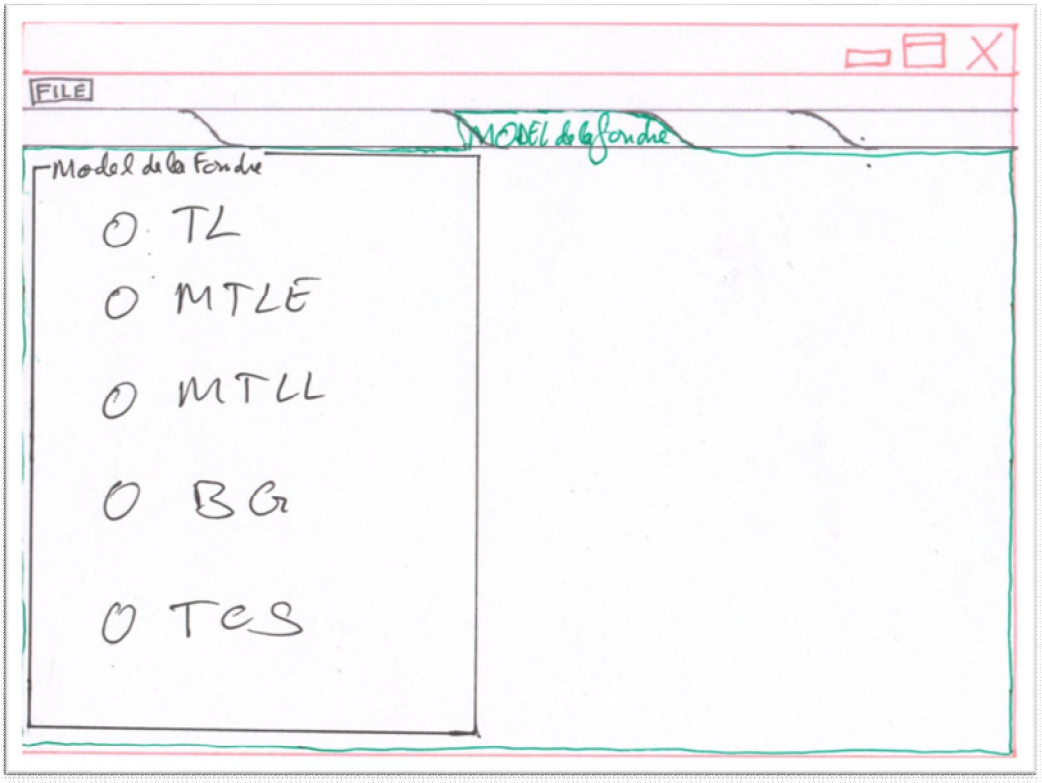


Figure 10 : Illustration du panel Model de la Foudre de l'onglet Model Foudre

Apres sélection du model effectuée, les champs de saisis des paramètres respectifs associés à cet model feront leurs apparitions dans le second panel de l'onglet comme suite :

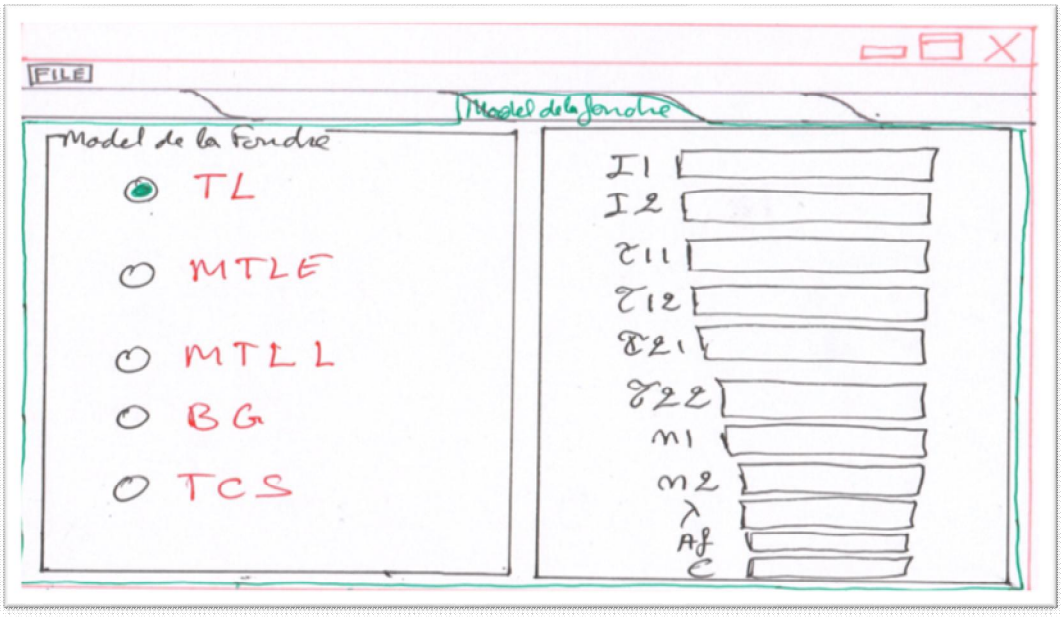


Figure 11 : Illustration des deux panels de l'onglet Model Foudre

II.1.4. L'onglet définissant les paramètres électromagnétiques

C'est dans ce quatrième onglet que le reste des paramètres nécessaires à la simulation seront définis. Il possède cinq panels qui sont : ***Informations relatives aux propriétés du sol, Stratification, Couches, Tour, Model de réflexion.***

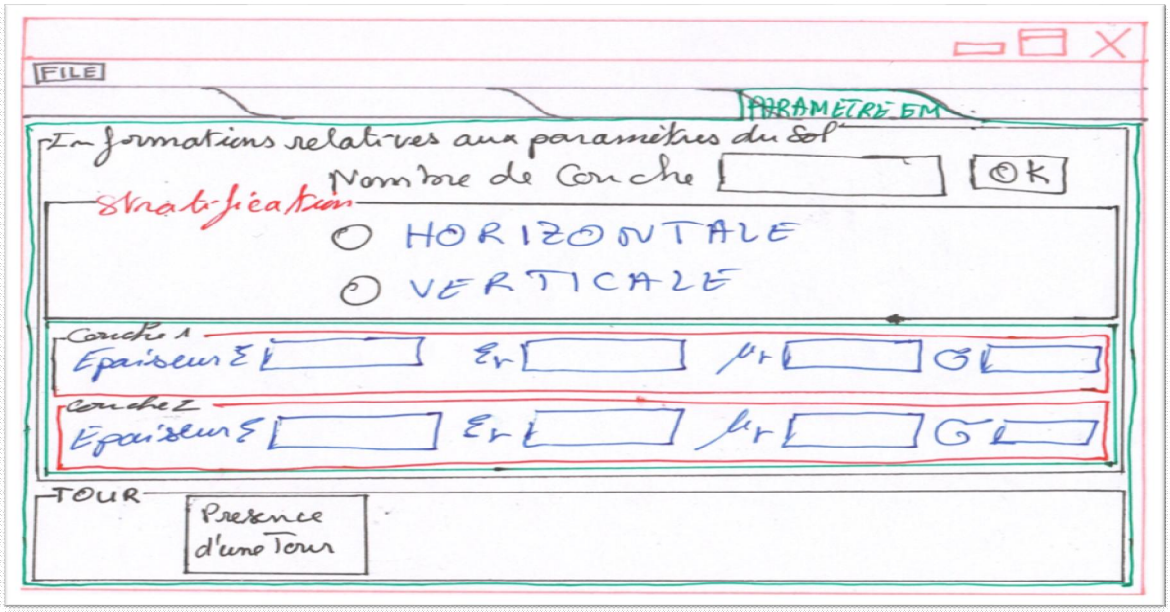


Figure 12 : Illustration conceptuelle de l'onglet Paramètres EM

panel Informations relatives aux propriétés du sol

Cette fenêtre amène l'utilisateur à saisir le nombre de couche du sol et à valider son choix, ceci grâce à un champ de saisis et un bouton de validation. Selon le nombre de couche entré par l'utilisateur, deux cas se présentent à lui :

- Si le nombre de couche saisi est 1 : c'est à dire que le sol est homogène alors l'utilisateur passe directement au panel ***Couches*** pour entrer les caractéristiques de son unique couche.
- Par contre Si le nombre de couche saisi est supérieur à 1 : l'interface sollicite dans ce cas l'utilisateur à spécifier tout d'abord le type de stratification du sol dans le panel ***Stratification*** ensuite il devra fournir les paramètres spécifiques de chaque couche de son sol.

panel Stratification

Ce panel offre à l'utilisateur de l'interface la possibilité de spécifier, grâce à des boutons, le type de stratification du sol dans le cas où ce dernier possèderait plusieurs couches. C'est-à-dire spécifier la manière dont les différentes couches du sol sont séparées. La stratification s pourra être soit Horizontale ou Verticale.

panel Couches

C'est dans ce panel que les caractéristiques de la couche ou bien des couches du sol seront définies. En effet en fonction du nombre de couche validée par l'utilisateur, ce panel réservera une case dédiée pour chaque couche dans laquelle les informations sur la dite couches seront fournies par l'utilisateur.

Le panel dédié à une couche se présente comme suit :

Les couches sont différentiées par système de numérotation automatique ; Chaque couche possède quatre champs de saisie qui sont : μr la perméabilité magnétique, σ la conductivité électrique, εr la permittivité diélectrique de la couche, et un champ Epaisseur(si la stratification est horizontale) ou bien un champ Largeur(si la stratification est Verticale).

Un panel Tour :

Ici l'utilisateur pourra indiquer la présence ou non d'une tour.

Un panel Model de réflexion :

En cas de la présence d'une tour l'utilisateur doit sélectionner le type de réflexion : ***Baba et Rokov*** ou bien ***Rachidi.***

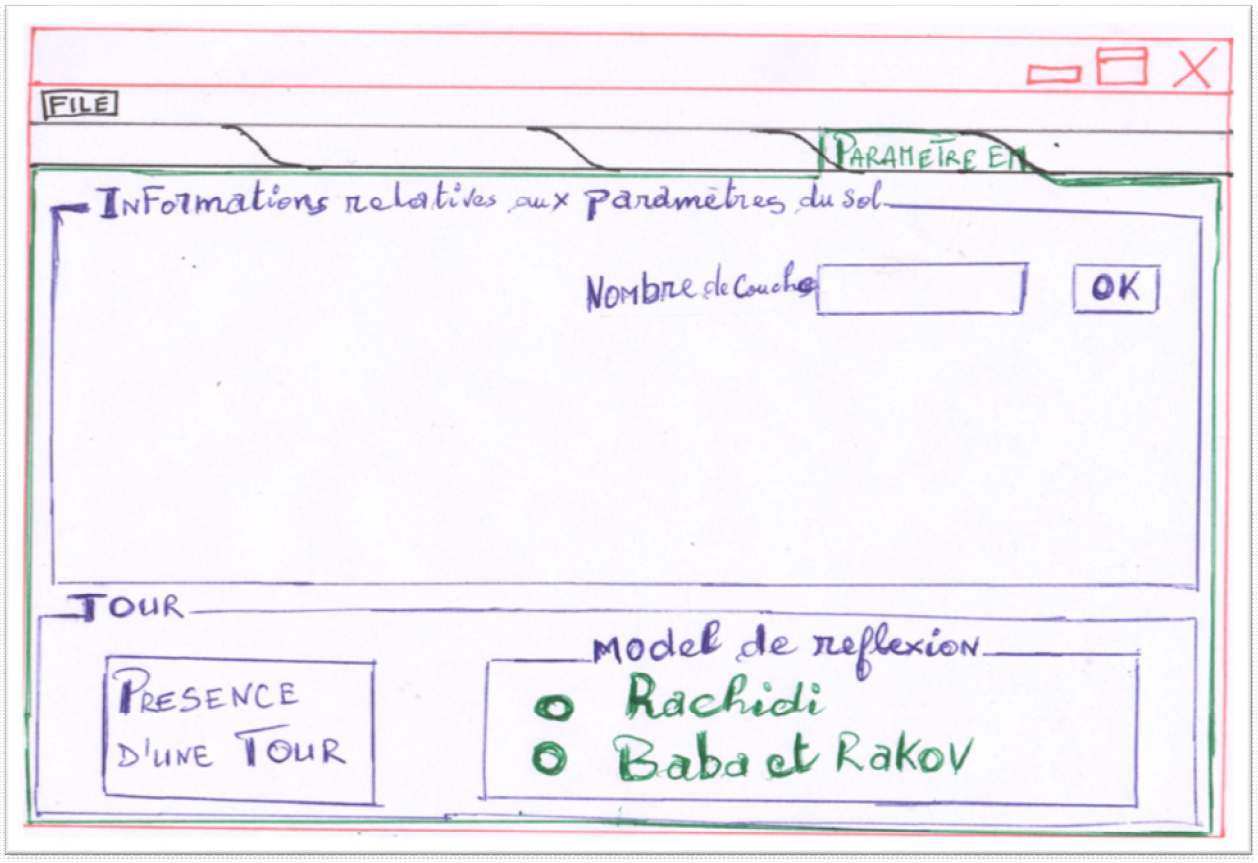


Figure 13 : Illustration conceptuelle du panel Tour de l'onglet Paramètres EM

II.1.5. L'onglet représentant l'ensemble des paramètres de la simulation

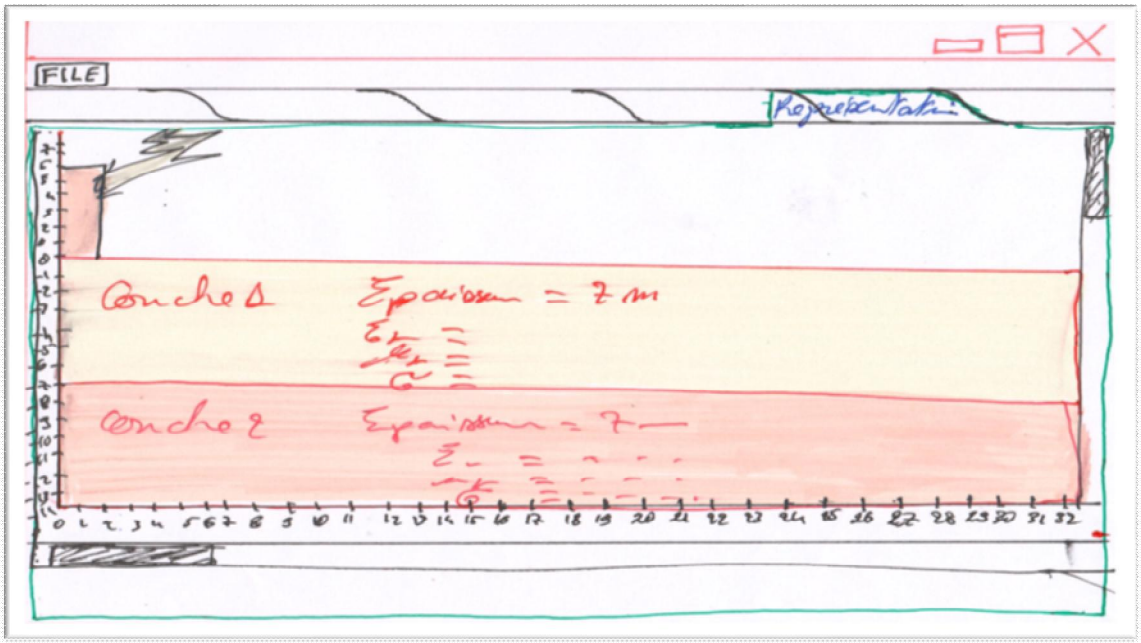


Figure 14 : Illustration conceptuelle de l'onglet Représentation Graphique

Cet onglet servira à récapituler de façon graphique l'ensemble des paramètres dans tous les onglets précédents, permettant ainsi à l'utilisateur de voir facilement ses erreurs qui seront logiques pour des vérifications effectuées par l'interface.

II.1.6. L'onglet montrant les résultats de la simulation sous forme de tableaux

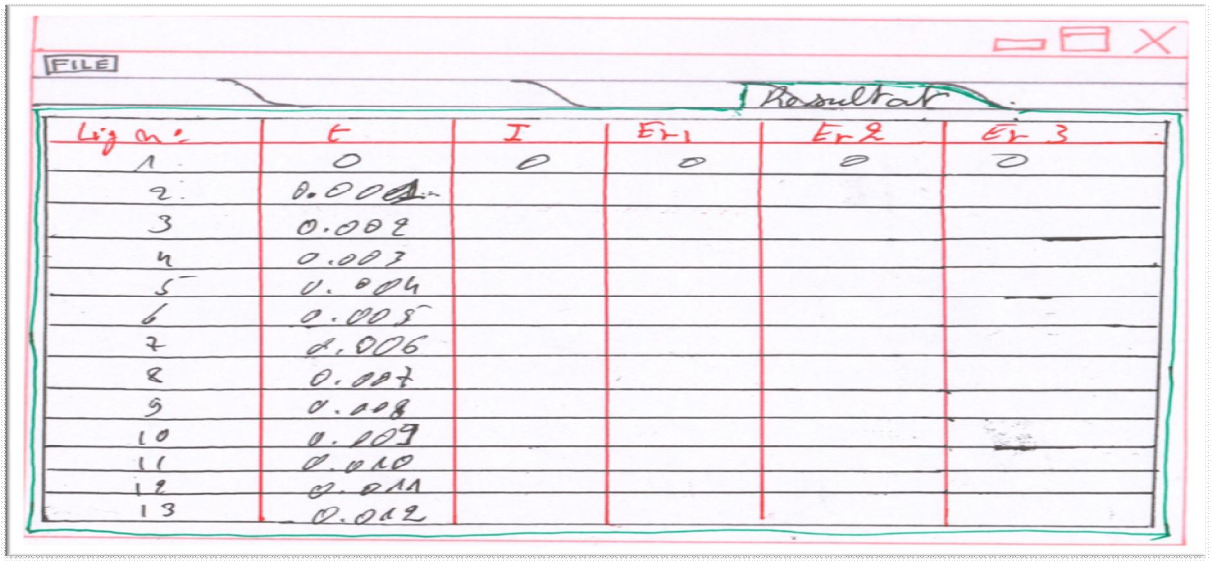


Figure 15 : Illustration conceptuelle de l'onglet Affiche Résultats

Cet onglet devra permettre une récupération avec affichage sous forme de tableaux des résultats obtenus de la simulation.

II.1.7. L'onglet affichant les résultats sous forme de courbes

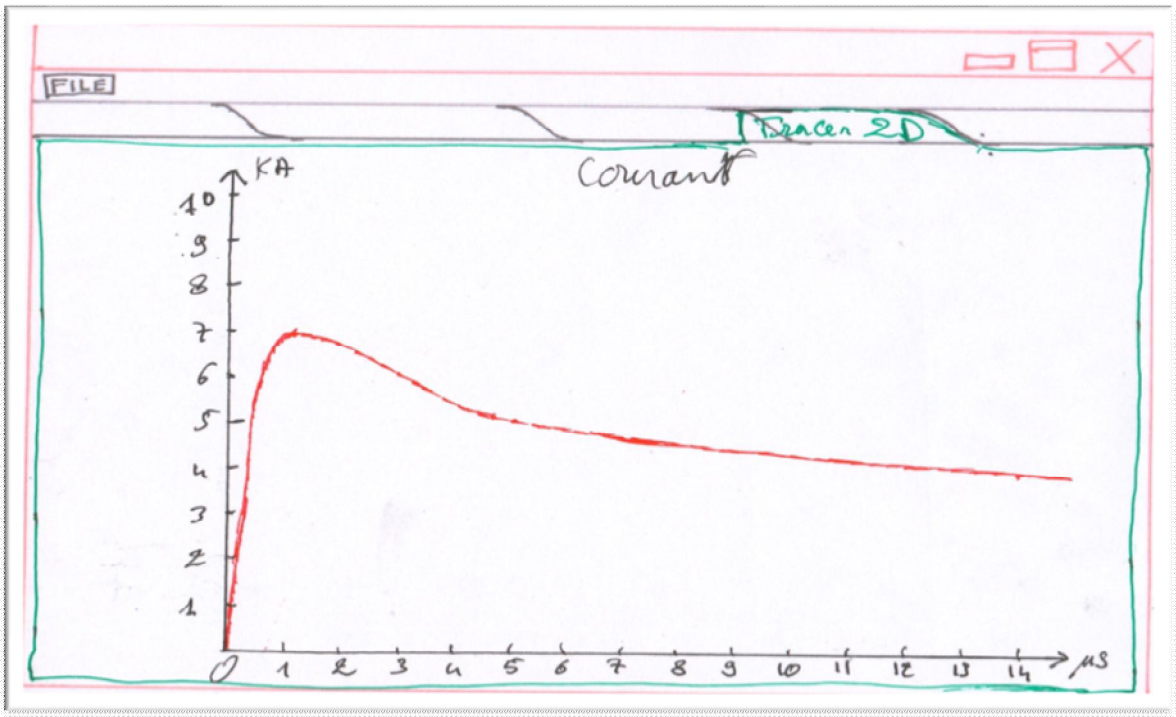


Figure 16 : Illustration conceptuelle de l'onglet Tracer 2D

Cet onglet va générer des courbes en deux dimensions à partir des données (résultats) récupérées, avec possibilité d'affichage multiple et de les sauvegarder en image.

Remarque : Un panel sera généralement situer au sud des onglets, et il aura pour but d'afficher la progression, les erreurs de l'utilisateur et ainsi que trois boutons (« Retour », « Modifier » et « Suivant »).

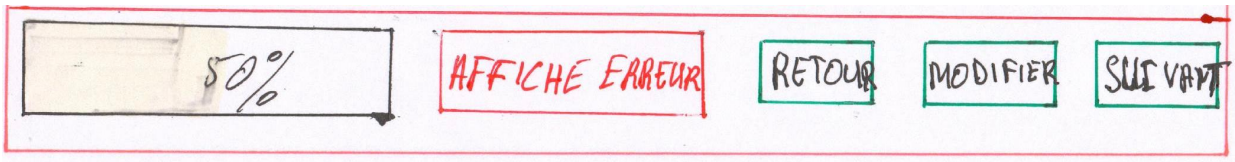


Figure 17 : Illustration conceptuelle du Panel Sud des onglets

CHAPITRE III
RÉALISATION DE L'INTERFACE

III.1. Introduction

Dans ce chapitre, nous nous intéressons a la partie programmation en java (réalisation) de l'interface déterminée lors de la conception. Il s'agira d'une description sommaire de l'agencement des différentes classes réalisées entrant dans la composition de l'application finale.

III.2. Les classes de contrôle de saisie

Pour minimiser, sinon voir rendre impossible les erreurs de saisie, nous avons créé des classes adaptées aux contrôles désirés car les objets prédéfinis par le JDK ne correspondent pas ici dans notre cas à nos besoin.

Nous avons donc crée 4 classes :

- Une classe pour les réels ;
- Une classe pour les réels positifs ;
- Une classe pour les réels positifs non nuls ;
- Une classe pour les entiers.

III.2.1. La classe pour les réels

Cette classe hérite de la classe prédéfinie KeyAdapter. Elle définit des variables d'instances (une dizaine) entrant dans le processus de vérification du caractère saisi, deux autres variables, un champ de saisie (celle qu'elle va contrôler), une zone de texte (ErrorLabel) pour signaler l'erreur à l'utilisateur à travers un affichage et enfin une variable de type double pour garder la valeur saisie.

Elle admet un constructeur de la forme suivante :

- `Reel(JTextField cd, double val,JLabel lab);`

Où JTestFiel= le champ de saisie à contrôler

double val= la variable devant contenir la valeur saisie

JLabel lab= zone pour afficher les erreurs.

Les méthodes qu'elle définit sont les suivantes :

- `public boolean num(char carac) ;`

Cette méthode prend un caractère et vérifie si cette dernière correspond aux caractères autorisés dans la syntaxe d'une variable de type réel. Si le caractère saisi est autorisé alors la méthode retourne Vrai, dans le cas échéant retourne Faux.

- `public void typeTraiter( KeyEvent e) ;`

  Cette méthode utilise la méthode num ( ) pour vérifier si le caractère saisi est autorisé.
  Si oui, elle vérifie si la position du caractère est valide sinon elle l'efface.
  Si non, elle efface le caractère saisie.

- `public void keyReleased(KeyEvent e);`

  Cette méthode hérite de la classe KeyAdapter permettant de réagir à un évènement de type touche relâchée. On utilisera ce écouteur(Listener) pour effectuer notre traitement (la méthode typeTraiter).

III.2.2. La classe pour les réels positifs

Elle hérite de la classe Reel, redéfinit sa méthode typeTrairer ( ) et admet un seul constructeur :

- `ReelPositif(JTextField cd, double val, JLabel lab);`

III.2.3. La classe pour les réels positifs non nuls

Elle hérite de la classe ReelPositif et redéfinit sa méthode typeTraiter ( ) pour effectuer la saisie de zéro en premier lieu (pour les valeurs de type 0.0001, il faudra utiliser la notation scientifique 1E-4).
Cette classe admet un seul constructeur :

- ReelPositifNonNul(JTextField cd, **double** val, JLabel lab);

III.2.4. La classe pour les entiers

Cette quatrième classe herite de la classe Reel et redefini ses methodes num ( ) et typeTraiter pour atteindre son object.
Elle a un seul constructeur :

- `Entier(JTextField cd, double val, JLabel lab);`

III.3. La classe PanelLabelText

Dans la conception, plusieurs onglets utilisent des champs de saisie de texte permettant à l'utilisateur d'introduire des valeurs. Cette classe nous permet de créer et placer à notre guise un ensemble de champs de saisie de texte. Pour ce faire :

- ✓ Elle hérite de la classe JPanel ;
- ✓ Elle admet les variables d'instances :
  - Un tableau de champs de texte (JTextField)

- Un tableau de label (JLabel)
- Un tableau de Double
- Un entier nombre de case
- Une zone d'affichage de texte (JLabel)

✓ Elle a un seul constructeur :

- Public PanelLabelText (int nbreCase, list<string> liste , int [ ] choix, JLabel lab) ;

nbreCase : définit le nombre total de champs de saisie à construire

liste : définit les labels à afficher à côté des champs.

choix : est un tableau d'entier pour effectuer un choix sur le type de contrôle à effectuer sur chaque champ de saisie.

lab : sert à afficher des messages à l'utilisateur concernant les erreurs qu'il aurait effectué.

Ce constructeur instanciera l'ensemble des variables d'instances. Une fois l'instanciation terminée, il dimensionne et place les composants en utilisant une combinaison des LayoutManager pour avoir l'affichage voulue à la conception.

En fonction du tableau d'entier «Choix», il associera un type de contrôle à chaque champ de saisie un KeyListener.

✓ La classe définit aussi quelques méthodes :

- Void inaccessible() ; pour rendre inaccessible à l'utilisateur les champs de saisie.
- Void accessible() ; pour rendre accessible à l'utilisateur les champs de saisie
- Boolean verifContend Part() ; cette méthode retourne un booléen qui sera True si tout les champs de saisie contiennent au moins un caractère sinon retourne False.
- Void validate( ) ; pour rafraichir l'ensemble des champs de saisie.
- Void supre() ; pour supprimer le dernier caractère de chaque champ de saisie.

III.4. La classe PanelRadioButton

Comme on peut le remarquer, lors de la conception, nous utilisons quelques groupes de boutons dépendants.

Pour ne pas avoir à écrire leur code chaque fois que cela sera nécessaire, on crée une classe qui prendra des paramètres et en fonction de ces derniers elle génère et places les boutons de façon automatique.

- Cette classe hérite de la classe JPanel qui contiendra les différents boutons.
- Elle génère des variables d'instances suivantes :
  - Un tableau de Panel pour aider lors du positionnement des boutons ;

- Un tableau de de JRadioButton (des boutons) ;
- Un entier correspondra au nombre de boutons à générer ;
- Et un groupe de boutons ButtonGroup pour regrouper les boutons ainsi que des icônes associées aux boutons.

o Elle admet un seul constructeur :

- Public PanelRadioButton (string Titre, int nbrBut, list<string> liste) ;

La variable Titre doit contenir le titre qui sera affecté et affiché au bordure du Panel (herité).

nbreBut : le nombre de boutons à créer.

liste doit contenir les captions (titre des boutons visible par l'utilisateur) des boutons. Il est alors évident que la liste doit correspondre au nbreBut.

Le constructeur va instancier l'ensemble des variables d'instances avant de les disposées convenablement dans le panel.

o La classe définit également deux méthodes :

- Void inaccessible() ; pour rendre inaccessible à la fois l'ensembles des composants du groupe de buttons.
- Void accessible() ; permet de rendre accessible à la fois l'ensembles des composants du groupe de buttons.

III.5. La classe PanelSud

Comme son nom l'indique, il s'agit d'une classe qui hérite de la classe JPanel et se trouvera au sud dans plusieurs onglets de notre interface.

Elle définit des variables d'instances dont 3 boutons(JButton), une zone de texte (JLabel), un panel et une barre de progression.

Elle possède un seul constructeur :

- PanelSud(JLabel labErr) ;

Ce constructeur va instancier ces variables d'instances de classes :

3 bouttons (Retour,Modifier,Suivant) ; une zone d'affichage de message d'erreur à l'intention de l'utilisateur puis place les composants comme suite :

Figure 18 : Réalisation du PanelSud

III.6. La classe Sys_Cord2

Cette classe devra servir d'onglet dédié à la définition de la géométrie de la simulation.

Réalisation :

On se servira des classes précédentes pour créer les champs de saisie, les boutons, etc...

Elle définit des variables d'instances pour stocker les valeurs que l'utilisateur aura saisi ; il s'agit notamment de :

le cas du système Polaire :

* rho_Min, rho_Max, delta_rho
* Z_Min, Z_Max, delta_Z

le cas du système Cartésien :

* X_Min, X_Max, delta_X
* Y_Min, Y_Max, delta_Y
* Z_Min, Z_Max, delta_Z

Ainsi que les champs concernant le temps qui sont communs aux deux systemes.

Axe temporel :

* Ts_Min, Ts_Max, delta_Ts

Le panel permettant à l'utilisateur de faire le choix du système de coordonnée sera une instance de PanelRadioButon et sera affiché dans la partie Ouest de l'onglet.

Pour obtenir la représentation voulue lors de la conception, nous placerons les champs par trois(3) dans un panel (défini dans PanelLabelText) dont le LayoutManager est de type FlowLayout. Puis on regroupe ces instances de PanelLabelText selon le cas de figure 2 pour le système polaire, 3 pour le système cartésien. Quant au domaine temporel un seul PanelLabelText de trois composants suffira.

********** Instanciation des differents panel **********

```
choix = new PanelRadioButton(choix_Titre,2,choix_Str);
temps = new PanelLabelText(3,temps_Str, k, labErr);
repere_pol1 = new PanelLabelText(3,rho_Str,k1, labErr);
repere_pol2 = new PanelLabelText(3,z_Str,k2, labErr);
repere_cart1 = new PanelLabelText(3,x_Str,k2, labErr);
repere_cart2 = new PanelLabelText(3,y_Str,k2, labErr);
repere_cart3 = new PanelLabelText(3,z_Str,k2, labErr);
```

****

On regroupe repere_pol1 et repere_pol2 dans un panel partie1_1 géré par un GrideLayout(2,0) ;

repere_cart1, repere_cart2 et repere_cart3 dans un panel partie1_2 géré par un GrideLayout(3,0) .

par la suite, on place partie1_1 et partie1_2 dans un autre panel partie1 géré par un FlowLayout qui sera à son tour placé dans un JScrollpane « partie1_Sup».

Le panel Temps est déposé dans un JScrollpane nommé partie2.

- Vient après le panel dédié au test de stabilité nommé partie3 qui sera mis à l'interieur d'un JScrollpane « partie3_Sup».

En fin, on place les trois panels (partie1_Sup, partie2, partie3_Sup) dans un panel « panneau_Centrale» géré par un GrideLayout(3,0). Ce dernier sera placé au centre de la classe.

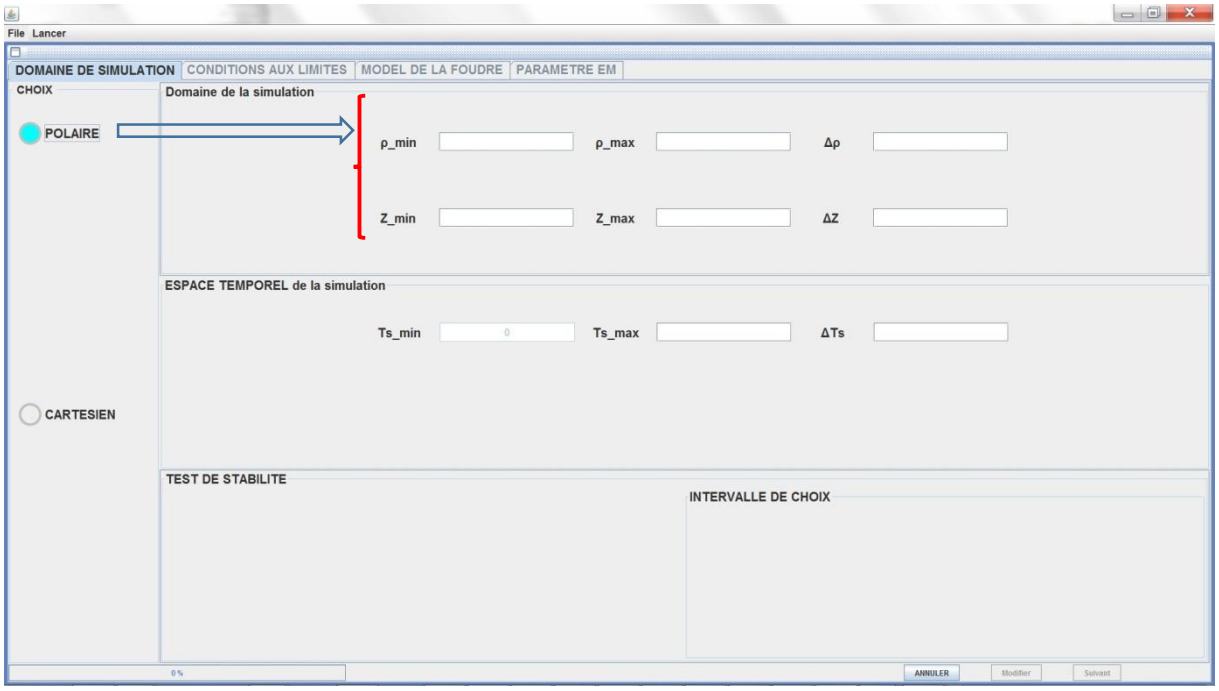


Figure 19 : Réalisation de l'onglet Domaine de simulation avec vu sur le cas Polaire

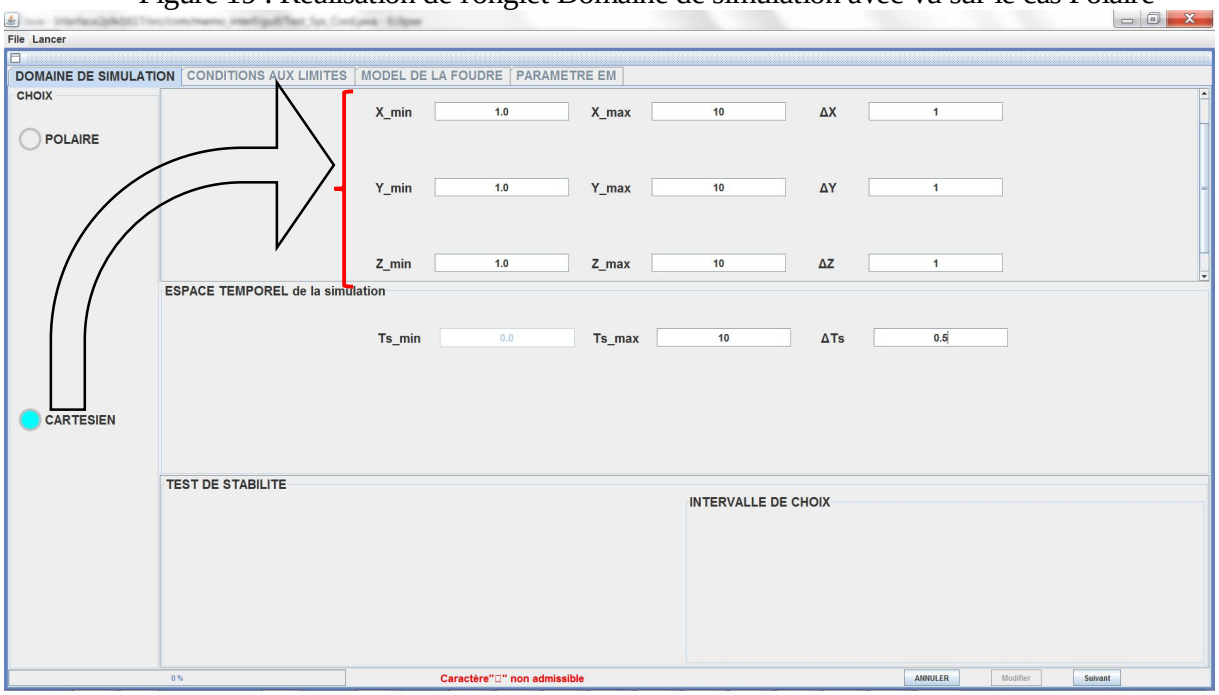


Figure 20 : Réalisation de l'onglet Domaine de simulation avec vu sur le cas Cartesien

- Pour finaliser le design de cet onglet, nous placerons une instance de la classe PanelSud panelSud au sud de la classe qui se chargera d'afficher les boutons «Annuler » «Modifier » «Suivant » ; la zone d'affichage des erreurs ainsi que le JProgressBar.

Notre classe Sys_Cord2 définit aussi quelques méthodes pour faciliter la gestion des interactions avec l'utilisateur :

- void Validation(Sys_Cord2 ex) ;

Cette méthode servira à valider et à affecter les valeurs saisies aux variables correspondantes.

- void actionAssoChampPanel(PanelLabelText j);

Elle permet d'associer à l'ensemble des champs de texte d'une instance de PanelLabelText un écouteur de type DocumentListener qui permet d'écouter les évènements de type insertion, suppression, changement d'un caractère.

La classe Sys_Cord2 contient aussi des classes imbriquées pour définir des actions associées aux évènements :

- La sous classe evenType :

```
class eventType implements ItemListener;
```

Cette sous classe définit les interactions de l'utilisateur avec le panel proposant le choix du système de coordonnée.

Elle possède un seul constructeur :

```
public eventType(Sys_Cord2 hy) ;
```

Implémenté depuis l'interface ItemListener.

Elle définit la méthode :

```
public void itemStateChanged(ItemEvent e);
```

La sous classe SuiviOnglet :

```
class SuivieOnglet implements DocumentListener ;
```

Cette sous classe gardera son constructeur par défaut, et elle va redefinir les methodes heritées de l'interface DocumentListener qui sont :

```
public void changedUpdate(DocumentEvent e) ;
public void insertUpdate(DocumentEvent e) ;
public void removeUpdate(DocumentEvent e) ;
```

Son rôle est de définir les actions à mener quand ces évènements se produisent sur les champs de saisie.

Elle est chargée de détecter si tous les champs de ce onglet sont remplis ou pas. Pour cela elle définit la methode VerifContTotal( ) qui retourne un booléen.

- boolean verifContTotal() ;

III.7. La classe Cond_Limit

Cette classe devra générer l'onglet dédié aux conditions limites de la simulation, comme établi lors de la conception, elle hérite de la classe JPanel :

public class Cond_Limit extends JPanel

- ❖ La classe définit aussi les variables d'instances suivantes:

- o Une instance de PanelSud « panelSud ».
- o Une zone d'affichage de texte « labErr » (JLabel) pour signaler les probables erreurs.
- o Une instance de PanelRadioButton «ex » pour créer les boutons permettant d'effectuer un choix ex = new PanelRadioButton("Conditions aux limites",3,Lst);
- o Un entier conditionLimite pour situer le choix de la condition limite.

Elle admet un constructeur sans parametre : Cond_Limit() ;

Cond_Limit contient aussi une sous classe class eventTypeCond implements ItemListener pour définir les actions à mener à la détection des évènements sur la sélection des différents boutons.

Cette sous classe eventTypeCond definit la methode :

public void itemStateChanged(ItemEvent e) de l'interface qu'elle implémente.

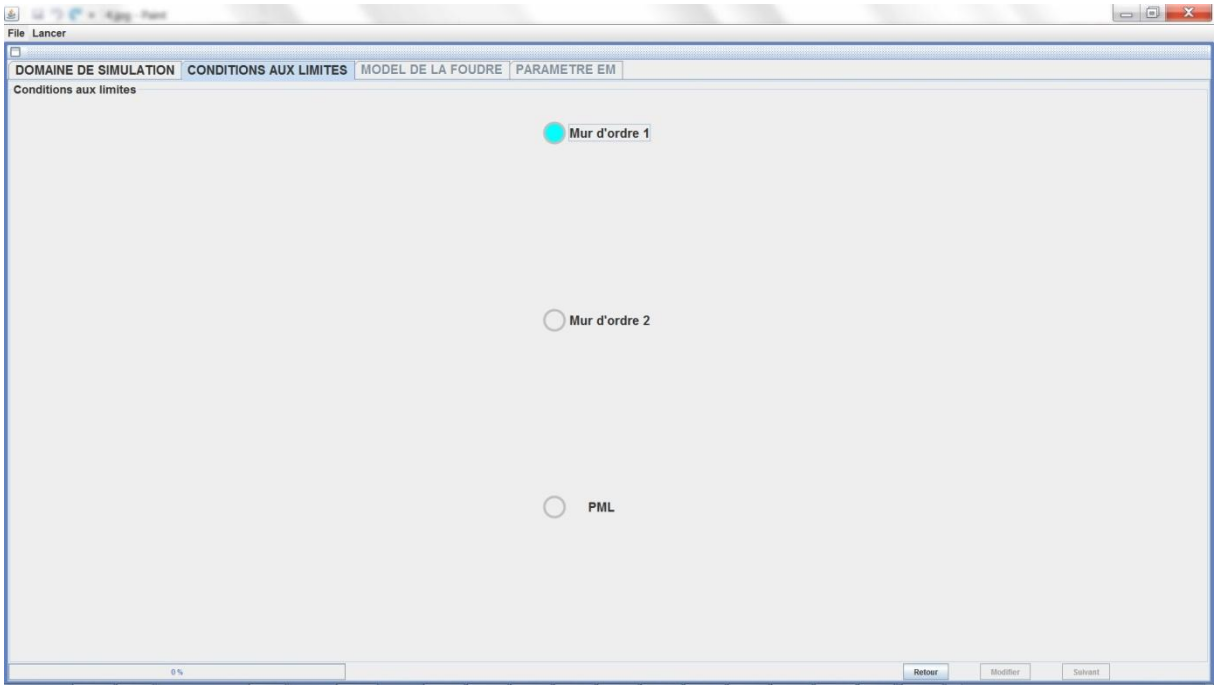


Figure 21 : Réalisation de l'onglet Conditions Limites

III.8. La classe Model_Foudre

Elle hérite de la classe JPanel et permet à l'utilisateur de faire un choix du model de la foudre ainsi que de saisir un certain nombre de paramètres concernant le courant.

Elle définit :

- ✓ PanelRadioButton(ex) et un(1) entier(modelFoudre) pour permettre et mémoriser le choix de la foudre
- ✓ 12 PanellabelText chaqu'un ayant un champ de saisie pour les données sur le courant de foudre et 12 variables de type Double pour mémoriser les valeurs saisies.
- ✓ 1 JLabel(labErr) pour l'affichage des erreurs
- ✓ 1 JScrollPane(param) pour permettre un affichage avec barre de défilement automatique
- ✓ 3 panels simples pour le placement
    - RegroupVariable ayant un GridLayout(12,0), dans lequel sont ajoutes les 12 PanelLabelText et lui-même placé dans le JScrollPane param.
    - panneauCentre ayant un GridLayout(0,2), pour permettre d'afficher longitudinalement le panel « ex » et « param » qu'elle va recevoir.

Les actions associées aux boutons radio de « ex » seront réalisées par l'intermédiaire d'ajout de class anonymes ItemListener() ; ces dernieres porteront sur un affichage des variables nécessaire en fonction du choix du model de la foudre. La classe associe aussi des actions sur les champs de saisie par l'instance de la classe imbriquée SuivieOnglet.

Cette classe définit aussi trois fonctions et une classe imbriquée :

- ✓ La méthode actionAssoChampPanel1 pour affecter les contrôles de saisie aux champs de saisie de l'onglet.
- ✓ La méthode Desactivation pour rendre inaccessible l'ensemble des champs de saisie
- ✓ La méthode Activation pour rendre accessible l'ensemble des champs de saisie sauf pour le champ qui est une constante.
- ✓ La classe imbriquée SuivieOnglet1 pour définir et implémenter les actions de vérification de la non nullité des contenu des champs de saisie de l'onglet.

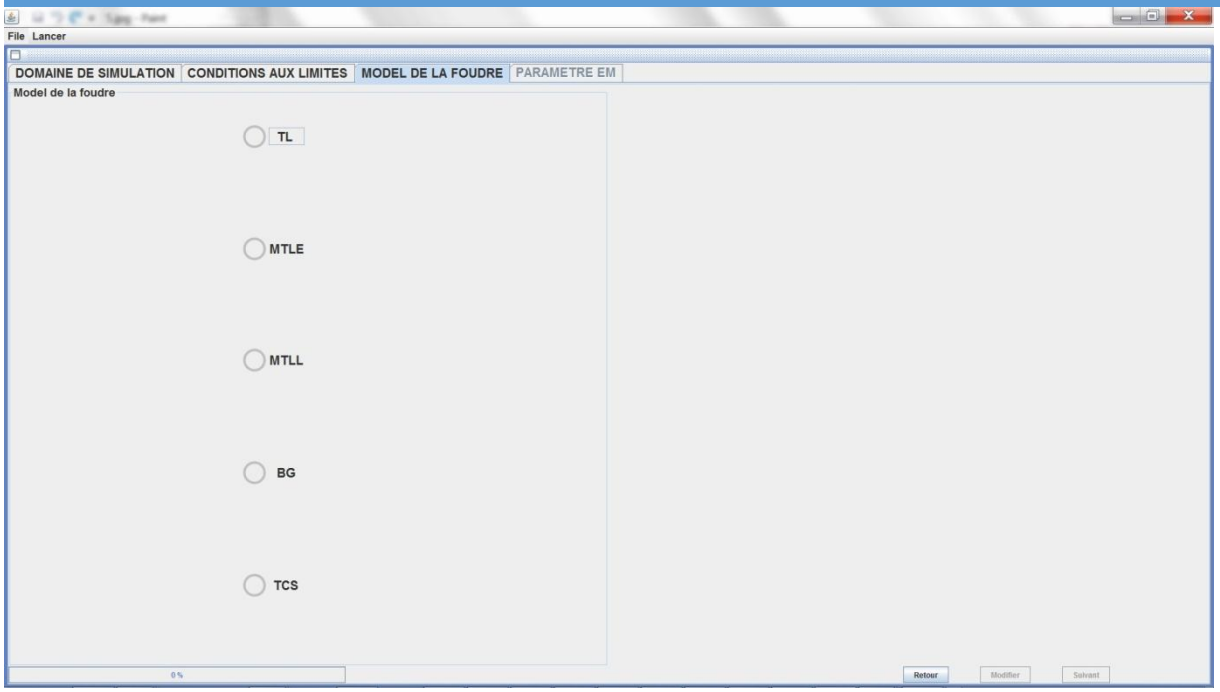


Figure 22 : Réalisation de l'onglet Model de la Foudre

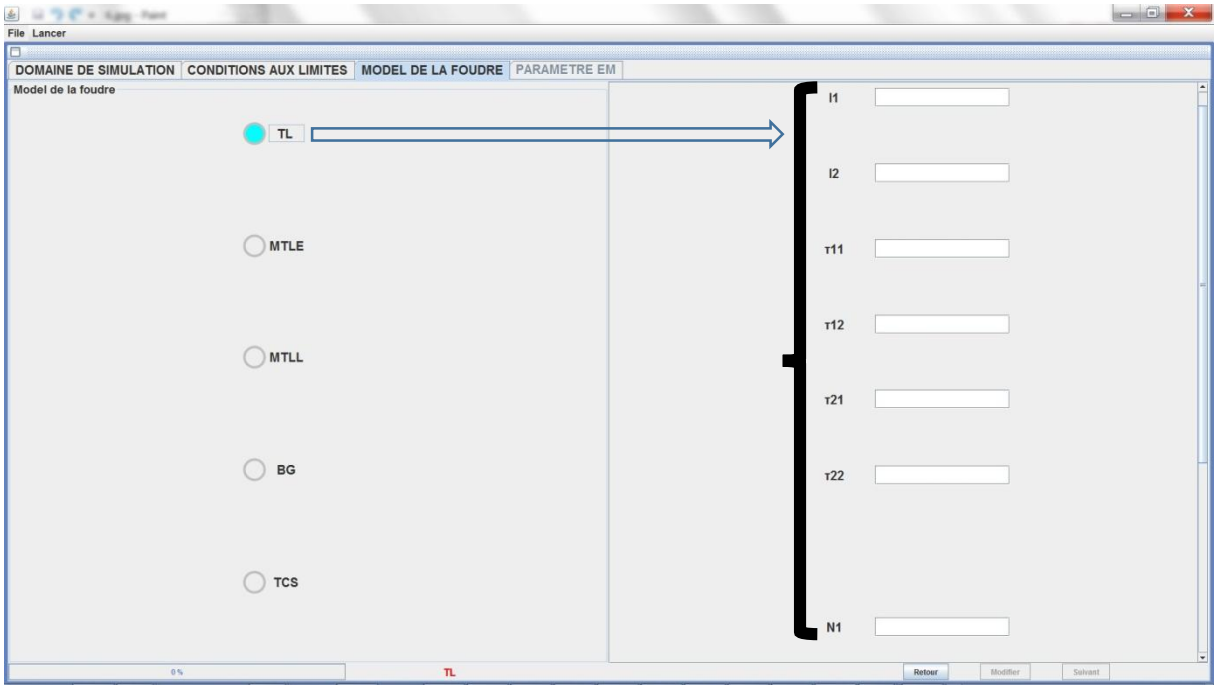


Figure 23 : Réalisation de l'onglet Model de la Foudre avec le choix de TL

III.9. La classe Param_EM

Cette classe génère l'onglet dédié aux paramètres électromagnétique pour cela, hérite de la class JPanel. Elle est gérée par un BorderLayoutManager ayant à son centre un Panel « nbrecouche » et une instance de PanelSud située à son sud.

Le panel nbrecouche ayant pour manager un GrideLayout(4,0) renferme 4 panels selon l'ordre :

- Le panel « txt » qui est une instance de la classe PanelLabelText contenant un seul champ de saisie permettant à l'utilisateur si nécessaire de saisir le nombre de couche et un bouton pour valider ce choix.
- Le panel « Stratification » qui est une instance de la classe PanelRadioButton avec deux boutons dépendantes permettant à l'utilisateur de faire un choix sur la stratification du milieu.
- Le panel « infoCouches » qui est un panel avec affichage si nécessaire des barres de défilement.

  Ce dernier contiendra un panel « interinfoCouches » qui à son tour affichera une série de panels de type PanelLabelText(un tableau) offrant à l'utilisateur des champs de saisie nécessaires aux renseignements sur les différentes couches.
- Le panel « Tour » permettant de signaler la présence ou l'absence d'une tour, est composé d'un Panel « tr » contenant un bouton sélectionnable et d'une instance de la classe PanelRadioButton « Tour_Model_Reflexion » visible dans le scénario de la présence d'une tour pour définir le model de réflexion de la tour.

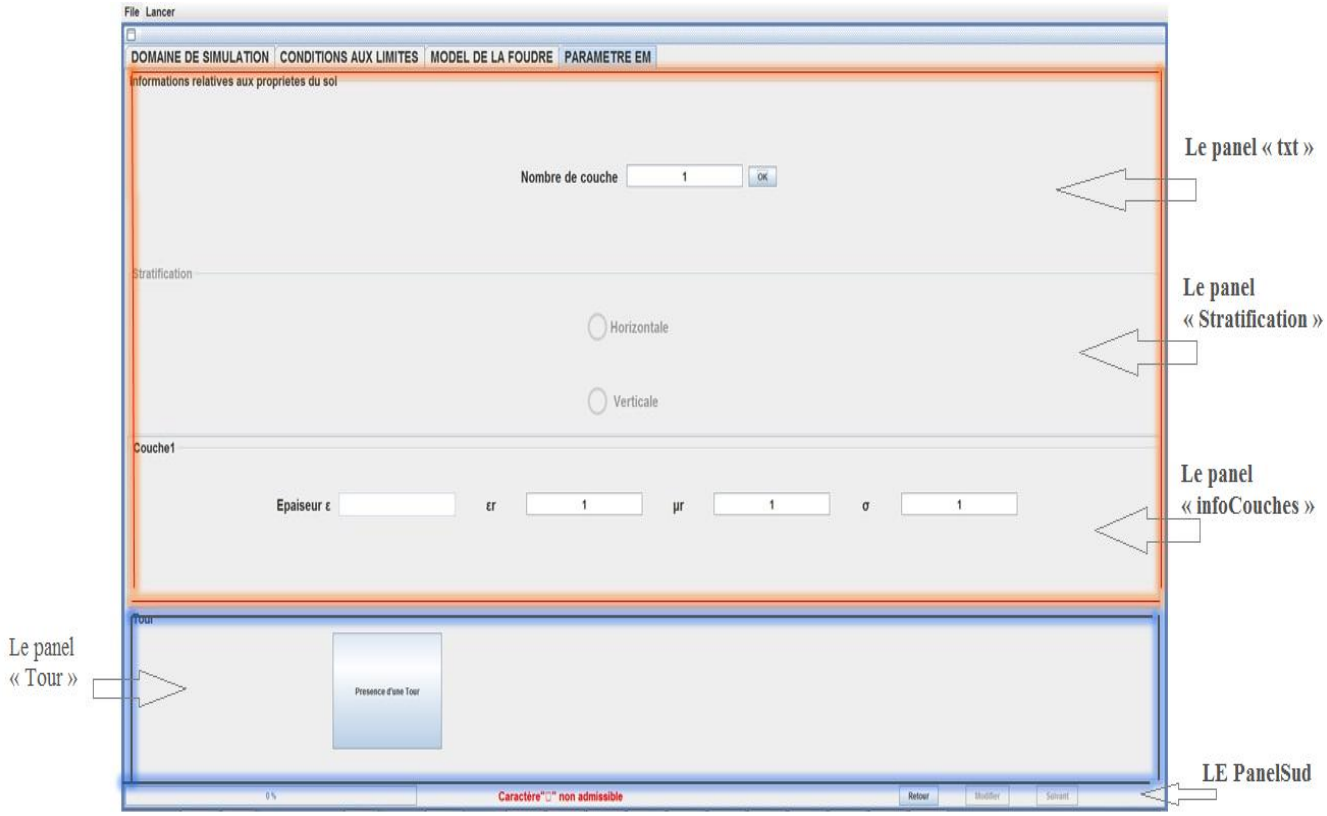


Figure 24 : Réalisation de l'onglet Paramètres EM avec signalisation des différents panels

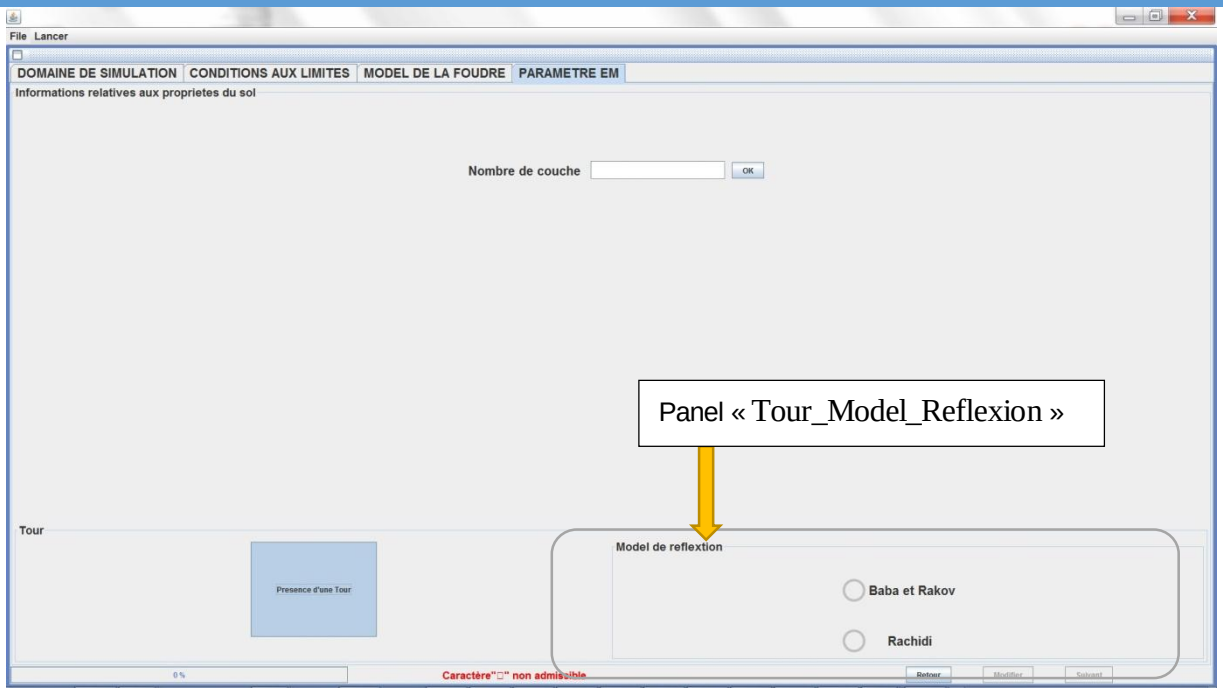


Figure 25 : Réalisation de l'onglet Paramètres EM avec le bouton présence de tour enfoncé

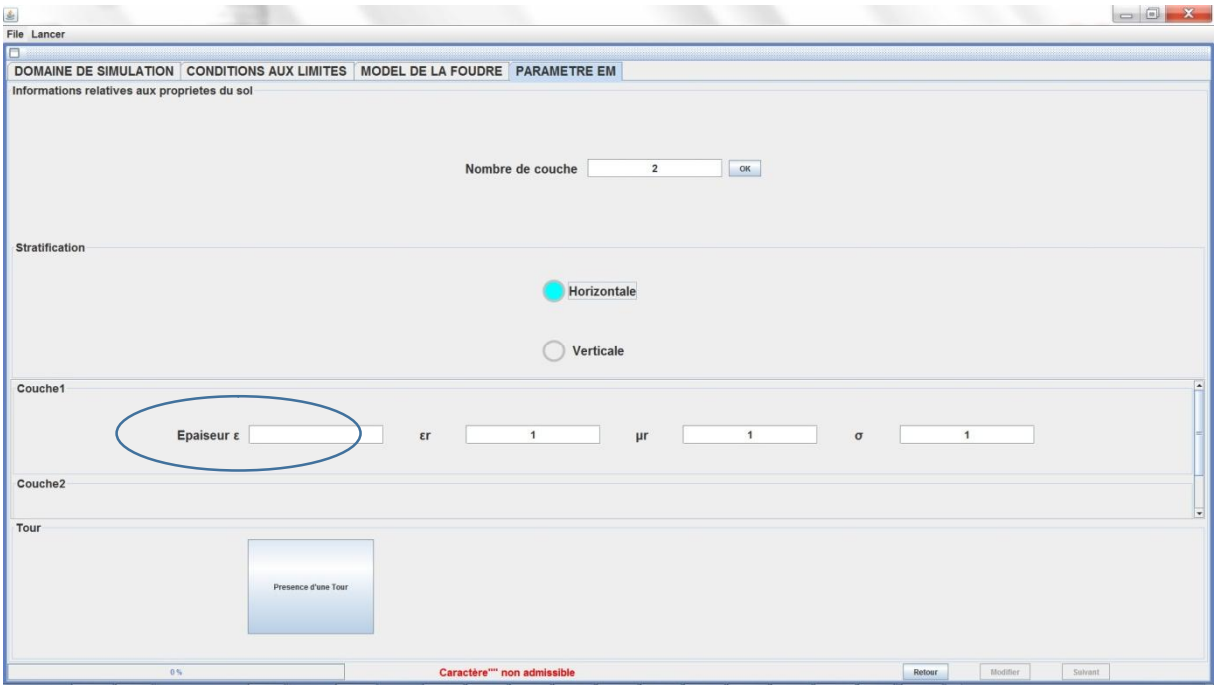


Figure 26 : Réalisation de l'onglet Paramètres EM avec le choix de stratification Horizontale

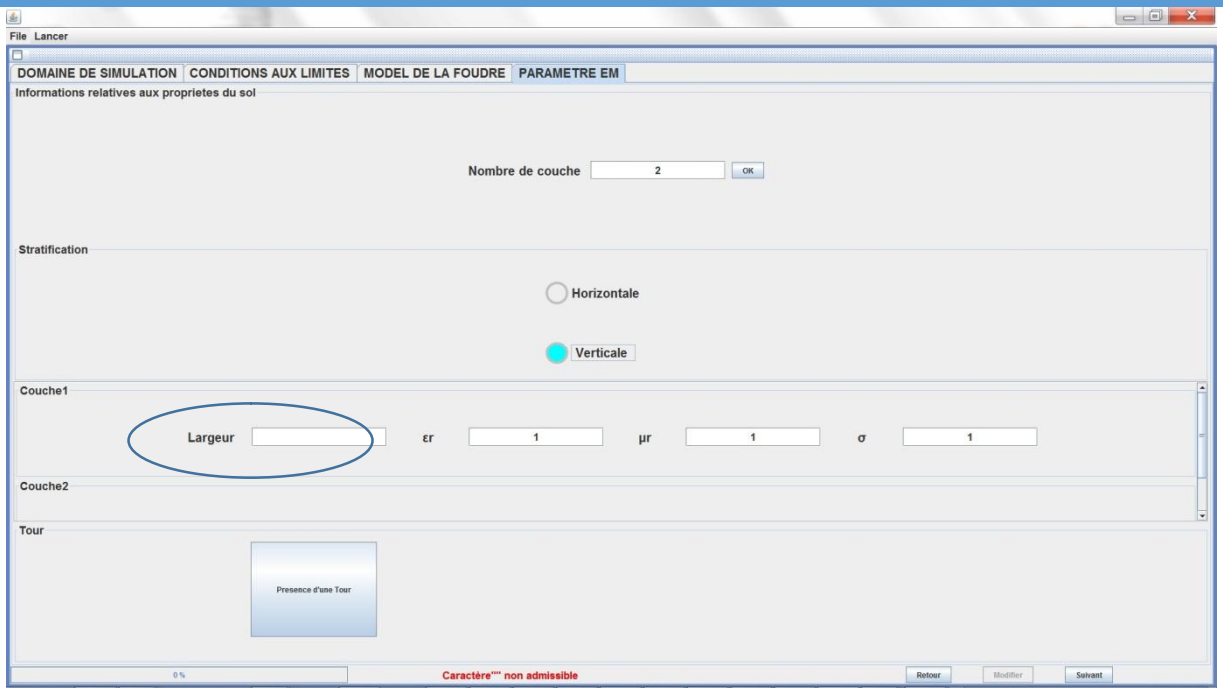


Figure 27 : Réalisation de l'onglet Paramètres EM avec le choix de stratification Verticale

En plus la classe Param_EM définit quelques méthodes et une classe imbriquée :

- Les méthodes :

❖ `Double sommeCouche(Param_EM ex) ;`

Pour calculer la somme des épaisseurs des couches.

❖ `List<Double> Validation(Param_EM ex) ;`

Pour récupérer dans un tableau de double l'ensemble des informations concernant les différentes couches.

❖ `void Desactivation (Param_EM ex) ;`

Pour rendre inaccessible l'ensemble des composants du panneau central.

❖ `void Activation(Param_EM ex) ;`

Pour rendre éditable l'ensemble des composants du panneau central.

❖ `boolean Test(PanelLabelText [] tab);`

Pour vérifier si tous les champs de saisie du Tableau de PanelLabelText sont remplis.

❖ `void actionAssoChampPanel2(PanelLabelText [] tab);`

Pour associer des écouteurs aux champs du Tableau de PanelLabelText et pour répondre à ces évènements.

- La classe imbriquée : class SuiviOnglet3

```
class SuivieOnglet3 implements DocumentListener ;
```

Cette sous classe gardera son constructeur par défaut, et elle va redéfinir les méthodes hérites comme dans le cas de la sous classe evenType ces trois méthodes héritées de l'interface DocumentListener qui sont :

```
public void changedUpdate(DocumentEvent e) ;
public void insertUpdate(DocumentEvent e) ;
public void removeUpdate(DocumentEvent e) ;
```

Son rôle est de définir les actions à mener quand ces évènements se produisent sur les champs de saisie. Elle est chargée de détecter si tous les champs de ce onglet sont remplis ou pas. Pour cela elle définit la methode `boolean verifContTotal() ;` qui retourne un booléen.

III.10. La classe Fenetre_Princip

Cette classe constituera le point d'entrée de l'application. Elle hérite de la classe JFrame, qui en soit est une fenêtre propre à Java. La classe Fenetre_Princip sera fenêtre principale de l'application.

Elle définit un menuBar composé d'un menu « File » comprenant deux menuItem :

« New » pour commencer un cas de simulation et « Exit » pour quitter l'application.

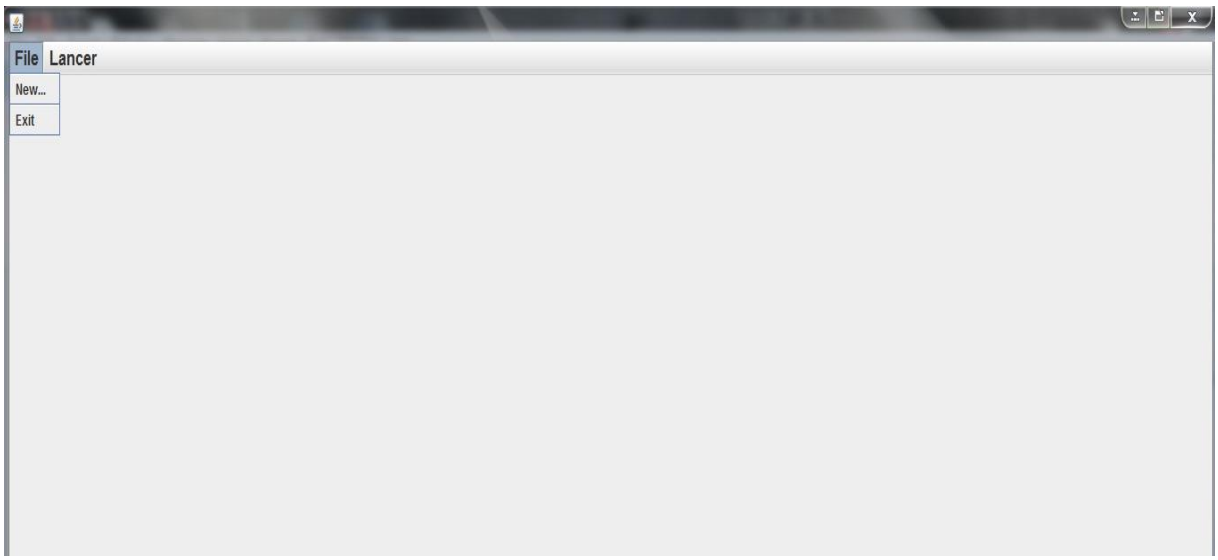


Figure 28 : Réalisation de la fenêtre principale avec vue sur le menuBar

La classe Fenetre_Princip definit une méthode « unCasSimul » retournant une fenêtre interne. Cette méthode sera associée au menuTem « New » pour lancer à chaque click cette dernière.

- Description de la méthode :

```
JInternalFrame unCasSimul() ;
```

Elle est dotée d'une instance finale de chacune des classes générant un onglet :

- `final Sys_Cord2 SystemCord;`
- `final Cond_Limit ConditionsLimites;`
- `final Model_Foudre ModelFoudre;`
- `final Param_EM ParametreElectromagnetique ;`
- `final JTabbedPane tabpane;`

Elle a une instance de JTabbedPane (classeur sous forme de pile) permettant d'organiser les panels (onglet) sous forme de volet à onglet.

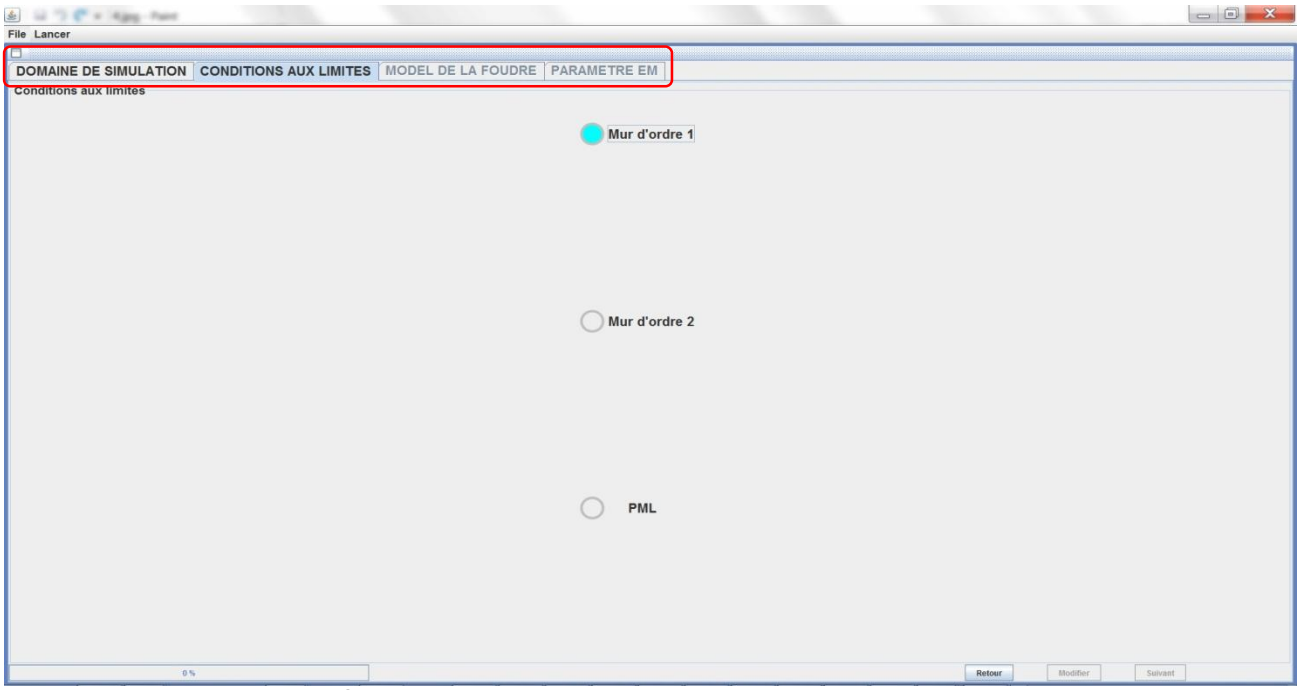


Figure 29 : Réalisation de la fenêtre principale montrant l'organisation des onglets

- Elle définit aussi une instance JIternalFrame « interieur » qu'elle va retourner lors de son appel.

```
final  JInternalFrame interieur = new JInternalFrame();
```

Apres avoir organisé les onglets dans le volet à onglet et ce dernier dans l'internalFrame, la méthode définira l'ensemble des actions et des vérifications associer à chaque boutons de chaque PanelSud de chaque Onglet.

- La Fenetre_Princip contient deux autres classes imbriquées :

1. `class ExFileChooser extends JFileChooser;`

permettant à l'utilisateur d'indiquer un dossier de sauvegarde des données.
Elle sera utilisée par la methode unCasSimul( ).

2. `class ecriture`

Ayant pour constructeur : `public ecriture(Path dos,String ecrit)`

Cette classe aura pour but de faciliter l'écriture dans des fichiers. Elle sera aussi utilisée par unCasSimul( ) pour écrire dans le fichier qu'il aura créé.

III.11. Les classes utilisées pour le traçage 2D

Pour le traçage, nous avons commencé à réutiliser une série de classe définie par Olly Oechsle, university of Essex, date 20-Nov_2017 qu'on a puis téléchargé sur le lien « hf@liafa.jussieu.fr ».
Nous avons effectué une description de ces méthodes et classes en fin de mieux les comprendre et pouvoir les utilise convenablement. (Voir Annexes).

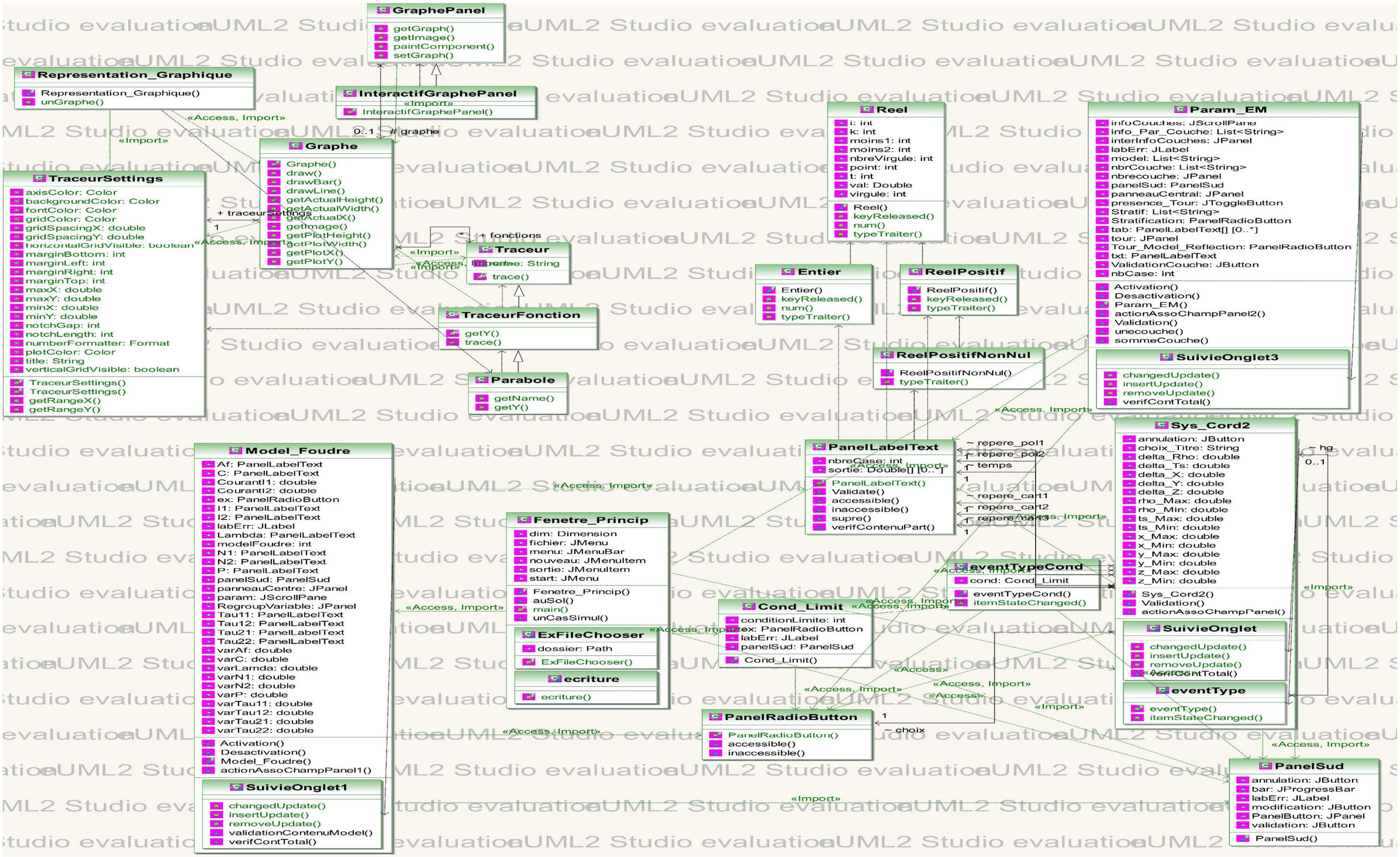


Diagramme 1: UML de l'application avec les relations

CONCLUSION ET PERSPECTIVES

Au terme de notre participation à ce projet un peu spécial pour notre spécialité, nous en sortons avec une expérience qui nous sera d'une très grande utilité dans notre parcours professionnel, sans parler de la possibilité de réutilisation des classes et méthodes que nous avons eu à développer au cours ce projet.
En effet, étant des futurs informaticiens industriels, nous serons amenés à programmer des systèmes industriels devant communiquer et interagir avec des utilisateurs peu familiers de l'informatique.
De ce fait, une interface différente des interfaces basiques facilitera grandement la formation des techniciens ou d'autres ingénieurs à l'exploitation de ces systèmes.
De plus ce projet nous a permis de faire notre entrée dans le monde de la conception d'application ainsi que celui de la programmation mixte, des mondes auxquels nous seront bientôt confrontés.
Ce fut une occasion de réviser, d'approfondir nos connaissances sur la programmation orienté objet et par la même occasion d'apprendre un nouveau langage de programmation « le langage JAVA ».

En fin, nous espérons que l'interface dont nous avons eu l'honneur de réaliser la base, sera enrichie par des fonctionnalités avancées telles que le lancement de la simulation sans que l'utilisateur n'ait à voir l'application fortran (par l'utilisation de la technologie de JNI); technologie qui permettrait d'ailleurs de rassembler les programmes Fortran dans l'interface graphique ; la mise en place d'un système gestion de base de données qui puisse gérer les résultats des simulations. C'est-à-dire sauvegarder au fur et à mesure les données et résultats des simulations dans une BD pour restituer ces résultats si l'utilisateur venait à introduire les mêmes données ; ce manœuvre aura pour but de minimiser le temps d'attente des résultants. Pourquoi pas si les moyens le permettent, d'emmagasiner les données et résultats de tous les possibles cas de simulation dans cette BD ; pour ensuite placer cette BD sur un serveur dans le but de pouvoir partager ces résultats dans le cadre d'une coopération inter universitaire sans partager le code source et par la même occasion faire gagner du temps à ceux qui utilisent les résultats . Nous souhaitons également si possible, que les responsables de ce projet fassent la proposition de poursuite de notre travail, à des étudiants depuis le master I, pour qu'ils puissent avoir le temps de bien assimiler le travail accompli et y apportent d'avantage d'amélioration.

RÉFÉRENCE BIBLIOGRAPHIQUE

Bibliographie

[1] Bien programmer en Java 7 EDITIONS EYROLLES ; 61, bd Saint-Germain 75240 Paris Cedex 05 WW.editions-eyrolles.com. (Pour la programmation)

[2] Myers B.A. User Software Tools. In TOCHI 2(1) 1995.

[3] Mickael Baron and Fréderic Martini "Exécuter du code natif en Java : JNI vs JNA publie le 27/2/2008 et mise à jour le 10/9/2013.

[4] Les Cahiers du programmeur Java 1.4 et 5 (ISBN 978-2-212-11916-9). (Pour la programmation)

Programmer en JAVA, préparation à la certification JAVA KAHLOULA BOUBAKER, les Editions Pages Bleues Internationales ; ISBN 978-9947-34-022-6 ; FEVRIER 2013. (Pour la programmation)

[5] Total Java 417 solutions express pour mordus de la programmation. (Pour la programmation)

[6] La thèse USTO 2007 « Analyse des problèmes de compatibilité électromagnétique par modélisation du rayonnement électromagnétique de la foudre » du Prof. Mimouni Abdendi à l'université IBN Khaldoun de Tiaret.

Webographie

http://jca.developpez.com/tutoriel/java/
http://baptiste-wicht.developpez.com/tutoriels/java/
https://www.developpez.net/
http://thierry-leriche-dessirier.developpez.com
https://openclassrooms.com/
http://blog.paumard.org/cours/java/
http://imss-www.upmf-grenoble.fr/prevert/Prog/Java/CoursJava/
https://www.ibisc.univ-evry.fr/
https://www.tutorialspoint.com/java/
http://www.java2s.com/Tutorial/Java/
http://perso.telecom-paristech.fr/
https://bugs.eclipse.org/
https://www.coursera.org/
https://www.jmdoudoux.fr/java/dej/

ANNEXES

Description des différentes classes de l'appli traceur

La classe démo 1

Classe de démonstration montrant l'utilisation des différentes classe à travers l'exemple d'une sinusoïdale et d'une parabole. Elle définit une fonction un Graphe retournant un objet de type Graphe qui sera passé en paramètre d'un objet « Traceur Swing » dans la fonction main de Démo. La fonction un Graphe définie deux objets (Traceur Selting, Graphe).

La classe sinusoïdale 2

Elle hérite de la classe Traceur Fonction, qui est une classe abstraite .donc elle définit les méthodes get Names () et get y ().

Sinusoïdale

Get y (double) ; double → pour définir les ordonnées

Get Name () ; String. → Pour définir les ordonnées

La classe parabole 3

Elle hérite de la classe Traceur Fonction, qui est une classe abstraire. Donc elle définit les méthodes get Name () et get y ().

Parabole

Get y (double) ; double / fonction retournant les valeurs de y.

Get Name () ; String.

La classe Traceur Selting 4

Cette classe définie l'ensemble des variables liées au panneau contenant le graphique (X_{min}, Xmax, Ymin, Y_{max}, margin Bolton (marge Basse), margin Top (marge Hante), margin eft (marge Gauche), margin Right (marge droite), axisColor (couleur de l'axe), plotcolor backgrandcolor, gridcolor fontcolor, notch Length (long-encoche) notchGap (aug de la distance entre l'encoche et le label),des loolearns(horizontalGridVisible, verticalGrid Visible) number, Format(Format),grid Spacing y, string Title)

La classe Traceur Fonction 5

Elle hérite de la classe Traceur, qui est une classe abstraite dont elle ne définit pas la méthode get Name () par conséquent la classe Traceur Fonction est une classe abstraite.

Elle définit la méthode get y () héritée de la classe Traceur ainsi que sa méthode trace (Graphe p, Graphies f, crit largeur, crit haut). La fonction trace est responsable de la tracée.

Traceur fonction :

Get y (double) : double

Trace (graphe, Graphies, int, int) : void

La classe Traceur 7

C'est une classe abstraite ayant deux fonctions abstraites get Name () et trace () à définir en cas d'héritage,

Traceur

- A get Name (), String
- A trace (Graphe, graphies, int, int)
- La classe Graphe panel 8

Cette classe hérite de la classe Jpanel et possède une variable de classe de type Graphe de visibilité protected. Et elle définitif les fonctions set Graphe (), get Graphe () pour accéder à la variable graph. Elle redéfinit la méthode Paint compomarty et une méthode get Imarge () pour sauvegarder l'image.

Graphe panel

- Graphe ; Graphe
- Set Graph (Graphe) ; void
- Paint component(Graphies) : void

- Get Image () ; Buffered Image
- Get Graph () ; Graphe

La classe Interactif Graphe Panel 9

Elle hérite de la classe Graphe Panel et définie les évènements liés au panel du graphique Interactif Graphe panel

Codage de l'alphabet Grec

PDF : fr [archive] en [archive] v · d · m	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
037	□	□	□	□	ʹ	͵	□	□			ͺ	ͻ	ͼ	ͽ	;	□
038					΄	΅	Ά	·	Έ	Ή	Ί		Ό		Ύ	Ώ
039	ΐ	A	B	Γ	Δ	E	Z	H	Θ	I	K	Λ	M	N	Ξ	O
03A	Π	P		Σ	T	Y	Φ	X	Ψ	Ω	Ϊ	Ϋ	ά	Ϊ	Ϋ	ί
03B	ΰ	α	β	γ	δ	ε	ζ	η	θ	ι	κ	λ	μ	ν	ξ	ο
03C	π	ρ	ς	σ	τ	υ	φ	χ	ψ	ω	ϊ	ϋ	ό	ύ	ώ	□
03D	ϐ	ϑ	ϒ	ϓ	ϔ	ϕ	ϖ	ϗ	Ϙ	ϙ	Ϛ	ϛ	Ϝ	ϝ	Ϟ	ϟ
03E	Ϡ	ϡ	Ϣ	ϣ	Ϥ	ϥ	Ϧ	ϧ	Ϩ	ϩ	Ϫ	ϫ	Ϭ	ϭ	Ϯ	ϯ
03F	ϰ	ϱ	ϲ	ϳ	ϴ	ϵ	϶	Ϸ	ϸ	Ϲ	Ϻ	ϻ	ϼ	Ͻ	Ͼ	Ͽ

Tableaux de correspondance entre les types de JAVA et ceux du langage natif

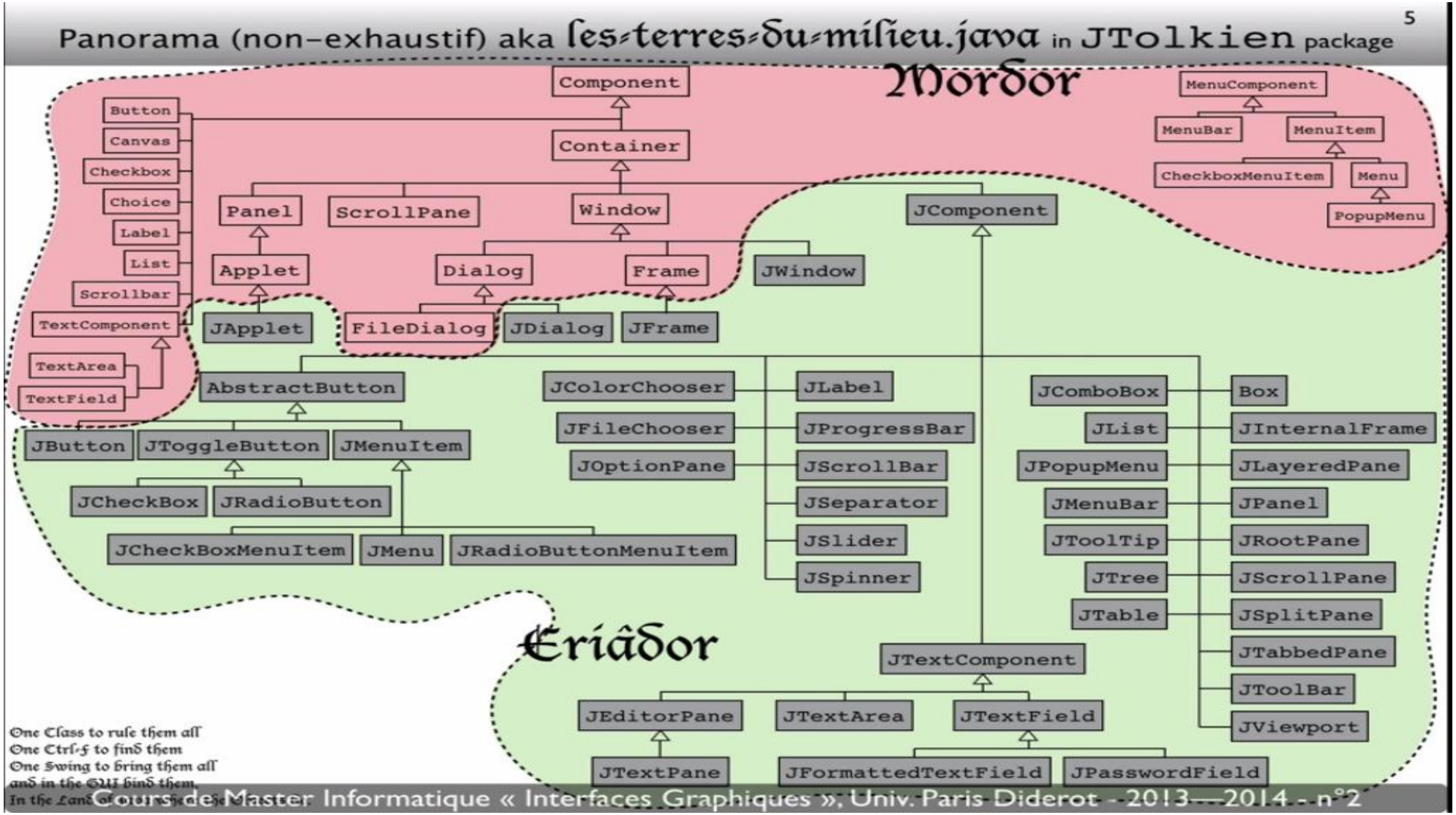
Primitive Java	Type natif
boolean	jboolean
byte	jbyte
char	jchar
double	jdouble
int	jint
float	jfloat
long	jlong
short	jshort
void	void

Objet C	Objet Java
jobject	java.lang.object
jstring	java.lang.String
jclass	java.lang.Class
jthrowable	java.lang.Throwable
jarray	type de base pour les tableaux
jintArray	int []
jlongArray	long []
jfloatArray	float []
jdoubleArray	double []
jobjectArray	Object []
jbooleanArray	boolean []
jbyteArray	byte []
jcharArray	char []
jshortArray	short []

Le tableau suivant présente un aperçu du Data mapping (Le **data mapping** est un procédé permettant de définir au niveau d'un langage de programmation la correspondance entre deux modèles de données) entre Java et le code natif pris en charge par la bibliothèque JNA.

Native Type	**Size**	**Java Language Type**	**Common Windows Types**
char	8-bit integer	byte	BYTE, TCHAR
short	16-bit	short	short WORD
wchar_t	16/32-bit character	char	WCHAR, TCHAR
int	32-bit integer	int	DWORD
int	boolean value	boolean	BOOL
long	32/64-bit integer	NativeLong	LONG
long long, __int64	64-bit integer	long	
float	32-bit FP	float	
double	64-bit FP	double	
char*	C string	String	LPCTSTR
void*	pointer	Pointer	LPVOID, HANDLE, LPXXX

Une représentation non exhaustive des classes Swing



RESUME

L’objectif du projet décrit dans ce document est la réalisation d’une interface graphique facilement maintenable en langage Java intégrant un ou des programmes fortran qui traitent de la simulation des effets électromagnétiques de la foudre sur son environnement. Nous abordons, premièrement l’intérêt, voir la nécessité d’une interface IHM (Interface Homme Machine) pour ces programmes ; deuxièmement les différentes méthodes pour parvenir à intégrer (relier l’interface Java au programme Fortran) le code Fortran ; et dans la Troisième partie est consacrée au design (la conception) et la réalisation de base de la dite l’interface. Le document se termine avec une synthèse des résultats trouvés et les perspectives visant à améliorer l’interface obtenue.

The objective of the project described in this document is the realization of a graphical interface easily maintainable in Java language integrating one or more Fortran programs that deal with the simulation of the electromagnetic effects of the lightning on its environment. First, we discuss the need for an HMI (Human Machine Interface) interface for these programs; Secondly the different methods to integrate (linking the Java interface to the Fortran program) the Fortran code; And in the Third part is devoted to the design and the realization of the so-called interface. The document ends with a synthesis of the results found and the prospects to improve the interface obtained.