James Manley
09/07/04
v 1.0

Spécifications techniques

du projet

"manipulation du genome"

de Guide

Table des matières

I) Introduction

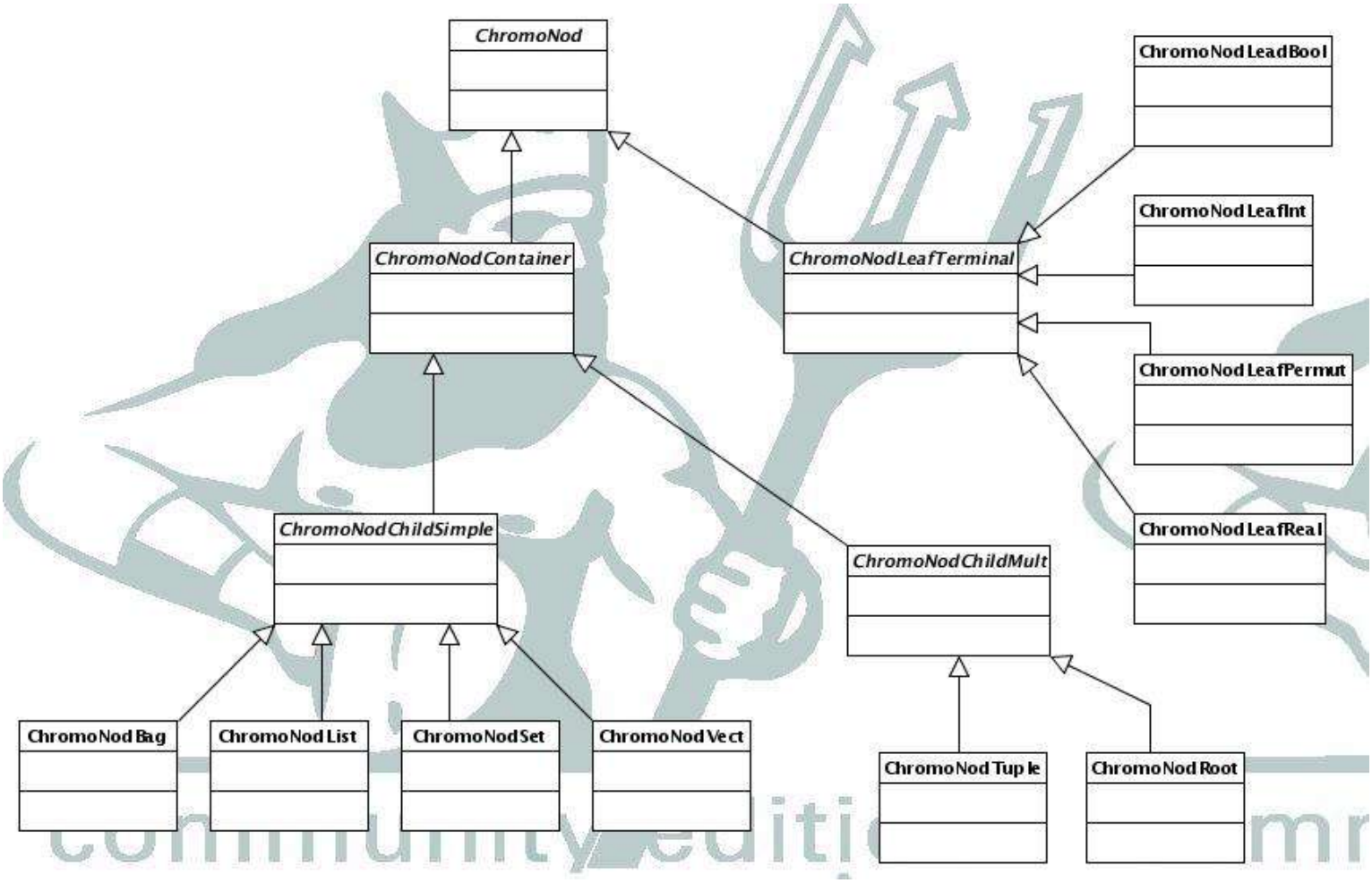
Ce document décrit l'implémentation de la partie "manipulation du genome" de l'application Guide. Guide est un programme qui génère des algorithmes évolutionnaire à partir de paramètres choisit par l'utilisateur. L'utilisateur choisit ces paramètres sur une interface graphique. Une partie préexistante à ce projet permet de définir les paramètres propres au moteur évolutionnaire. Ces paramètres définissent les paramètres de sélection et de remplacement. De même que les paramètres pour les algorithmes en îles. Chaque île a une population d'individus manipulé par un algorithme évolutionnaire. Des individus peuvent passer d'une île à une autre. Ces paramètres concerne les choix des parents lors de croisement, du taux de mutation et de croisement, la méthode de selection, la migration d'individus entre îles etc.. La partie qui est implanté par ce projet, appelé "manipulation du genome", concerne la structure et les opérations sur le génome. La structure est une forme en arbre où chaque nœud est un type conteneur et chaque feuille est un type de base. Les deux sont appelé chromosome. Cette structure est construite par l'utilisateur (sous partie création-structure). L'utilisateur peut choisir des opérateurs, d'initialisation, de mutation et de croisement sur ce genome, en choisissant ceux qui s'exerce sur chaque chromosome. Ces opérateurs sont soit prédéfinit, soit définit par l'utilisateur. Dans ce dernier cas, l'utilisateur doit tapper le code de l'opérateur. L'utilisateur devra coder une fonction d'évaluation pour créer l'exécutable. Une fois le genome et tous les opérateurs choisit, l'utilisateur peut compiler et créer un exécutable répondant aux paramètres voulus. Pour cela, il lui suffit d'appuyer sur un bouton"generate", un bouton"build" et un bouton "run". Ce document décrit l'architecture, les classes, leur héritage, leur associations, leurs méthodes et leurs attributs.

Le decoupage repose sur les diverses fonctionnalités des classes. Une distinction doit être faite entre la partie création-structure du génome et la partie opérations (initialisation, mutation et croisement). Une autre distinction est faite pour mettre en évidence la classes de chargement/sauvegarde et les classes de generation de code eo. Le genome du projet est définit par la partie création-structure. Les autres classes (opérations) s'appuient sur les classes création-structure. La partie opération a trois composants: une partie initialisation, une partie mutation et une partie croisement. Une classe vient s'ajouter pour prendre en charge la sauvegarde/chargement. Et dix classes prennent en charge la génération de code EO. Une partie de de la génération de code EO est sous traité par les classes opérateurs et classes structure - création et l'autre par les classes GeneratorXXX.

II) classes création-structure

Le sous ensemble principale correspond à la classe ChromoNod et ses sous classes. Ce sont les classes du modèle, qui contiennent les donnée et les méthodes pour gérer la création - structure. Les classes sélections permettent de créer des objets ChromoNod en réponse à l'utilisateur (action directe ou chargement). Il s'agit la de la vue. Les classes descendant de GraphChromo sont spécialisé pour l'affichage d'un chromosome à l'écran. Il s'agit la aussi de la vue (et du contrôle).

A) les classes de structure(modèle)



La classe abstraite ChromoNod est la racine du génome, elle généralise toutes les méthodes qui lui sont applicables. Dans l'arbre d'héritage, une première distinction est faite entre les classes de bases (ChromoNodLeafTerminal) et les conteneurs (ChromoNodContainer). Différentes méthodes abstraites sont implantés à ce niveau. Les conteneurs se divisent en deux parties, les conteneurs à enfants simples (ChromoNodChildSimple) et les conteneurs à enfants multiples (ChromoNodChildMult). Beaucoup de méthodes et l'attribut enfant(s) dépendent du nombre d'enfants autorisés (simple ou multiple). C'est pour cela que ces classes sont utiles. Jusque ici nous avons des classes abstraites, mais en dessous de ChromoNodChildSimple, ChromoNodChildMult et ChromoNodLeafTerminal nous avons des classes concretes. Ces classes implémentent un chromosome qui intervient dans la construction du génome.

La il existe divers types de chromosome: le type de base et abrégé, le type long (type donnant le nom du type en entier, destiné à être affiché), le type complet avec les paramètres(du chromosome), le type CPP donnant le nom de type pour le fichier c++, le type CPP long qui évite le type d'etre confondu avec un autre dans un namespace, le type

unique qui tiens compte des descendants et le type général (ressemble au type CPP mais est destiné a entrer dans les noms de variables). Un type de chromosome est une chaîne de caractères, qui peut être affiché, servir de séparateur entre opérateurs définit par l'utilisateur ou servir de type pour la génération de code EO (c++). Les opérateur sont unique à un type de chromosome. Il est impossible de mélanger les opérateurs de types de base différents. Le type abrégé est le nom du type sous forme condensé et est le type le plus courrament utilisé. Le type long est le nom du type sous sa forme longue et il est aussi destiné a être affiché. Le type complet prend en compte les paramètres du chromosome en plus du nom de type long pour l'affichage. Le type unique prends en compte le type des descendants. Si le type unique change (un descendant à été modifié), un opérateur définit par l'utilisateur peut devenir erronée. Le type unique sert à gérer cela. Une opération peut être importée d'un type unique vers un autre, sous la responsabilité de l'utilisateur. Les types CPP sont la chaîne de caractères du type du chromosome généré en EO. Le type CPP long ajoute un espace et un double deux points " ::" devant le type CPP, pour éviter la confusion avec d'autres types ayant même nom, qui sont importés des librairies et venant d'un autre namespace. Le type général est basé sur le type CPP mais est destiné à entrer dans un nom de variable. Donc si un même type à deux classes (tuple1 et tuple2), le type général sera de nom de la classe. Certains types (type unique et type CPP et CPP long) contiennent le nom type de leur descendants, ce n'est pas le cas du type général (bag<bool>).

Les chromosomes de type vect, liste, bag et set sont des conteneurs à enfants simple (unique) même si ils contiennent plusieurs éléments du même type et ayant le même nom. Dans ce cas l'enfant n'est pas un élément mais une liste d'éléments. Pour parcourir tous les éléments il faut parcourir la liste des éléments. Cette liste est généré par Guide en c++ comme un descendant de eoAbstractVector. Cette classe abstraite est implémenté pour cette raison et généralise des comportement des quatre type. La classe eoAbstractVector hérite de stl::vector. Certains types de chromosomes (vect et bag) ont une taille fixe non modifiable. Aucune instruction (code C++ généré), d'un opérateurs prédéfinit, ne doivent en modifier la taille. Par contre l'utilisateur peut coder une tel instruction pour une opération qu'il définit. Il est seul responsable du code qu'il génère.

Il faut signaler que des méthodes gèrent du code EO. Et en fin d'autres méthodes gèrent les liens entre des opérateurs et le chromosome. On peut ajouter, supprimer et changer (le remplacer par un autre) un opérateur sur un chromosome. Il existe un

opérateur par défaut, qui peut être null (pas d'opérateur par défaut, à éviter!). Cet opérateur est ajouté à chaque création d'un chromosome ou si un chromosome n'a plus d'opérateur. Cependant lors du chargement, le chromosome est crée avant que les opérateurs ne soient chargées. Il faut donc effacer les opérateurs par défaut pour les remplacer par les opérateurs chargées.

1) la classe ChromoNod

a) les attributs:

- protected String name : Nom donné par l'utilisateur au chromosome. Lors de la compilation, ce nom doit être unique.

- protected String cppType : C'est le type du chromosome dans le code EO généré.

- protected String longCppType : C'est le meme type que cppType, à la différence que cppType peut être confondus avec une autre classe, d'une des libraires EO et STL. L'attribut cppType met une portée sur le type (" ::").

- private ChromoNod parentNod : Lien vers le chromosome parent. C'est ancêtre du chromosome dans l'arbre. Le chromosome racine n'a pas de parents la valeur de cet attribut est alors null.

- private ExamineOperModel operModel : Quand une modification est faite sur un chromosome, les opérateurs définis par l'utilisateur peuvent devenir incorrects, s'il se trouvent sur le chromosome ou sur un de ces ancêtres. Donc les modification se propagent vers les parents. Il demander à l'utilisateur s'il veut ou non garder les opérateurs qu'il à définit sur ces chromosomes. Or le dialogue qui demande ceci est dans la vue, donc il faut une classe qui soit dans le modèle et qui lance des événements pour le dialogue quand une demande est posé à l'utilisateur. Cela à été fait avec le modèle MVC.

- private Vector vectInit : liste des opérateurs d'initialisation sur ce chromosome.

- private Vector vectMut : liste des opérateurs de mutation sur ce chromosome.

- private Vector vectCross : liste des opérateurs de croisement sur ce chromosome.

- static private Vector vectNames : attribut static contenant tous les noms de

chromosomes.

b) les méthodes non abstraites:

- static public ChromoNod clone(ChromoNod chromo) duplique le chromosome.

- static public String getValideName() : retourne un nom de chromosome n'ayant pas encore été employé.

- public ChromoNod(ChromoNod prev, String name) throws ExpNameIncorrect : constructeur. Le paramètre prev est le chromosome parent. Le chromosome est marqué comme enfant de sont parent. Le paramètre name est le nom du chromosome. Les espaces sont remplacées par des '_'. Ce nom est vérifié, il ne faut pas qu'il contienne de caractères interdits. Les opérateurs par défaut lui sont ajoutés (il n'a aucun opérateur pour le moment). Un objet operModel est crée (cf attribut operModel).

- public ExamineOperModel getExamineOperModel() : retourne l'objet operModel (cf attribut operModel).

- private void addName(String name) throws ExpNameIncorrect : verifie qu'un nom a bien été passé en paramètre. Il ne faut pas que cette variable soit null. Que ce nom est correctement tapé. Il sera utilisé pour créer des noms de variable en C++, il ne faut pas qu'il contienne de mauvais caractères ou des espaces. Il est ajouté à l'attribut vectNames avant d'etre affecté au nom du chromosome name.

- public ChromoNod getParent() : retourne le chromosome parent.

- public boolean isParent(ChromoNod chromo) : retourne vrais si le chromosome en paramètre est sont parent. (Et faux sinon)

- public void addParentNod(ChromoNod prev) : Le chromosome en paramètre est mis comme parent. Le chromosome this est mis comme l'enfant de son parent (si cela n'est pas deja le cas).

- public void removeParentNod(ChromoNod prev) : Retire prev comme étant le parent de this. Le chromosome this n'a plus de parents et prev n'a plus this comme enfant.

- public String getGeneralType() : Retourne le type général. Ce type va servir dans le nom de variables généré par un type. Si ce type génére plusieurs classes (tuple1 et tuple2), ce type est le nom de la classe. Ce type ne tiens pas compte des types des enfants.

- public String getCPPType() : Retourne le type CPP. Si le type CPP n'a pas encore été

affecté, setCPPType est appélé. Le type CPP est le nom du type généré du chromosome, dans le code généré EO.

- public String getLongCPPType() : Retourne le type CPP long. Si le type CPP et CPP long n'ont pas encore été affectés, setCPPType est appélé. Le type CPP est le nom de la classe, dans le code généré EO, du chromosome. Et le type CPP long est composé de double point devant chaque type CPP. Le type CPP long évite les ambiguitées avec d'autres librairies (d'autres namespaces).

- public void reinitCPPType() : reinitialise le type CPP. Le type CPP choisit n'est plus valide. Le prochaine appel de getCPPType ou getLongCPPType, lancera un appel à setCPPType. Un nouveau type CPP et type CPPLong sera choisit. La méthode reiniCPPType s'exécute récusivement sur tous les antécédents.

- public String getName() : retourne le nom du chromosome.

- public void update() : Des modification ont été faites sur cet objet (du modèle), il faut réactualiser l'interface (la vue), pour que l'utilisateur puisse voir ces modifications. Un événement est lancé. Nous sommes dans le modèle MVC.

- public Object clone() throws CloneNotSupportedException : duplique l'objet.

- public Vector getChromoInit() : Retourne une copie de la liste des opérateurs d'initialisation sur ce chromosome.

- public void addChromoInit(ChromoInit_Root chromoInit) : ajoute un opérateur d'initialisation à ce chromosome. L'opérateur reçoit une notification du chromosome sur lequel il est ajouté.

- public void removeChromoInit(ChromoInit_Root chromoInit) : L'opérateur d'initialisation est retiré du chromosome. La méthode verifyDefaultValues() est lancée pour verifier qu'il y a bien au moins un opérateur d'initialisation.

- public void changeChromoInit(ChromoInit_Root oldCh, ChromoInit_Root newCh) : remplace un opérateur d'initialisation par un autre (l'ordre est respecté).

- public Vector getChromoMut() : Retourne une copie de la liste des opérateurs de mutation sur ce chromosome.

- public void addChromoMut(ChromoMut_Root chromoMut) : ajoute un opérateur de mutation à ce chromosome. L'opérateur reçoit une notification du chromosome sur lequel il est ajouté.

- public void removeChromoMut(ChromoMut_Root chromoMut) : L'opérateur de mutation est retiré du chromosome. La méthode verifyDefaultValues() est lancée pour verifier qu'il y a bien au moins un opérateur de mutation.

- public void changeChromoMut(ChromoMut_Root oldCh, ChromoMut_Root newCh) : remplace un opérateur de mutation par un autre (l'ordre est respecté).

- public Vector getChromoCross() : Retourne une copie de la liste des opérateurs de croisement sur ce chromosome.

- public void addChromoCross(ChromoCross_Root chromoCross) : ajoute un opérateur de croisement à ce chromosome. L'opérateur reçoit une notification du chromosome sur lequel il est ajouté.

- public void removeChromoCross(ChromoCross_Root chromoCross) : L'opérateur de croisement est retiré du chromosome. La méthode verifyDefaultValues() est lancée pour verifier qu'il y a bien au moins un opérateur de croisement.

- public void changeChromoCross(ChromoCross_Root oldCh, ChromoCross_Root newCh) : remplace un opérateur de croisement par un autre (l'ordre est respecté).

- protected void setChangedValue(boolean b) : Si le type unique (le type du chromosome ou un de ses descendants) change, il faut verifier qu'aucun opérateur définit par l'utilisateur n'a été utilisé. Si c'est le cas on marque l'objet operModel avec le chromosome et l'opérateur, pour que l'utilisateur decide si oui ou non il veut retirer cet opérateur du chromosome. Comme l'utilisateur est libre d'utiliser les enfants dans la définition de ses opérateurs, si les enfants changent, l'opérateur peut devenir incorrect. Si b est vrais le type unique est considéré comme ayant été modifié et se propage récursivement vers les parents.

- private void verifyDefaultValues() : Vérifie qu'il y a au moins un opérateur de chaque sorte (initialisation, mutation et croisement). Si c'est pas le cas l'opérateur par défaut (quand il existe) est installé avec un poids de un.

- public void removeDefaultValues() : Lors du chargement, le chromosome est crée avant que ses opérateurs lui soient affféctés. Or lors de cette création, les opérateurs par défauts lui sont affectés. Pour les retirer et charger les opérateurs normalement, il faut appeler cette méthode.

- public static void reinitialise() : Vide la liste des noms de chromosome. Cette méthode

est appelé lors du chargement, les anciens noms sont vidés de la liste.

- public void isCorrect() throws ExpNameInDouble, ExpContainerNoChild : Vérifie que chaque chromosome à un nom unique et qu'un chromosome qui n'est pas un type feuille, à au moins un enfant. Cette méthode est la partie publique. Elle appelle isCorrrectRec().

- private void isCorrectRec(Vector names) throws ExpNameInDouble, ExpContainerNoChild : Parie récursive et privé de isCorrect().

c) les méthodes abstraites:

- abstract public Vector getChildren() : retourne la liste des enfants du chromosome dans l'ordre de création.

- abstract public Vector getChildrenAlphaOrder() : retourne la liste des enfants du chromosome dans l'ordre alphabétique des noms des enfants.

- abstract public void changeChildren(ChromoNod oldCh, ChromoNod newCh) : remplace un enfant par un autre.

- abstract public boolean isChild(ChromoNod chromo) : retourne vrais si this est un enfant de chromo et retourne faux sinon.

- abstract public void addChildNod(ChromoNod next) : ajoute un enfant au chromosome.

- abstract public void removeChildNod(ChromoNod next) : retire un enfant du chromosome.

- abstract public boolean isLeaf() : retourne vrais si le chromosome est un type de feuille. Il ne peut avoir d'enfants.

- abstract public String getType() : retourne le type de base du chromosome.

- abstract public String getLongType() : retourne le type long du chromosome (le type est sous sa version longue).

- abstract public String getFullType() : retourne type contenant les paramètres du chromosome.

- abstract public String getUniqueType() : retourne le type unique du chromosome. Ce type prends en compte le type des enfants.

- abstract public boolean mutilpleChildren() : retourne vrais si le chromosome peut contenir des enfants multiples et retourne faux sinon. Les types bag, list, set et vect ne sont pas des types multiples car il contiennent des éléments de type (et paramètres) identiques.

- abstract public boolean isRoot() : retourne vrais si cette classe de chromosomes est de type racine de l'arbre du génome et faux sinon.

- abstract protected void setCPPType() : Définit le type CPP et CPP long. Cette méthode est appelé quand l'un des deux est demandé et qu'il ne sont pas définit.

- abstract public boolean isSingleVectorTypeChild() : retourne vrais si le chromosome est un conteneur à enfants unique qui dans le code EO généré va hériter de eoAbstractVector.

- abstract public ChromoInit_Root getDefaultInitialisator() : retourne l'opération d'initialisation du chromosome par défaut.

- abstract public ChromoMut_Root getDefaultMutator() : retourne l'opération de mutation du chromosome par défaut.

- abstract public ChromoCross_Root getDefaultCrossover() : retourne l'opération de croisement du chromosome par défaut.

- abstract public String[] getParamType() : retourne un tableau des types de tous les paramètres du chromosome (dans l'ordre).

- abstract public String[] getParamName() : retourne un tableau des noms de tous les paramètres du chromosome (dans l'ordre).

- abstract public String[] getParamValue() : retourne un tableau des valeurs de tous les paramètres du chromosome (dans l'ordre).

- abstract public boolean isScalarType() : retourne vrais si le chromosome est un type scalaire. Ce type ne proviens pas d'une classe.

- abstract public boolean isTemplateType() : retourne vrais si le tChromoNodChildMultype du chromosome (généré en EO) est un type "template" ou paramétrable. Si vrais est retourné, il ne peut avoir d'enfants multiples.

- abstract public Vector getGenomeCodeLines() : retourne les lignes de code EO, pour générer la classe du chromosome.

2) La classe ChromoNodContainer

Les chromosomes de cette classe contiennent d'autres chromosomes. Cette classe hérite de ChromoNod. Elle à trois méthodes publics et un constructeur. C'est une classe abstraite.

- public ChromoNodContainer(ChromoNod prev, String name) throws ExpNameIncorrect : Constructeur. Transmet le parent prev et le nom name à la classe ChromoNod.

- public boolean isLeaf() : retourne faux car les objets de cette classe peuvent contenir des enfants.

- public boolean isScalarType() : retourne faux car il n'existe pas de type scalaire en C++.

- public Object clone() throws CloneNotSupportedException : duplique l'objet en suivant la hiérarchie d'héritage.

3) La classe ChromoNodChildMult

Cette classe hérite de ChromoNodContainer. Les chromosomes de cette classe peuvent avoir des enfants multiples.

a) Les attributs

- private Vector vectChildren = new Vector() : liste des chromosomes enfants.

- static private Vector vectUniqueType = new Vector() : liste de tous les types uniques pour cette classe de chromosome.

- static private Vector vectCPPType = new Vector() : liste de tous les types CPP pour cette classe de chromosome. Ces deux variables statics contienne le meme nombre d'éléments, tel que pour chaque index, un type unique (du vecteur vectUniqueType) correspond à un type CPP (du vecteur vectCPPType).

b) Les méthodes

- public ChromoNodChildMult(ChromoNod prev, String name) throws

ExpNameIncorrect : Constrcteur qui passe le parent prev et le nom name à ChromoNodContainer.

- public String getUniqueType() : retourne le type unique du chromosome.

- public String getGeneralType() : retourne le type générale du chromosome. Cette chaine de charcateres et identique au type CPP.

- public void setCPPType() : définit le type CPP et le type CPP long en tenant compte des enfants. Si deux chromosomes on deux enfants de nom différents, deux types CPP (et CPP long) vont etre généré. Si deux chromosomes ont les memes enfants (meme types et noms) mais placées dans un ordre différent, un seul type CPP (et CPP long) va étre généré.

- public boolean isChild(ChromoNod chromo) : vérifie que chromo est un enfant du chromosome.

- public Vector getChildren() : retourne la liste des enfants dans l'ordre inséré.

- public Vector getChildrenAlphaOrder() : retourne la liste des enfants dans l'ordre alphabetique.

- public void changeChildren(ChromoNod oldCh, ChromoNod newCh) : remplace un enfant par un autre. Il faut reinitialiser les types CPP (reinitCPPType()). Il faut retirer ce chromosome comme étant parent de l'ancien enfant et declarer ce chromosome comme parent du nouveau. La mise à jour de la vue est faite.

- public boolean mutilpleChildren() : retourne vrais car ce chromosome peut contenir plusieurs enfants.

- public void addChildNod(ChromoNod next) : ajoute un enfant. Il faut reinitialiser les types CPP (reinitCPPType()). Il faut déclarer ce chromosome comme parent du nouveau. La mise à jour de la vue est faite.

- public void removeChildNod(ChromoNod next) : retire l'enfant next. Il faut reinitialiser les types CPP (reinitCPPType()). Il faut retirer ce chromosome comme étant parent de l'ancien enfant. La mise à jour de la vue est faite.

- public boolean isSingleVectorTypeChild() : retourne faux car ces chromosomes ne sont pas de type eoAbstractVector et peuvent avoir des enfants multiples.

- public Object clone() throws CloneNotSupportedException : duplique l'objet en suivant

la hiérarchie d'héritage.

- public Vector getGenomeCodeLines() : retourne le code d'une classe généré par ces chromosomes.

4) La classe ChromoNodChildSimple

Cette classe hérite de ChromoNodContainer. Les chromosomes de cette classe peuvent avoir des enfants unique.

a) Les attributs

- private ChromoNod childNod : Le chromosome enfant. Il est unique.

b) Les méthodes

- public ChromoNodChildSimple(ChromoNod prev, String name) throws ExpNameIncorrect : Constrcteur qui passe le parent prev et le nom name à ChromoNodContainer.

- public void setCPPType() : définit le type CPP et CPP long du chromosome en tendant compte des enfants.

- public String getUniqueType() : retourne le type unique. (depends des type des enfants)

- public Vector getChildren() : retourne la liste (avec zéro ou un éléments) des enfants.

- public Vector getChildrenAlphaOrder() : retourne la liste (avec zéro ou un éléments) des enfants par ordre aphabetique. Comme au plus un enfant existe, getChildren() et getChildrenAlphaOrder() sont identiques.

- public void changeChildren(ChromoNod oldCh, ChromoNod newCh) : remplace l'enfant par un autre. Il faut reinitialiser les types CPP (reinitCPPType()). Il faut retirer ce chromosome comme étant parent de l'ancien enfant et declarer ce chromosome comme parent du nouveau. La mise à jour de la vue est faite.

- public boolean isChild(ChromoNod chromo) : retourne vrais si chromo est l'enfant du chromosome this.

- public void addChildNod(ChromoNod next) : ajoute un enfant. S'il existait un ancien, il faut le retirer et déclarer que son parent n'est plus this. Il faut reinitialiser les types CPP (reinitCPPType()). Il faut declarer ce chromosome comme parent du nouvel enfant. La mise à jour de la vue est faite.

- public void removeChildNod(ChromoNod next) : retire l'enfant next. Il faut reinitialiser les types CPP (reinitCPPType()). Il faut retirer ce chromosome comme étant parent de l'ancien enfant. La mise à jour de la vue est faite.

- public boolean mutilpleChildren() : retourne faux car il ne peut y avoir qu'un enfant.

- public Object clone() throws CloneNotSupportedException : duplique l'objet en suivant la hiérarchie d'héritage.

5) Les classes concrètes

Certains chromosomes ont des paramètres. Le nom des paramètres est donné comme variable static de manière public. Cette manière de faire permet de les rendre visible pour les routines de chargement. Lors de la sauvegarde ce paramètre est sauvé avec sa valeur sous ce nom. Grace à ce nom, la valeur du paramètre peut ainsi être chargé dans une variable et passé au constructeur du chromosome. A l'exception des paramètres seul la définition des méthodes abstraites héritées sont implantées. Seul les attributs propres au paramètres sont ajoutées.

a) les classe descendant de ChromoNodLeafTerminal

- classe ChromoNodLeafBool: C'est le chromosome booléen. Il prend la valeur vrais ou faux.

- classe ChromoNodLeafInt: C'est le chromosome entier. Il a deux paramètres, le minimum et le maximum. Il prend une valeur entière dans ces bornes (inclues).

- classe ChromoNodLeafPermut: C'est le chromosome permutation. C'est une liste ordonnée et fixe de N entiers. Chaque entier, de 0 à N-1, est present à un et un seul index. Il a un paramètre, sa taille. Ces contraintes peuvent etre modifié par un opérateur définit par l'utilisateur.

- classe ChromoNodLeafReal : C'est le chromosome réel. Il a deux paramètres, le minimum et le maximum. Il prend une valeur réels (double) dans ces bornes (inclues).

b) les classe descendant de ChromoNodChildSimple

Les quatre descendants de ChromoNodChildSimple sont des conteneurs d'éléments à un enfant. Ce chromosome enfants représente tous les éléments contenus. Ces chromosomes sont soit ordonnée, l'ordre compte ou non. Soit la taille est fixé soit elle est variable. Dans tous les cas il faut un chromosome enfant à ces chromosomes.

- classe ChromoNodBag : non ordonné taille fixe. Il faut un paramètre taille.

- classe ChromoNodList : ordonné taille variable.

- classe ChromoNodSet : non ordonné taille variable.

- classe ChromoNodVect : ordonné taille fixe. Il faut un paramètre taille.

c) les classe descendant de ChromoNodLeafChildMult

Ces deux classes peuvent avoir un nombre indéfini d'enfants. Chaque enfant est présent comme un élément, il est présent une et une seul fois. Il faut un minimum d'un enfant.

- classe ChromoNodRoot : C'est la racine du génome. Il est toujours présent au sommet de l'arbre. Il ne peut etre présent ailleur.

- classe ChromoNodTuple : C'est un chromosome comme les autres. Il peut etre présent partout sauf à la racine.

B) la classes de selection (la vue)

Ces classes permettent de sélectionner un chromosome. Ces classes sont divisé en deux parties. Une classe SelectorCreate_Types permet de sélectionner un sélecteur. Et ensuite la classe SelectorCreate_Root et ses descendants sont les sélecteurs des chromosomes. Ils permettent de créer un nouveau chromosome, suite à une action utilisateur. Deux actions sont possibles, soit l'utilisateur crée un nouveau chromosome, soit un chromosome est crée après une action de chargement.

B.1) Les sélecteurs de création du genome partie Type

La classe SelectorCreateTypes contient la liste de tous les sélecteur de chromosomes. Cette liste est en deux parties, une liste des sélecteurs de chromosomes pouvant être mis n'importe où dans l'arbre sauf à la racine, et le sélecteur du chromosome racine. Cette distinction est importante car le sélecteur racine ne figure pas comme choix de chromosome à ajouter. Par contre il figure dans la liste des chromosomes dont il faut traiter le chargement à partir d'un fichier. Cette classe permet de fournir la liste des chromosomes pouvant être sélectionné pour être ajouté au génome. Cette liste est sous forme d'un tableau donnant le nom du type du chromosome. Elle permet de transmettre au bon sélecteur l'ordre de création ou de chargement. Elle permet aussi de transmettre l'aide mémoire sur le chromosome, une petite description. Pour ajouter un chromosome

au programme, il faut l'ajouter à cette liste. Comme des opérateurs son nécessaires, cette action est insuffisante. Puisque chaque sorte d'opérateurs (initialisation, mutation et croisement) doit contenir une classe sélecteur Type qui définit la liste des opérateur spécifique à ce type. De plus la hiérarchie du modèle fait que chaque opérateur de chaque sorte hérite d'une classe spécifique au type et à la sorte. Donc il faut ajouter six classes dont trois abstraites (modèle). Ce sélecteur s'appelle "Type" puisqu'il gère tous les types.

1) Le détail de la classes

a) attributs privées

- private SelectorCreate_Root[] selectors
- private SelectorCreate_Root SelectorRoot

b) méthodes

- public SelectorCreate_Types() : constructeur

- public String[] getAllLongTypes(): retourne un tableau des types long de tous les chromosomes

- public String longTypeToType(String longType) : retourne à partir d'un type long le type de base du chromosome

- public String[] explanations(String type) : retourne à partir du type de base l'aide mémoire du chromosome. Pour cela, la méthode getExplanations du sélecteur du chromosome du type est appelé .

- public ChromoNod create(JDialog dialog,String type,String name,ChromoNod oldNod) : Fait suivre la demande de création d'un chromosome au sélecteur du type voulu. La variable dialog est le dialogue contenant l'éventuel dialogue de saisie des paramètres. Son nom est name et il remplace un chromosome oldNod (qui peut etre null).

- public ChromoNod load(String type,String name,Vector paraNames,Vector paraValues) : fait suivre au sélecteur voulu est désigné par "type", l'ordre de création d'un chromosome par chargement.

B.2) Les sélecteurs de création du genome partie sélecteur de chromosomes

Tous ces sélecteurs descendent de la classe abstraite SelectorCreate_Root. Un

sélecteur correspond à un chromosome du modèle. Et chaque chromosome du modèle a un sélecteur. L'idée sous-jacente est que les méthodes de SelectorCreate_Types transmettent les ordres de création à un sélecteur. C'est utile pour la gestion des paramètres. Il faut ouvrir un dialogue particulier pour les saisi. Ensuite il faut les transmettre au constructeur du chromosome. C'est une tache spécifique à chaque chromosome ayant des paramètres. De plus certaine informations, sur le nom du type de base, du type long et de l'aide mémoire, sont des données static du chromosome. Pour les accéder il faut connaître la classe du chromosome dans sa partie modèle. Les sélecteurs font ce lien.

1) la classe SelectorCreate_Root

a) méthodes de SelectorCreate_Root

- public SelectorCreate_Root() constructeur

- protected String getParam(String name,Vector paraNames,Vector paraValues) : méthode protégé qui permet de retourner la valeur d'un paramèter. Il faut entrer le nom du paramètre, la liste de tous les noms et de toutes les valeurs. Ces deux listes sont ordonnées, le premier nom correspond à la première valeur de l'autre liste. Cette méthode est utile pour le chargement.

b) méthodes abstraites de SelectorCreate_Root

- abstract public String[] getExplanations() : retourne le texte d'explication le type de chromosome. C'est l'aide mémoire.

- abstract public String getType() : retourne le type de base du chromosome.

- abstract public String getLongType() : retourne le type long du chromosome.

- abstract public ChromoNod create(JDialog owner,String name,ChromoNod oldNod) : retourne le chromosome nouvellement crée. La variable owner est le dialogue dans père du dialogue de saise d'éventuels paramètres. Le chromosome à un nom et peut remplacer un autre chromosome. Dans ce cas il est préférable que le dialogue de saisie de paramètres (si nécessaire) affiche par défaut les valeurs des paramètres de l'ancien chromosome. Il n'est pas possible dans l'état du programme de modifier les paramètres d'un chromosome. Il faut le remplacer par un nouveau ayant les paramètres désirées. Cette méthode lance les exceptions suivantes: ExpNameIncorrect, le nom est incorrect

(caractères interdits), ExpCancel, l'utilisateur à annulé l'action et ExpBadParameters, les paramètres sont incorrects (probabilité négative par exemple).

- abstract public ChromoNod load(String name,Vector paraNames,Vector paraValues) : Retourne un chromosome à partir d'un nom, d'une liste de paramètres et de leur valeur. Les deux listes sont ordonnées. La première valeur correspond au premièr paramètre. Cette méthode est définie pour les actions de chargements. Elle lance une exception IOException.

c) Les descendants de SelectorCreate_Root

Chaque chromosome à un sélecteur. Ce sélecteur implémente les méthodes abstraites de SelectorCreate_Root. Comme chacune de ces méthodes dépends de la classe du chromosome (modèle), elles ne peuvent être génériques.

C) L'affichage de l'arbre du génome

Le génome est affiché sous forme d'arbre. Chaque chromosome est un nœud avec son nom, son type et un bouton sur le quel est affiché sont type complet. Le bouton permet de modifier ce chromosome. Un click droit permet d'ajouter ou de retirer des chromosomes. Or cette partie est intimement liée à l'affichage les opérateurs. Dans ce cas le click droit n'existe plus et le bouton affiche et gère la liste des opérateurs du chromosome. Cette partie fait donc l'objet d'une explication à part.

III) classes opérateurs

Les chromosomes ont besoin d'opérateur pour que quelque chose agisse sur eux. Dans le code généré, les opérateurs d'initialisation donnent une valeur initiale, la plus part du temps aléatoire, au chromosome. L'initialisation (dans le sens algorithme évolutionnaire) est effectué après la construction du génome. Parfois le chromosome n'est pas entièrement construit et l'initialisation doit terminer le travail. C'est le cas des chromosomes de sorte liste (bag, list, set et vect). Quant un deces chromosomes est crée, il n'a aucun élément. Pour les listes de taille fixe, il faut un nombre d'éléments fixe non nulle. Pour les autres on peut souhaiter un nombre d'éléments initiaux non nulls. Les opérateurs de mutation modifient un individus sélectionné par le moteur d'évolution. Une donnée de l'individu sera modifié d'une manière en partie aléatoire. Le croisement va mettre le génome de deux parents en commun, pour former le genome de deux enfants.

Ces trois opérateurs commencent à la racine du génome et descendent de parent vers les enfants. Il est possible de bloquer ce passage et l'opérateur s'arrête sur le parent. Lors de l'initialisation il faut éviter ces blocages, certains chromosomes pouvant avoir des valeurs aléatoires et incohérents. Un exemple de blocage sur un parent est le suivant. Une mutation est faite sur une liste, un élément est supprimé. Or aucune mutation n'est faite sur un ou des éléments, la mutation se bloque sur la liste. Il faut remarquer que même si certains opérateurs sur un chromosome ne sont jamais exécuté pour cette raison, il faut qu'ils soient présent. Il faut au moins un opérateur de chaque sorte sur chaque chromosome. Sinon le code EO généré produira un bug, car il part du principe qu'un opérateur est toujours présent.

Dans le code implémentant l'opérateur, il est possible et conseillé d'utiliser des primitives. Ces primitives permettent de décrire une tache et sera étendue en plusieurs lignes de code. Il est à noter, comme il a été dit plus haut, que la propagation d'une opérations vers les chromosomes enfants n'est pas automatique. Pour propager cette action, il est possible d'utiliser la primitive "do_children". Cette primitive va propager l'action aux opérateurs de tous les enfants et éléments du chromosome. Si un chromosome de sorte liste (avec plusieurs éléments) est présents, l'action est propagé à tous les éléments. Pour exécuter les opérateurs sur un seul élément de ces chromosomes, il existe la primitive "apply_on_index(x)" qui applique l'action sur l'index x. Il est possible de coder plusieurs fois les primitives et de les utiliser dans une boucle. Donc il est possible d'utiliser "apply_on_index(x)" sur plusieurs éléments et a certaines conditions. Pour les même chromosomes, lors de l'initialisation, une primitive "setSize" ou "setSize(s)" permet de fixer leur taille. Si elle n'est pas utilisé, le chromosome aura aucun élément. Ces primitives sont utiles aussi bien dans le code prédéfini que définit par l'utilisateur.

Les opérateurs vont être généré en code C++ comme des foncteurs. Chaque foncteur reçoit un foncteur par enfant en paramètre du constructeur, en plus des paramètres du chromosome et de l'opérateur. Or ces foncteurs passés en paramètres ne sont pas obligé d'être des opérateur en tant que tel. Il est possible, et c'est comme ça que le fichier c++ est généré, de créer un foncteur de type combinaison et qui contient plusieurs opérateurs avec un poids. L'opérateur qui s'applique sur un enfant sera choisit aléatoirement, dans la liste des opérateurs, en fonctions du poids. Les paramètres du chromosome et de l'opérateur sont donné en paramètre au constructeur. Ainsi dans le

code du foncteur, elles sont visibles et utilisables.

Nous allons présenter deux sous parties de la parties opération. La première memorise tous les informations sur une opération: son poids, ses paramètres, le chromosome sur laquelle elle est attachée, son nom etc.. La seconde permet de créer (au sens objet) ces opérations de diverses manières, en répondant aux actions des utilisateurs, en chargement. Elle met à la disposition des dialogues, des listes d'opérations que l'utilisateur peut créer.

Il existe deux sortes d'opérateurs, les opérateurs dont le code généré est prédéfini et ceux où il est défini par l'utilisateur. Ce code est celui qui sera généré dans le foncteur. Ils utilisent les mêmes primitives et les même opérations. Il est possible par le bouton import de copier un opérateur pour en créer un autre, avec un nom différent. Si le type unique (depends des types des descendants) d'un chromosome a changé, un opérateur défini par l'utilisateur peut ne plus s'appliquer. Le bouton import permet de le copier, tant que le type est le même.

A) les classes d'opérateurs (modèle)

La classe racine de l'arbre d'héritage des classes d'opérateurs est la classe abstraite ChromoOperator. Cette classe contient le maximum de méthodes pour généraliser les opérateurs au maximum, même si un certain nombre d'entre elles sont abstraites. Cette classe contient une méthode pour générer le code en EO. Le code généré prendra la forme d'un foncteur. D'autres méthodes retournes le nom du foncteur, ainsi que la définition de la méthode opération (c++) du foncteur.

Plusieurs méthodes gèrent les paramètres des opérateurs. Ces paramètres doivent être distinguées des paramètres du chromosome. Même si les paramètres des chromosomes n'as de sens que pour les opérateurs. Dans le code généré c'est seulement au niveau des opérateurs que ces paramètres sont utilisées. Plusieurs opérateurs peuvent être combinées pour offrir une liste d'opérateur dont un sera exécuté avec une probabilité dépendant de son poids. La classe ChromoOperator prends en charge la gestion des poids de l'opérateur. Chaque opérateur à un nom. Il peut aussi avoir un nom long qui prends en compte les paramètres. Un opérateur est associé à un type de chromosome et en aucun cas il ne peut être affecté à un autre chromosome différent de son type. Toutefois la méthode appelé "getUniqueType()" est particulière. Elle retourne le type unique de l'opérateur s'il est définit par l'utilisateur mais retourne le type (simple) s'il est prédéfinit.

Le type unique permet de définir un chromosome par rapport à son type et à celui de ses descendants. Si l'utilisateur définit une opération dont le corps dépends des enfants, l'opération peut devenir erroné s'il change les enfants (ajout - retrait - changement de type - parfois leur nom). Pour un tuple le nom des enfants est pris en compte dans le type unique. Ainsi si le nom des enfants change le type unique aussi. C'est le cas uniquement pour le tuple car pour les autres types, utilisateur ne devrais pas accéder aux enfants par leur nom. Un opérateur défini par l'utilisateur ne peut être affecté qu'à un seul type unique. Une méthode permet d'obtenir l'implémentation de l'opérateur. Il faut appeller un préprocesseur pour traiter l'implémentation brute. Cet implementation va devenir le corps de la méthode opérateur du foncteur crée. Si des paramètres (issus du chromosome ou directement de l'opérateur) ils peuvent être traité dans ce corps. Évidemment cette implémentation est définit par le développeur pour les opérateurs prédéfinit et par l'utilisateur pour les autres.

Les opérateurs se distinguent par leur sorte (initialisation, mutation, croisement) qui est inscrit dans leur nom de classe, ainsi ChromoInit_Root est la racine de tous les opérateur d'initialisation, ChromoMut_Root de tous les opérateur de mutation et ChromoCross_Root de tous les opérateurs de croisement. Plus bas dans l'arbre d'héritage une distinction est faite entre opérateurs prédéfinit (ChromoXXX_PreDefined) et définit par l'utilisateur (ChromoXXX_UserDefined). La classe ChromoXXX_UserDefined est une classe concrète alors que ChromoXXX_PreDefined est abstraite. En dessous de ChromoXXX_PreDefined viennent les classes abstraites spécifiques à un type. Par exemple ChromoInit_TypeBool est la classe abstraite des initialisateurs de type booléens. Ces classes spécifient leur type. Et en fin en dessous de ces classes abstraites viennent les classes qui peuvent être réellement implantées comme opérateurs prédéfinit. Par exemple ChromoInitBoolFalse, cette opérateur initialise le chromosome de type booléen à faux.

1) la classe ChromoOperator (abstraite)

a) les attributs

- protected ChromoNod chromoNod : chromosome sur le quel s'applique l'opérateur.

- protected String varName : lors de l'appel du foncteur le chromosome est passé en paramètre. Ce paramètre à un nom et peut etre utilisé dans le code. Ce nom est inscrit dans varName.

- private double weight : poids de l'opérateur. Si le chromosome a plusieurs opérateurs, l'opérateur choisit dépends de son poids. Il est tiré par roulette.

- private int tempWeight : poids provisoire du chromosome.

b) les méthodes

- static public void reinitialise() : retire tous les opérateurs défini par l'utilisateur de la mémoire (lors du chargement)

- public ChromoOperator() constructeur

- public boolean isTemplateType() : retourne vrais si le type du chromosome va générer une classe C++ de type template.

- public double getWeight() : retourne le poids

- public void setWeight(double weight) : change le poids

- public int getTempWeight() : retourne le poids temporaire.

- public void setTempWeight(int weight) : modifie le poids temporaire.

- public void setChromoNod(ChromoNod chromoNod) : attache un chromosome à l'opérateur. Initialise varName.

- public String getVarName() : retourne varName.

- private Vector evaluateLines(String stg) : préprosesseur. La variable stg prends le code de l'opérateur (C++) du foncteur avec les primitives. Cette méthode retourne dans un vecteur de chaine de caractères le code étandu sans primitives.

- public Vector getCodeLines(String appliName) : retourne une liste de chaines de caractères définissant le code complet de la classe du foncteur. appliName est le nom donné aux données (cf fichier de sauvegarde).

- public String getFuncDeclaration(String type, String name,String varName) : Lors de la saisie du code par l'utilisateur, elle semble etre dans une fonction. Cette méthode retourne la signature de cette fonction.

- public String getFunctorName() : retourne le nom de la classe du foncteur.

- public boolean isRoot() : retourne vrai si le chromosome attaché est un chromosome racine.

- protected Object clone() : copie de la l'objet.

-private String modifyVarNames(ChromoNod chromo, String code) : point d'entrée de la partie qui modifie les noms des descendants en accesseurs. L'opérateur peut utiliser les noms de ses descendants. Cette méthode les transforme leur appel en méthodes accesseur. chromo est le chromosome mère et code est le code de l'opérateur.

-private String modifyVarNamesRec(ChromoNod chromo, String code) : partie recursive sur les chromosomes de la méthode modifyVarNames. chromo est le chromosome courant et code le code de l'opérateur.

c) les méthodes abstraites

- abstract public String getOperatorType() : sorte abrégé de l'opérateur. Soit init, mut ou cross pour initialisation, mutation ou croisement.

- abstract public String getOperatorSuperClass() : super classe de la classe foncteur. La chaine de caractère ne prends pas en compte le paramètre du template de cette classe.

- abstract public String getOperatorCombinaison() : type de combinaison des enfants. Toujours la meme que getOperatorSuperClass(). La chaine de caractère ne prends pas en compte le paramètre du template de cette classe.

- abstract public String getFullName() : Nom de l'opérateur avec tous ses paramètres.

- abstract public String getName() : Nom de l'opérateur.

- abstract public String getType() : type du chromosome de l'opérateur.

- abstract public String getImplementation() : code qui sera généré dans le foncteur en EO.

- abstract public boolean isUserDefined() : l'opérateur est définit par l'utilisateur (vrais - faux)

- abstract public String getUniqueType() : type unique du chromosome de l'opérateur. Un opérateur défini par l'utilisateur ne s'applique qu'a un type unique.

- abstract public String[] getParamType() :liste ordonné des type générés en C++ des paramètres de l'opérateur. Fonctionne avec getParamName() et getParamValue().

- abstract public String[] getParamName() : liste ordonné des noms des paramètres de l'opérateur. Fonctionne avec getParamValue() et getParamType().

- abstract public String[] getParamValue(): liste ordonné des valeurs des paramètres de l'opérateur. Fonctionne avec getParamName() et getParamType().

- abstract public String[] getExplanations(): Explications sur l'opérateur. Aide mémoire.

- abstract public String getOperReturnType() : type de retourn en C++ de l'opérateur (C++) du foncteur.

- abstract public boolean isOneParamOp() : retourne vrais si un seul paramètre est demandé pour l'opérateur (C++) du foncteur. L'initialisation et la mutation demandent un paramètre, le chromosome sur le quel ils agissent. Le croisement demande deux paramètres, les deux chromosomes issus des individus enfants.

- abstract protected String getBeginFunctorOperator() : première ligne du code de l'opérateur (C++) du foncteur.

- abstract protected String getIterateFunctorOperatorDebLigne() : Lors de l'appel d'un opérateur sur un enfant ou un élément du chromosome, cette méthode est le debut de la ligne. Elle est utile pour mémoriser si une modification est faite. Pour la transmettre en cas de valeur vrais comme valeur de retourne. Elle fonctionne avec getIterateFunctorOperatorFinLigne()

- abstract protected String getIterateFunctorOperatorFinLigne() : Lors de l'appel d'un opérateur sur un enfant ou un élément du chromosome, cette méthode est le fin de la ligne. Elle est utile pour mémoriser si une modification est faite. Pour la transmettre en cas de valeur vrais comme valeur de retourne. Elle fonctionne avec getIterateFunctorOperatorDebLigne().

- abstract protected String getEndFunctorOperator() : dernière ligne du code de l'opérateur (C++) du foncteur. Cette ligne est la ligne de retours.

2) les classes ChromoInit_Root, ChromoMut_Root et ChromoCross_Root (abstraite)

Ces classes sont spécifiques à une sorte d'opérateur (initialisation, mutation, croisement).

a) les attributs

- static public String operatorType : donne l'abréviation de la sorte d'opérateur.

- static private String operReturnType : donne le type de retour de l'opérateur (c++) du

foncteur.

b) les méthodes

- static public ChromoInit_Root clone(ChromoInit_Root chromo) : copie l'opérateur

- static public boolean doesNameExist(String name, String type) : retourne vrais si le nom d'opérateur name existe déjà pour ce type de chromosome.

- public ChromoInit_Root() : constructeur

- protected String getBeginFunctorOperator() : première ligne du code de l'opérateur (C++) du foncteur.

- protected String getIterateFunctorOperatorDebLigne() : Lors de l'appel d'un opérateur sur un enfant ou un élément du chromosome, cette méthode est le debut de la ligne. Elle est utile pour mémoriser si une modification est faite. Pour la transmettre en cas de valeur vrais comme valeur de retourne. Elle fonctionne avec getIterateFunctorOperatorFinLigne()

- protected String getIterateFunctorOperatorFinLigne(): Lors de l'appel d'un opérateur sur un enfant ou un élément du chromosome, cette méthode est le fin de la ligne. Elle est utile pour mémoriser si une modification est faite. Pour la transmettre en cas de valeur vrais comme valeur de retourne. Elle fonctionne avec getIterateFunctorOperatorDebLigne().

- protected String getEndFunctorOperator() : dernière ligne du code de l'opérateur (C++) du foncteur. Cette ligne est la ligne de retours.

- public String getOperatorType(): sorte abrégé de l'opérateur. Soit init, mut ou cross pour initialisation, mutation ou croisement.

- public String getOperatorSuperClass(): super classe de la classe foncteur. La chaine de caractère ne prends pas en compte le paramètre du template de cette classe.

- public String getOperatorCombinaison(): type de combinaison des enfants. Toujours la meme que getOperatorSuperClass(). La chaine de caractère ne prends pas en compte le paramètre du template de cette classe.

- public String getOperReturnType(): type de retourn en C++ de l'opérateur (C++) du foncteur.

- public boolean isOneParamOp(): retourne vrais si un seul paramètre est demandé pour

l'opérateur (C++) du foncteur. L'initialisation et la mutation demandent un paramètre, le chromosome sur le quel ils agissent. Le croisement demande deux paramètres, les deux chromosomes issus des individus enfants.

- protected Object clone() : copie de l'opérateur

c) les variantes

Les deux attributs suivantes sont spécifiques à ChromoMut_Root et ChromoCross_Root. Ils permettent la mise en relief du code saisi par l'utilisateur. Ils permettent de montrer la première et la dernière ligne du code du foncteur.

- static public String beginFunctorOperator="bool modified=false;";

- static public String endFunctorOperator= "return modified;";

3) les classes ChromoInit_UserDefined, ChromoMut_UserDefined et ChromoCross_UserDefined

Ces classes contiennent toutes les information sur les objets opérateurs défini par l'utilisateur. Ils sont stocké dans plusieurs vecteurs privées. Pour accéder à leurs informations if faut consulter ces vecteurs en demandant l'index de l'opérateur. Ce sont des classes concrètes.

a) les attributs

- static public String name = "user defined": nom donné à un opérateur pas encore crée.

- static public String[] getLoadParamName(){return new String[0];} : liste des noms des paramètres. Ils sont vides car ils ne peuvent avoir des paramètres

- static public String[] getLoadParamType(){return new String[0];}: liste des types des paramètres. Ils sont vides car ils ne peuvent avoir des paramètres

- static private Vector names = new Vector() : noms des opérateurs (par index).

- static private Vector implementation = new Vector() : implémentation des opérateurs (par index).

- static private Vector typeVect = new Vector() : types des opérateurs (par index).

- static private Vector uniqueTypeVect = new Vector() : type unique des opérateurs (par

index).

- static private Vector varNamesVect = new Vector() : varName des opérateurs (par index). C'est le nom donné à la variable en C++ lors de la définition du code du foncteur.

- private int index : index de l'objet opérateur dans les données : names, implementation, typeVect, uniqueTypeVect et varNamesVect

b) les méthodes

- static public String[] explanations() : aide mémoire. Il déclare que l'opérateur à été définit par l'utilisateur.

- static public void reinitialise() : vide tous les opérateurs de la mémoire. Utilisé lors du chargement.

- static public Vector getUserDefNames(String type) : retourne la list de tous les noms des opérateurs de ce type.

- static public Vector getUserDefTypes(String type) : retourne la liste de tous les types uniques des opérateurs (dans le meme ordre que getUserDefNames)

- static public Vector getUserDefNames(String type,String uniqueType): retourne la list de tous les noms des opérateurs de ce type et de ce type unique

- public ChromoCross_UserDefined(String newName, String newImplement, String type, String uniquetype, String varName) : constructeur, crée un nouveau objet opérateur.

- public ChromoCross_UserDefined(String name,String type) : constructeur. Crée un objet opérateur qui existe déjà. Ce sont deux objets differents mais leur index dans les données contenus dans les vecteurs ci-dessus est identique.

- public void change(String newName,String newImpl) : modifie le nom et l'implémentation d'un opérateur.

- public String getUniqueType() : retourne le type unique.

- public String getFullName() : retourne le nom de l'opérateur

- public String getName() : retourne le nom de l'opérateur.

- public String getType() : retourne le type de l'opérateur

- public String getImplementation() : retourne l'implémentation de l'opérateur.

- public String[] getParamType() : retourne un tableau vide car aucun paramètre est permis.

- public String[] getParamName() : retourne un tableau vide car aucun paramètre est permis.

- public String[] getParamValue() : retourne un tableau vide car aucun paramètre est permis.

- public String[] getExplanations() : retourne l'aide mémoire qui dit que l'opérateur a été défini par l'utilisateur.

- public boolean isUserDefined() : retourne vrais car l'opérateur est défini par l'utilisateur.

- protected Object clone() : copie de l'opérateur.

4) les classes ChromoInit_PreDefined, ChromoMut_PreDefined et ChromoCross_PreDefined

Ce sont des classes abstraites.

- public ChromoInit_PreDefined() : constructeur.

- public boolean isUserDefined() : retourne faux car l'opérateur est prédefini.

- public String getUniqueType() : retourne le type unique qui est identique au type.

- protected Object clone() copie de l'opérateur.

5) les classes ChromoYYY_TypeXXX

Ces classes généralisent le type des opérateurs. YYY désigne la sorte (init, mut ou cross) et XXX le type de l'opérateur.

- static public String type : C'est le type de l'opérateur.

- public ChromoCross_TypeSet() : constructeur

- public String getType() : C'est le type de l'opérateur.

- protected Object clone() : copie de l'opérateur.

6) les classes concètes des opérateurs ChromoYYY_XXXZZZ

YYY désigne la sorte (init, mut ou cross), XXX le type de l'opérateur et ZZZ est le nom.

a) les attributs

- static public String name : c'est ne nom de l'opérateur. Il est constant, il peut etre compié mais ne doit etre redéfini null part ailleurs. De cette manière il peut etre modifié facilement. Dans ce cas il faut remarquer que les fichiers sauvegardées contiennent l'ancien nom et ne reconaitra plus l'opérateur. Pour un type donné, il doit etre unique.

b) les méthodes

- static public String[] getLoadParamName(): liste des nom des paramètres pour le chargement.

- static public String[] getLoadParamType(): liste des types des paramètres pour le chargement. Fonctionne avec getLoadParamName(), le premier élément correspond au premier de getLoadParamName(), le deuxième aussi et ainsi de suite.

- static public String[] explanations(): aide mémoire pour cet opérateur.

- public ChromoCrossVect1Point() : constructeur.

- public String getFullName() : retourne le nom complet avec la valeur des paramètres.

- public String getName() : retourne le nom sans la valeur des paramètres.

- public String getImplementation() : retourne l'implémentation qui servira à générer le code EO

- public String[] getParamType() : tableau des types des paramètres par index.

- public String[] getParamName() : tableau des noms des paramètres par index.

- public String[] getParamValue() : tableau des valeurs des paramètres par index.

- public String[] getExplanations() : aide mémoire

-protected Object clone() : copie de l'objet

B) les classes d'opérateurs (vue et sélection)

Ces classes servent à sélectionner et à créer un opérateur. Il peut être crée par l'utilisateur ou charge d'un fichier. Un nouvel opérateur peut être crée de trois manières. Soit il est ajoute simplement (avec éventuellement saisie de paramètres). Soit il modifie un autre et donc le nouveau remplace l'ancien. Si c'est le même opérateur et s'il prend des paramètres, les paramètres saisi sont par défaut ceux de l'ancien opérateur. Soit il crée un opérateur par défaut avec les valeurs des paramètres par défaut. Ce cas est utile pour

importer un opérateur. Son implémentation est affiché dans une fenêtre et l'utilisateur peut le modifier et créer un nouveau type d'opérateur. Dans ce cas les paramètres n'ont pas d'importance car l'opérateur n'est crée que pour être copié.

Il existe deux sortes de sélecteurs, les sélecteurs de type et les sélecteurs d'opérateurs. Les sélecteurs de types parcourent une liste de sélecteurs d'opérateurs et quant il trouvent le bon lui demande de créer l'objet opérateur du modèle.

B.1) Les sélecteurs de types

Il existe trois niveau de sélecteurs de types, les classes SelectorXXX_Root, SelectorXXX_Type et SelectorXXX_TypeYYY. XXX désigne la sorte d'opérateur et YYY leur type. SelectorXXX_Type est une classe abstraite et SelectorXXX_TypeYYY est leur descendants. Lors d'une recherche de sélecteur d'opérateur les classes SelectorXXX_Root va parcourir la liste des types. Une fois le bon type trouvé, il demandent à une classe SelectorXXX_Type de continuer. Cette objet va parcourir la liste des opérateurs pour trouver le bon sélecteur d'opérateur et lui appliquer la requête. Il faut noter d'autre part que les sélecteurs d'opérateurs héritent de SelectorXXX_Root qui est abstraite. Ce sont ces méthodes statics qui font partie des sélecteurs de types.

1) Les classes SelectorInit_Root, SelectorMut_Root et SelectorCross_Root

a) les attributs

- static private SelectorXXX_Type[] selectors : liste des sélecteur de types descendant de SelectorXXX_Type.

b) les méthodes

- public SelectorXXX_Root() : constructeur

- static public SelectorXXX_Root[] getInitForType(String type,String uniqueType) : retourne la liste des sélecteurs d'opérateurs du type et du type unique.

- static public SelectorXXX_Root[] getAllInitForType(String type) : retourne la liste des sélecteurs d'opérateurs du type.

- static public ChromoXXX_Root load(boolean userdef,String type,String name,Vector paraNames,Vector paraValues) : retourne le chromosome chargé d'un ficher. LA variable userdef est vrai si l'opérateur a été defini par l'utilisateur, type est son type, name

son nom. Les variables paraNames et paraValues fonctionnent ensemble. Une (paraNames) désigne une liste de nom de paramètres et l'autre (paraValues) désigne ses valeur. Ces deux listes sont passées par la routine de chargement pour créer le chromosome.

- static private SelectorXXX_Type getSelector_Type(String type) : retourne un sélecteur de type descendant de SelectorXXX_Type.

c) les méthodes abstraites

- abstract public ChromoXXX_Root create(JDialog owner,ChromoNod chromo) : crée un nouveau objet opérateur. Le dialogue paramètres est dans owner et le'opérateur est sur le chromosome chromo.

- abstract public ChromoXXX_Root create(JDialog owner,ChromoNod chromo, ChromoXXX_Root init) : crée un nouveau objet opérateur à partir d'un ancien. Le dialogue paramètres est dans owner et le'opérateur est sur le chromosome chromo. L'ancien opérateur est init.

- abstract public ChromoXXX_Root duplicate(JDialog owner,ChromoNod chromo) : crée provisoirement un objet opérateur pour figurer dans la routine import. Il n'est pas besoin de saisir les paramètres quant ils sont présent. Il est crée pour pouvoir lire l'implémentation. Le dialogue paramètres est dans owner et l'opérateur est sur le chromosome chromo.

2) Les classes SelectorInit_Type, SelectorMut_Type et SelectorCross_Type

a) les méthodes abstraites

- abstract protected String getType() : retourne le type de l'opérateur

- abstract protected SelectorXXX_Root[] getPredefSel() : retourne la liste des sélecteurs d'opérateurs prédéfinis

b) les méthodes

- public SelectorXXX_Root[] getXXXForType(String uniquetype): retourne la liste de tous les sélecteurs d'opérateurs pour ce type unique.

- public SelectorXXX_Root[] getAllXXXForType() : retourne la liste de tous les

sélecteurs d'opérateurs pour ce type.

- public ChromoXXX_Root load(String name,Vector paraNames,Vector paraValues) : retourne le chromosome chargé d'un ficher. La variable name est son nom. Les variables paraNames et paraValues fonctionnent ensemble. Une (paraNames) désigne une liste de nom de paramètres et l'autre (paraValues) désigne ses valeur. Ces deux listes sont passées par la routine de chargement pour créer le chromosome.

B.2) Les sélecteur d'opérateurs

Pour chaque opérateur il existe un sélecteur opérateur nommé SelectorXXXYYYZZZ. XXX est la sorte de l'opérateur. YYY le type du chromosome sur lequel il se déclanche, et ZZZ est son nom. Comme cela a été dit plus haut, les sélecteur d'opérateur répondent à une action utilisateur et crée un opérateur dans le modèle.

a) les attributs

- static private String name : c'est le nom de l'opérateur.

b) les méthodes

- public SelectorXXXYYYZZZ : constructeur

- public ChromoXXX_Root create(JDialog owner,ChromoNod chromo) : construit un nouvel opérateur sur le chromosome chromo. La variable owner est le dialogue dans lequel se place un éventuel dialogue de saisie de paramètres.

- public ChromoXXX_Root create(JDialog owner,ChromoNod chromo, ChromoXXX_Root xxx) construit un nouvel opérateur sur le chromosome chromo. Ce nouvel opérateur remplace l'ancien xxx. Si des paramètres sont saisis, il sont égaux à ceux de xxx par défaut. La variable owner est le dialogue dans lequel se place un éventuel dialogue de saisie de paramètres. Le poids de l'ancien opérateur est copié dans le nouveau.

- public ChromoMut_Root create() : Crée un nouvel opérateur avec les paramètres par défaut. Aucun paramètre ne peut etre saisi. Cette méthode est utile pour créer un opérateur qui ne sera pas affécté à un chromosome mais servira à etre copié dans une action "import" pour créer une nouvelle sorte d'opérateur.

- String getName() : retourne le nom de l'opérateur.

- public String[] explanations() : retourne l'aide mémoire de l'opérateur.

- public ChromoMut_Root load(Vector paraNames,Vector paraValues) : retourne le chromosome chargé d'un ficher. Les variables paraNames et paraValues fonctionnent ensemble. Une (paraNames) désigne une liste de nom de paramètres et l'autre (paraValues) désigne ses valeur. Ces deux listes sont passées par la routine de chargement pour créer le chromosome.

IV) La vue- contrôle

A) L'affichage des chromosomes

Les chromosomes s'affichent à l'écran en mettant en avant leur structure d'arbre. Les parents se placent au dessus des enfants sous une forme pyramidale. Nous avons quatre mode d'affichage. Un mode création qui permet de créer et de modifier le génome. Un mode initialisation qui affiche les opérateur initialisation sur chaque chromosome. Un mode mutation qui affiche les opérateur initialisation sur chaque chromosome. Et un mode croisement qui affiche les opérateur de croisement sur chaque chromosome. Nous avons cinq onglet, les quatre premier sont dédié à un mode d'affichage de chromosomes. Le dernier est l'onglet de la fonction d'évaluation. Chaque chromosome dans cette vue est un objet graphique GraphChromo. Il contient son nom, son type et un bouton. Le contenu et l'action du bouton dépend du mode. Dans le mode création un click droit sur un chromosome permet d'ajouter ou de supprimer des chromosomes. L'affichage de l'ensemble des chromosomes est faite par les classes GraphPainter et GraphMainPainter.

A.1) L'affichage des chromosomes (à l'unité)

1) La classe abstraite GraphChromo

Cette classe généralise le vue et le contrôle sur un chromosome. Un chromosome peut être sélectionné de différentes manières. Une sélection simple fait suite à une action souri sur un chromosome, son bord devient bleu. Une sélection de sous arbre affriche en vert clair le sous arbre qui viens d'etre copié. Cette classe généralise les actions et l'affichage d'un chromosome des quatre modes.

a) les attributs

- private GraphChromo parent : parent du chromosome

- protected ChromoNod chromoNod : chromosome dans sa version modèle

- private JLabel nameLabel : label du nom du chromosome

- private JLabel spaceLabel : espace entre le nom et le type

- private JLabel typeLabel : label du type du chromosome

- protected JButton button : bouton du chromosome

- protected Vector children : chromosomes enfants

- private int borderThickness : épaisseur de la bordure

- protected GraphPainter painter : panel défié à l'affichage des chromosomes dans lequel il se trouve

- private boolean selected : indique si le chromosome est sélectionné par l'utilisateur (vrais = sélectionné)

- private boolean treeSelected : indique si le chromosome fais partie du sous arbre sélectionné (vrais = sélectionné)

- private boolean warning : indique si le chromosome contient une erreur. Un nom qui n'est pas unique ou un chromosome conteneur qui n'a pas d'enfants. (vrais = erreur)

- private JPanel panelOnButton : Le texte du bouton est sous forme de panel. Ce panel est dans le bouton.

- static ExamineOperVue operVue=null : objet de confirmation de retrait d'un opérateur sur ordre du modèle (partie vue)

b) les méthodes

- protected GraphChromo(ChromoNod nod,GraphPainter painter) : constructeur. Le chromosome nod est dans le modèle. Le chromosome this et nod sont mis en correspondance. LA zone d'affichage du chromosome est painter.

- public void addTextToButton() : ajoute le texte issu de getButtonString() dans le bouton. Si le chgromsome est le chromosome racine seul "ROOT" est affiché et le bouton est invalidé. Les action de mise à jour des tailles des composants sont faites, sur le panel du bouton, sur le bouton et le chromosome

- protected void setChildren() : crée récursivement les enfants du chromosomes. Dans un premièr temps ils sont retiré et ensuite recrée. Cette méthode vérifie que le chromosome

est correct sinon warring est mis a vrais.

- private void removeChildrenNod() : retire et efface tous les enfants.

- public Vector getChildrenChromo() : copie l'attribut children

- public GraphChromo getParentChromo() : retourne le parent.

- public int getMaxWidth(int space) : largeur maximale des enfants d'un chromosome et de ses descendants. Si plusieurs chromosomes sont descendant d'un meme parent, la largeur est la somme de leur largeur avec un espacement de "space".

- public int getMaxWidthChildren(int space): largeur maximale des descendants d'un chromosome (sans tenir compte du chromosome lui meme). Si plusieurs chromosomes sont descendant d'un meme parent, la largeur est la somme de leur largeur avec un espacement de "space".

- public boolean isSelected() : retourne vrai si le chromosome est sélectionné.

- public void select(): sélectionne le chromosome. Indique au conteneur que le nouveau chromosome est sélectionné et colorie le bord en bleu.

- public void selectTree() : sélectionne comme sous arbre le chromosome. Indique au conteneur que le nouveau sous arbre est sélectionné et colorie le bord en vert clair.

- private void selectTreeRec(): procède à la sélection du sous arbre engendré par selectTree() par récurrence.

- public void deSelect() : désélectionne le chromosome d'une sélection ordinaire. Redessine le bord.

- public void deSelectTree() : désélectionne le chromosome d'une sélection de sous arbre. Redessine le bord. Cet action est propagé par récurrence aux enfants.

- public void setWarning(boolean war) : met à jour la variable warning et affiche la bordure en conséquence. Si un warning est vrai, le bord devient rouge.

- public boolean getWarning() : retourne l'état de l'attribut warning

- public boolean isOkay() : retourne faux si le chromosome ou un de ses descendants a un warning a vrais.

- private void setBorderColor() : redessine le bord du chromosome dans la couleur adéquat.

- public ChromoNod getChromoNod() : retourne le chromosome du modèle.

-public GraphChromo getGraphChromoFromNod(ChromoNod chromo): La variable chromo est le chromosome du modèle dont on cherche par recurrence le chromosome graphique.

- public Dimension getPreferredSize() : retourne la dimension préféré du chromosome. Cette méthode est utilisé pour fixer la taille du chromosome.

- public void update(Observable o, Object arg) : un événement à eu lieu sur le chromosome du modèle. Une mise à jour est nécessaire.

c) les méthodes abstraites

- abstract protected String[] getButtonString() : C'est le texte qui doit etre affiché sur le bouton
- abstract protected GraphChromo newGraphChromo(ChromoNod nod,GraphPainter painter) : appel un constructeur de GraphChromo. Comme GraphChromo est abstraite, il faut une méthode qui appel un constructeur concret.

- abstract protected void verifyNod(): vérifie que si des erreurs existent sur le chromosome. Si deux chromosomes ont le meme nom, si un conteneur n'a pas d'enfants, si un chromosome, si un chromosome n'a pas d'opérateur de la sorte. Si une erreur ou non est trouvé l'attribut warning est affecté.

2) La classe GraphChromoCreate

Cette classe affiche un chromosome en mode création. C'est à dire que le chromosome peut être modifié. Un chromosome peut être ajouté ou supprimé.

a) les attributs

- private JPopupMenu popup; menu popup issu d'un click droit

- private JMenuItem insertOver = new JMenuItem("insert over") : menu du popup.

- private JMenuItem insertUnder = new JMenuItem("insert under") : menu du popup.

- private JMenuItem deleteSubTree = new JMenuItem("delete subtree") : menu du popup.

- private JMenuItem copySubTree = new JMenuItem("copy subtree") : menu du popup.

- private JMenuItem pasteSubTree = new JMenuItem("paste subtree") : menu du popup.

b) les méthodes

- public GraphChromoCreate(ChromoNod nod,GraphPainter painter) : constructeur. Le chromosome nod est dans le modèle. Le chromosome this et nod sont mis en correspondance. LA zone d'affichage du chromosome est painter. Initialise les reflex sur les menus du popup et le bouton.

- private void drawPopup(MouseEvent e) : initilise le menu popup par rapport au chromosome. Certains menus peuvent etre interdit comme le fait de créer un enfant sous un type booléen qui ne peut en contenir.

- protected String[] getButtonString() : La chaine de caractères affiché sur le bouton est le type complet du chromosome.

- protected GraphChromo newGraphChromo(ChromoNod nod,GraphPainter painter) : retourne un nouveau un objet GraphChromoCreate

- protected void verifyNod() : lance à la verifyNodRec() a la racine du génome avec deux vecteurs vides.

- private void verifyNodRec(Vector names, Vector chromos): vérifie par récurrence que chaque chromosome a un nom unique et qu'un conteneur a au moins un enfant. Si ce n'est pas le cas setWarning(true) est appellé.

3) Les classes GraphChromoInit, GraphChromoMut et GraphChromoCross

Ces trois classes affiche un chromosome dans l'un des trois modes opérateurs. La liste des opérateur du mode peut être modifié sur un chromosome.

a) les méthodes

- public GraphChromoXXX(ChromoNod nod,GraphPainter painter): constructeur. Le chromosome nod est dans le modèle. Le chromosome this et nod sont mis en correspondance. La zone d'affichage du chromosome est painter. Initialise les reflex sur le bouton.
- protected String[] getButtonString() : retourne le texte du bouton. Il s'agit de la liste des opérateurs de la sorte sur le chromosome. Leur nom est complet et comprend les paramètres.
- protected GraphChromo newGraphChromo(ChromoNod nod,GraphPainter painter): retourne un nouveau un objet GraphChromoXXX
- protected void verifyNod() : vérifie qu'au moins un opérateur est présent sur le

chromosome. Si ce n'est pas le cas setWarning(true) est appellé.

4) Les classes GraphEvalFunc

Cette classe est responsable de la saisie et de l'affichage de la fonction d'évaluation.

a) les attributs

- private JTextArea textArea : zone de saisi de texte.

- private String evalFunc="" : partie de la fonction d'évaluation à retenir et à insérer dans la méthode EO adéquat.

- private boolean editable=true : permet d'activer et de désactiver le mode modification. Si cet attribut n'existait pas, une modification engendrait une modification qui appelait de nouveau la méthode. Une boucle infinie s'en suivait. La méthode qui formate le texte, inhibe l'appel de la méthode de formatage (qui s'exécute suite à une modification) lors de modification, avec cet attribut.

- private ChromoNod genome : racine du génome.

- private int len=0: nombre de lignes de commentaires désignant tous les chromosomes du génome.

- private String[] helpVarNames = new String[0] : lignes de commentaires désignant tous les chromosomes du génome.

b) les méthodes

- public GraphEvalFunc(ChromoNod chromo) : constructeur, chromo est la racine du génome.

- public String getEvalFunc() : met à jour et retourne l'attribut evalFunc.

- public void setEvalFunc(String text,ChromoNod chromo, int l): initialise la fonction d'évaluation avec le texte "text", sur génome de racine chromo et ayant l lignes de commentaires (inclus dans les premières lignes du text).

- private String getFullText(String text) : met à jour le texte de la fonction d'évaluation dans la zone de texte. text est le texte de evalFunc.

- private void setEvalText() : reformat la zone de texte après une modification de celle-ci.

- private String[] getHelpVarNames() : retourne les commentaires sur les noms, types et

chemins des chromosomes. Point d'entrée.

- private void setHelpVarNamesRec(ChromoNod chromo,String start, Vector vect) : retourne les commentaires sur les noms, types et chemins des chromosomes. Partie récursive. chromo est le chromosome courrant. start le chemin jusqu'au chromosome courant inclus et vect l'ensemble des lignes de commentaires.

- public int getNbHelpVarNames() : retourne le nombre de lignes de commentaires.

5) La classes GraphPainter

Cette classe affiche tous les chromosomes du génome. Elle hérite de JPanel. Elle est appliqué dans les quatre modes.

a) les attributs

- final public static int CREATE=0 : Chromosomes de sorte création

- final public static int INITIALIZE=1 : Chromosomes de sorte initialisation

- final public static int CROSSOVER=2 : Chromosomes de sorte croisement

- final public static int MUTATOR=3 : Chromosomes de sorte muatation

- private GraphChromo chromoStart : chromosome graphique racine du génome.

- private GraphChromo chromoSelect : chromosome sélectionné.

- private GraphChromo chromoSelectTree : racine du sous arbre sélectionné

- private ChromoNod copiedNod : racine du sous arbre copié.

- private JFrame mainFrame : frame principal du panel.

- private int type : sorte des chromosome du panel. (CREATE, INITIALIZE, CROSSOVER ou MUTATOR).

b) les méthodes

- public GraphPainter(int type, ChromoNod nod, JFrame frame) : constructeur. type est la sorte de chromosomes. nod est le chromosome racine (modèle) et frame la fenetre principale.

- public void setChromoNodStart(ChromoNod nod) : reinitialise avec un nouveau

chromosome racine nod (modèle).

- public JFrame getFrame() : retourne le frame

- public void paint(Graphics g) : affiche le génome sous forme d'arbre dans le graphic g.

- public void setSelect(ChromoNod chromoNod): chromoNod est sélectionné.

- public void setSelectTree(ChromoNod chromoNod) : le sous arbre de racine chromoNod est sélectionné.

- public GraphChromo getSelected() : retourne le chromosome sélectionné.

- public GraphChromo getSelectedTree() : retourne la racine du sous arbre sélectionné.

- public void setCopiedNod(ChromoNod chromo) : copie un sous arbre.

- public ChromoNod getCopiedNod() : retourne le sous arbre copié.

5) La classes GraphMainPainter

Cette classe affiche tous les chromosomes du génome. Elle contient des assenceurs lorsque le génome est trop grand. Elle continent un GraphPainter.

a) les méthodes

- public GraphMainPainter(int type, ChromoNod nod, JFrame frame) : constructeur. type est la sorte de chromosomes. nod est le chromosome racine (modèle) et frame la fenetre principale.

- public void setChromoNodStart(ChromoNod nod) : reinitialise avec un nouveau chromosome racine nod (modèle).

B) Les Dialogues de saisie de paramètres

B.1) Les dialogue de la partie création du génome

Lors de l'ajout ou la modification d'un chromosome du génome, le dialogue DialogCreate_ChooseType est appelé. Il permet de choisir le type du chromosome et par le jeu des sélecteurs, de le créer. Cette classe hérite de JDialog.

1) Le classe DialogCreate_ChooseType

a) les attributs

- private String[] comboBoxObj : liste des noms longs de tous les chromosomes.

- private SelectorCreate_Types typesSelector; : objet sélecteur de types. Contient la liste d tous les sélecteurs de chromsomes

- private ChromoNod chromoNod; : nouveau chromosome crée.

- private ChromoNod oldNod; : ancien chromosome qui va etre remplacé par le nouveau. (=null si absent)

- private static String lastType; : dernier type de chromosome choisit.

- private JComboBox comboBox; : objet combo box pour la sélection du type de chromosome.

- private JTextArea text; : zone de texte pour l'aide mémoire

- private int minTextRows=2; : nombre de ligne minimum de la zone de texte

b) les méthodes

- public DialogCreate_ChooseType(Frame owner) : constructeur. owner est la dialogue mère.

- public DialogCreate_ChooseType(Frame owner, ChromoNod oldNod) : constructeur. owner est la dialogue mère. et oldNod est le chromosome à remplacer.

- private void OK_Button(String name,String type) : Le bouton OK a été appuyé. name est le nom du chromsome et type sont type.

- private void setExplanations() : affiche l'aide mémoire sur le type de chromosome sélectionné

- public ChromoNod getChromoNod(): retourne le chromosome crée.

B.2) Les dialogues de la partie opérateur

Pour chaque sorte d'opérateur (XXX = Init, Mut ou Cross), nous avons huit dialogues qui gèrent l'ajout la modification, la suppression, la saisi de code, la définition de son poids d'un opérateur. Le premier dialogue ouvert est DialogXXX_Main

1) DialogXXX_Main

C'est le dialogue principal de la modification des opérateur sur un chromosome. Il permet de choisir une action ajout, modification, suppression ou import d'un opérateur. Il permet de choisir les poids des opérateurs.

a) les attributs

- private ChromoNod chromoNod : chromosome sur lequel les actions sont faites.

- private JPanel panelMain;: panel où se trouve la liste des opérateurs avec leur poids.

- private Vector goalVect; : liste des opérateurs du chromosome en cours de modifiacation. Si l'utilisateur appye sur le bouton "ok", elle devient effective.

- private Vector addVect : liste des opérateur à ajouter

- private Vector removeVect : liste des opérateurs à retirer

- private Vector changeVectOld; : liste des opérateur ayant été modifié (retiré et remplacé par des nouveaux)

- private Vector changeVectNew : liste des opérateur ayant remplacé ceux de changeVectOld (les index correspondent)

- private JButton buttonModify : bouton modifier

- private JButton buttonRemove : bouton retirer

b) les méthodes

- public DialogXXX_Main(Frame owner,ChromoNod chromo) : Constructeur, initialise graphiquement le dialogue avec tous ses composants et ses réflexes. Le frame owner est le fenetre mère et chromo est le chromosome sur lequel il les opérateurs sont modifiés.

- public void addChromoXXX(ChromoXXX_Root cross) : ajoute un opérateur, redessine le dialogue.

- public void removeChromoXXX(ChromoXXX_Root cross): supprime un opérateur, redessine le dialogue.

- public void changeChromoXXX(ChromoXXX_Root oldOp,ChromoXXX_Root newOp) : remplace l'opérateur oldOp par newOp. Le dialogue est reactualisé.

- public void redraw() : réactualise le dialogue, il est redessiné

- private void ok_button() : valide les modification et ferme le dialogue.

- private void cancel_button() : annule et ferme le dialogue

- private void add_button(): lance le dialogue ajout d'opérateur.

- private void remove_button(): lance le dialogue retrait d'opérateur.

- private void modify_button() : lance le dialogue modification d'opérateur.

- private void import_button(): lance le dialogue import. Il s'agit de copier un opérateur pour en créer un nouveau type défini par l'utilisateur.

3) DialogXXX_Root

Cet classe est abstraite. Elle généralise la sélection d'un opérateur. Cette classe hérite de JDialog.

a) les attributs

- protected ChromoCross_Root chromoCross : opérateur sélectionné

b) les méthodes

- public ChromoCross_Root getChromoCross() : renvoie l'opérateur séectionné.

c) les méthodes abstraites

- abstract protected void OK_button();: reponce à l'action appuye du bouton Ok.

4) DialogXXX_RemoveModify

Cet classe est abstraite. Elle généralise les actions de retrait et de modification (le choix de l'opérateur à remplacer) d'un opérateur. Cette classe hérite de DialogXXX_Root.

a) les attributs

- private Vector vectXXX; : liste des opérateur disponibles (à retirer ou à remplacer)
- private int selected=-1; : index de l'opérateur sélectionné dans la lise vectXXX. Par défaut -1, aucun opérateur sélectionné.

b) les méthodes

- public DialogXXX_RemoveModify(JDialog owner,Vector vectXXX) : constructeur. A pour dialogue mère owner. La liste des opérateurs disponible est dans vectXXX

- protected void OK_button() : le bouton Ok à été appuyé.

c) les méthodes abstraites

- abstract protected String getDialTitle() : titre du dialogue.

- abstract protected String getDialLabel() : petit texte qui explique l'action à effectuer.

5) les classes concrètes DialogXXX_Remove et DialogXXX_Modify

Elles redefinissent les deux méthodes abstraites hérité de DialogXXX_RemoveModify. Ces classes permettent de sélection un opérateur pour le retirer de la liste ou pour le remplacer par un autre.

a) les méthodes

- protected String getDialTitle() : titre du dialogue.

- protected String getDialLabel() : petit texte qui explique l'action à effectuer.

6) DialogXXX_ChooseSelect

Elle permet de sélectionner un sélecteur d'un opérateur, en vue de créer un nouvel opérateur. Trois modes sont possibles soit on crée un nouveau opérateur à partir de rien ou on crée un nouvel opérateur en remplaçant un ancien, soit on "import", c'est à dire on duplique et on modifie le nom et l'implémentation d'autre opérateur. En cas de modification l'ancien opérateur permet de d'initialiser les paramètres par défaut lors de la saisi. Cette classe hérite de DialogXXX_Root.

a) les attributs

- private ChromoXXX_Root cross (init ou mut) : nouvel opérateur crée.
- private JComboBox comboBox : boite de sélection.
- private JTextArea text : zone de texte de l'aide mémoire
- private SelectorCross_Root[] selectors : liste des sélecteurs d'opérateurs valides
- private ChromoNod chromoNod : chromosome sur lequel effectue l'opération.
- private int minTextRows=3 : nombre de lignes minimum de l'aide mémoire.
- private boolean addRem : prends la valeur vrais si le mode est ajout ou modification, et faux si le mode est import.

b) les méthodes

- public DialogXXX_ChooseSelect(JDialog owner,ChromoNod chromo,boolean addRem) : constructeur. owner est le dialogue mère, chromo le chromosome de l'opérateur et addRem sélectionne le mode (ajout ou modifier = vrais).

- public DialogXXX_ChooseSelect(JDialog owner,ChromoNod chromo,ChromoXXX_Root cross,boolean addRem) : constructeur. owner est le dialogue mère, chromo le chromosome de l'opérateur, cross (init ou mut) l'ancien opérateur et addRem sélectionne le mode (ajout ou modifier = vrais). Initialise les réflexes des composants.

- protected void OK_button() : valide les action. Stock le nouvel opérateur dans chromoNod.

- private void setExplanations() : affiche l'aide mémoire de l'opérateur

7) DialogXXX_User

Ces classes généralisent l'action de définition d'un nouvel opérateur défini par l'utilisateur. Cet opérateur va recevoir une implémentation. Ces classes saisissent le nom et l'implémentation de l'opérateur. Dans la zone implémentation, des lignes qui ne peuvent etre modifié font croire que la saisie est une fonction dont le nom dépends du nom de l'opérateur, son type et sa sorte. Cette fonction peut avoir un type de retours. En réalité le code généré est une méthode.

a) les attributs

- private JTextField textField : zone de saisie du nom de l'opérateur.

- private JTextArea text : zone de texte pour la saisi de l'implémentation.

- private String name : nom courant de l'opérateur.

- private boolean updatable=true: pour éviter un bouclage lors de la modification des parties constantes du texte. Un appel de updateName() va entrainer une modification du texte qui va vouloir appeler cette méthode de nouveau. Cet attribut est mis a faux dans updateName() pour emplécher une récursivité infinie, et mis a vrais pour autoriser l'appel de updateName().

- private ChromoCross_UserDefined cross (init ou mut);

- private String type : type de l'opérateur.

- private String objName=null : nom de l'opérateur après appuye de "ok"

- private String objImplem : implémentation

- private String varName : le nom de la variable (le chromosome) passé paramètre à la fonction (méthode)

b) les méthodes

- public DialogXXX_User(JDialog owner,String type,String varName) : constructeur. owner est le dialogue mère, type le type de l'opérateur, varName le nom de la variable (le chromosome) passé paramètre à la fonction (méthode).

- public DialogXXX_User(JDialog owner, String type,String varName,ChromoXXX_Root chromoXXX) : constructeur. owner est le dialogue mère, type le type de l'opérateur, varName le nom de la variable (le chromosome) passé paramètre à la fonction (méthode), chromoXXX l'ancien opérateur à modifier. Ce constructeur met en forme et ajoute les reflexes aux composants.

- protected void OK_button() : le bouton ok a été appuyé. Il faut mémorise le nom et l'implémentation de l'opérateur.

- private void setCrossName(String newName): insère un nouveau nom d'opérateur. Les caractères interdits sont remplacé par un '_'. updateName est appellé.

- private void updateName() : met en forme les composants et le code produit pour simuler la saisie d'une fonction. La saisie du code de l'opérateur est une méthode mais pour des raisons d'érgonomie, la saisie ressemble à une fonction C++, dont la signature et le code de retours ne peuvent pas etre modifié. Si elle le sont elles sont reécrites par cette méthode. La signature dépends du nom de l'opérateur.

- public String getObjName() : retourne le nom de l'opérateur.

- public String getObjImplem() : retourne l'implémention (sans signature et ligne de retours) de l'opérateur.

- public String getFuncDeclaration(String type,String name,String varName) : retourne la signature de la fonction (destiné à etre affiché à l'écran)

c) les méthodes abstraite

- abstract protected boolean isNamesDifferentOrNotAllowedToBeSame(String n1,String

n2) : retourne faux si n1 et n2 sont autorisé a etre identiques. Lors de la modification il est autorisé de conserver le meme nom. Alors que dans un import c'est interdit.

8) DialogXXX_User_Define et DialogXXX_User_Import

Les classes DialogXXX_User_Define définissent des opérateur nouveau ou des modification d'un opérateur définit par l'utilisateur existant. Les classes DialogXXX_User_Import crée un nouvel opérateur à partir d'une copie d'un autre.

a) les méthodes

- public DialogXXX_User_YYYY(JDialog owner,String type,String varName) : constructeur. owner est le dialogue mère, type le type de l'opérateur, varName le nom de la variable (le chromosome) passé paramètre à la fonction (méthode).

- public DialogXXX_User_YYY(JDialog owner, String type,String varName,ChromoXXX_Root chromoXXX) : constructeur. owner est le dialogue mère, type le type de l'opérateur, varName le nom de la variable (le chromosome) passé paramètre à la fonction (méthode), chromoXXX l'ancien opérateur à modifier. Ce constructeur met en forme et ajoute les reflexes aux composants.

- abstract protected boolean isNamesDifferentOrNotAllowedToBeSame(String n1,String n2) : retourne faux si n1 et n2 sont autorisé a etre identiques. Lors de la modification il est autorisé de conserver le meme nom. Alors que dans un import c'est interdit.

B.3) Les autres dialogues saisi de paramètres

Ces dialogues sont spécialisés dans la saisie de paramètres. Il s'adaptent aussi bien à la partie création que la partie opérateur. Ils héritent tous de JDialog.

1) Dialog_IntMinMax

Permet la saisie de deux valeurs minimum et maximum pour des entiers. Ces valeurs peuvent etre plus ou moins l'infinit.

a) les attributs

- private JTextField textFieldMin : champs minimum

- private JCheckBox checkBoxMin : boite moins l'infini

- private JTextField textFieldMax : champs maximum

- private JCheckBox checkBoxMax : boite plus l'infini

- private boolean hasCancel : l'action à été annulé

- private static int lastMin =0 : minimum par défaut

- private static int lastMax =0 : maximum par défaut

- private int min=0 : valeur minimal courante

- private int max=0 : valeur maximale courante

b) les méthodes

- public Dialog_IntMinMax(JDialog owner) constructeur. owner est le dialog mère.

- public Dialog_IntMinMax(JDialog owner, int oldMin, int oldMax) constructeur. owner est le dialog mère. oldMin et oldMax les valeurs par défaut. Initialise les réflexes et l'aspect graphique.

- private void cancel() : action annuler

- protected void OK_button() : action "ok"

- public int getMin() throws ExpCancel : retourne le minimum

- public int getMax() throws ExpCancel : retourne la maximum

2) Dialog_RealMinMax

Permet la saisie de deux valeurs minimum et maximum pour des réels. Ces valeurs peuvent etre plus ou moins l'infinit.

a) les attributs

- private JTextField textFieldMin : champs minimum

- private JCheckBox checkBoxMin : boite moins l'infini

- private JTextField textFieldMax : champs maximum

- private JCheckBox checkBoxMax : boite plus l'infini

- private boolean hasCancel : l'action à été annulé

- private static double lastMin =0 : minimum par défaut

- private static double lastMax =0 : maximum par défaut

- private double min=0 : valeur minimal courante

- private double max=0 : valeur maximale courante

b) les méthodes

- public Dialog_RealMinMax(JDialog owner) constructeur. owner est le dialog mère.

- public Dialog_RealMinMax(JDialog owner, double oldMin, double oldMax) constructeur. owner est le dialog mère. oldMin et oldMax les valeurs par défaut. Initialise les réflexes et l'aspect graphique.

- private void cancel() : action annuler

- protected void OK_button() : action "ok"

- public double getMin() throws ExpCancel : retourne le minimum

- public double getMax() throws ExpCancel : retourne la maximum

3) La classe Dialog_K

Paramètre K. K peut etre le nombre de mutation à effectuer dans un bag par exemple.

a) les attributs

- private JTextField textField : champs de saisie

- int K = 1 : Valeur du paramètre

- static int lastParam = 1 : valeur par défaut.

b) les méthodes

- public Dialog_K(JDialog owner) : constructeur. owner est le dialog mère.

- public Dialog_K(JDialog owner, int oldValue) : constructeur. owner est le dialog mère. oldValue est la valeur par défaut. Initialise les réflexes et l'aspect graphique.

- protected void OK_button() : action "ok"

- public int getK() : retourne la parmètre K

4) La classe Dialog_Proba

La classe insère une probabilité.

a) les attributs

- private JTextField textField : champs de saisie

- double proba =0.5: Valeur du paramètre

- static double lastParam =0.5 : valeur par défaut.

b) les méthodes

- public Dialog_Proba(JDialog owner) : constructeur. owner est le dialog mère.

- public Dialog_Proba(JDialog owner, double oldValue) : constructeur. owner est le dialog mère. oldValue est la valeur par défaut. Initialise les réflexes et l'aspect graphique.

- protected void OK_button() : action "ok"

- public double getProba() : retourne la paramètre proba

5) La classe Dialog_Size

La classe insère une taille. Pour les bag et vect ainsi que pour les permut. Peut etre utilisé ailleur.

a) les attributs

- private JTextField textField : champs de saisie

- static int lastParam =0 : valeur par défaut.

- int size =0: Valeur du paramètre

- boolean hasCancel : l'utilisateur a annulé?

b) les méthodes

- public Dialog_Size(JDialog owner) : constructeur. owner est le dialog mère.

- public Dialog_Size(JDialog owner, int oldSize) : constructeur. owner est le dialog mère. oldSize est la valeur par défaut. Initialise les réflexes et l'aspect graphique.

- private void OK_button() : action "ok"

- public int getNbElem() : retourne la paramètre proba

C) classes de confirmation de retrait d'opérateurs (modèle - vue - contrôle)

Lorsque l'un chromosome est ajouté, retiré ou modifié, les opérateurs définis par l'utilisateur peuvent ne plus etre valable. Si seul des modifications sur les paramètres d'un chromosome le prblème ne se pose pas. Par contre si le type d'un chromosome est modifié le problème se pose. Puisqu'il peuvent utiliser leur descendants dans leur code et devenir erroné, il faut demander à l'utilisateur s'il veut les garder. Le problème c'est que ces modifications concernent le modèle et que le modèle ne peut demander directement à l'utilisateur s'il veut garder ou non. Parsque une telle action releve de la vue et du controle. L'idée est de créer une classe du modèle qui va mémoriser les opérateur et les chromosomes litigieux et lancera un événement à une classe de la vue - controle. La classe du modèle s'appelle ExamineOperModel et celle de la vue ExamineOperVue. Un objet ExamineOperModel est crée comme attribut dans le constructeur de chaque chromosome. Et un objet ExamineOperVue est crée dans chaque objet chromosome coté vue (GraphChromo). Pour que les événements soient interceptés, il faut que les objets ExamineOperModel soient incrit à un objet ExamineOperVue. Cette action est faite dans le constructeur GraphChromo. Cette action est donc effective pour tout chromosome affiché dans à l'écran, ce qui est toujours le cas.

1) La classe ExamineOperModel

a) les attributs

- private Vector chromoToExamine : liste des chromosomes à examiner. chromoToExamine et operatorToExamine sont liées et sont dans le meme ordre.

- private Vector operatorToExamine : liste des opérateur a examiner. chromoToExamine et operatorToExamine sont liées et sont dans le meme ordre.

b) les méthodes

- public ExamineOperModel() : constructeur

- public synchronized void push(ChromoNod chromo, ChromoOperator oper) : ajout d'un nouvel opérateur à examiner. Le chromosome sur lequel est l'opérateur oper est chromo. Cette méthode lance un événement en direction de la vue.

- public synchronized ChromoNod[] getChromoToExamine() : retourne la liste des chromosomes à examiner. Fonctionne avec getOperatorToExamine()

- public synchronized ChromoOperator[] getOperatorToExamine() : retourne la liste des opérateurs a examiner. Fonctionne avec getChromoToExamine()

2) La classe ExamineOperVue

a) les méthodes

- public ExamineOperVue() : constructeur

- public void update(Observable obs,Object obj) : intercepte les événements du modèle. La variable obs est l'objet qui lance l'événement et obj les paramètres. Dans notre cas obj n'est pas utilisé. Cette méthode va ouvrir un dialogue et demander la confirmation pour retirer l'opérateur du chromosome.

V) Le génération de code

Le but du programme est de générer du code. Ce code est écrit en C++ avec une librairie EO. Il comprends plusieurs fichiers. Si "Data" est le nommé aux données, un fichiers "DataEA.cpp"(corps de l'algorithme) , "eoData.h"(génome), "eoDataInit" (initilisation du génome), "eoDataMutation.h" (mutation), "eoDataQuadCrossover.h"(croisement) , "eoDataEvalFunc.h" (fonction d'évaluation), "eoDataStat.h"(statistiques) et un "Makefile" sont crées. Chaque fichier est généré par une classe. La classe GenerateCode est le point d'entrée de la génération des fichiers. Cette classe va appeller les autres classes et sauvegarder leur fichier.

Il faut remarquer que les nom et le type des chromosomes et opérateurs génèrent des noms de variables. Il faut donc que ces noms soient correctement écris (avec des lettres, des chiffres et des '_' et rien d'autre). Pour chaque fichier créé, il existe une classe.

1) La classe GenerateCode

Cette classe est le point d'entrée de la génération de fichiers. Elle a une méthode qui sauvegarde les fichiers. Le contenus des fichiers est démandé aux classes spécialisées. Si un fichier Makefile est déjà présent, une requette demande à l'utilisateur s'il veut la supprimer pour un nouveau.

a) les méthodes

- public GenerateCode() : constructeur.

- public void generateCode(String name,String eoPath,ChromoNod chromo, String evalFunc, bolean minimisation) :génére tous les fichiers. name est le nom des données. eoPath le répertoire. chromo est la racine du génome. evalFunc la fonction d'évaluation. Si minimisation est vrais la fonction d'évaluation est minimisé sinon elle est maximisé.

- private void saveFile(String fileName, Vector vect) : écrit sur disque dans le fichier fileName (répertoire + nom de fichier), les lignes de code contenus dans vect.

- private Vector readInclude(String userPath,String space) : des fichiers peuvent contenir des lignes de code venant de l'extérieur. Plus exactement des lignes de code contenus dans un fichier externe. Si ce fichier est absent rien n'est ajouté. Cette méthode lit le fichier userPath (répertoire + nom de fichier), insère des espaces "space" en debut de ligne et retourne le code à insèrer.

2) La classe GenerateCodeMain

Cette classe génère un vecteur de chaine de caractères, un élément par ligne qui correspond au code principal de l'algorithme. C'est l'ossature du programme généré en EO.

a) les méthodes

- public GeneratorCodeMain() : constructeur

- public Vector getCode(String name, Vector userInclude, bolean minimisation) : retourne le code du fichier "DataEA.cpp" . name est le nom des données (Data) et userInclude sont les lignes de code à insèrer (cf GeneratorCode). Si minimisation est vrais la fonction d'évaluation est minimisé sinon elle est maximisé.

3) La classe GenerateCodeGenome

Cette classe génère un vecteur de chaine de caractères, un élément par ligne qui correspond au code du génome.

a) les attributs

- private final String eoAbstractVector="eoAbstractVector" : nom du type "AbstractVector".

- private Vector vectGenomeTypes= new Vector() : liste des types rencontrées (les templates de meme type sont considérées identiques)

- private Vector vectIOGenomeTypes= new Vector() : liste des types rencontrées (les templates de meme type sont considérées différents)

b) les méthodes

- public GeneratorCodeGenome() : constructeur

- public Vector getCode(String appliName,ChromoNod chromo) : retourne le code du fichier "eoData.h" . appliName est le nom des données (Data) et chromo est la racine du génome. Elle appel generateChromoRec() sur la racine.

- private Vector generateChromoRec(ChromoNod chromo) : génére recursivement le code qui correspond à un type de chromosome.

- private Vector generateAbstractVector() : retourne le code correspondant a AbstractVector. Cette classe est la classe mère de bag, list, set et vect (chromosomes conteneurs d'éléments)

4) La classe GenerateOperators

Cette classe généralise les comportements de GenerateCodeInit, GenerateCodeMutation et GenerateCodeQuadCrossover.

b) les méthodes

- abstract Vector getCode(String name,ChromoNod chromo) : généralise l'accès au générateur de code.

- protected String getOpParameters(ChromoNod chromo,ChromoOperator oper,String init) : retourne la signature des paramètres lors de la construction d'un nouvel opérateur. chromo est le chromosome courant. oper l'opération en question et init est le nom du chromosome enfant pour les chromosomes collections. Si le chromosome ne veut pas accéder aux opérateurs d'initialisation sur son enfant ou si le chromosome n'est pas une collection d'éléments alors cette chaine de caractères doit etre vide.

5) Les classes GenerateCodeInit, GenerateCodeMutation et GenerateCodeQuadCrossover

Ces classes génèrent le code des classes EO d'initialisation, de mutation et de croisement. Il faut générer le code des classes des opérateurs et créer un objet qui effectue toutes les opérations de la sorte sur le tout génome. Pour cela il faut une classe

qui encapsule tous les opérateurs de la sorte pour chaque chromosome. Toute trois hérite de GenerateOperators.

a) les attributs

- private Vector vectFonctorsDef = new Vector() : liste des classes déjà générées.

b) les méthodes

- public GeneratorCodeInit() : constructeur

- public Vector getCode(String appliName,ChromoNod chromo) : retourne le code du fichier "eoDataXXX.h" (XXX = "Init", "Mutation" ou "QuadCrossover" . appliName est le nom des données (Data) et chromo est la racine du génome. Les méthodes uivantes sont appelées.

- private Vector generateChromoRec(ChromoNod chromo,String appliName) : génère l'implémentation des classes correspondant aux opérateurs par recurence. appliName est le nom des données (Data) et chromo est le chromosome courant.

- private Vector generateInitialCst(ChromoNod chromo,String appliName) : initialise l'ensembles des variables correspondant aux opérateurs sur les chromosomes (appel generateInitialCstRec). appliName est le nom des données (Data) et chromo est la racine du génome.

- private Vector generateInitialCstRec(ChromoNod chromo,String appliName) : initialise l'ensembles des variables correspondant aux opérateurs sur les chromosomes par récurence. appliName est le nom des données (Data) et chromo est le chromosome courant.

- private Vector generateInitialVarRec(ChromoNod chromo) : déclare l'ensembles des variables correspondant aux opérateurs sur les chromosomes par recurence. chromo est le chromosome courant

- private Vector generateInitialDelRec(ChromoNod chromo) : détruit (appel de destructeur) les variables correspondant aux opérateurs sur les chromosomes par recurence. chromo est le chromosome courant

c) cas particulier de GenerateCodeInit

Cette classe crée des accesseurs sur les opérateurs ou liste d'opérateurs de chaque

chromosome.

- private Vector generateInitialAccessRec(ChromoNod chromo) : déclare l'ensembles des accesseurs correspondant aux opérateurs sur les chromosomes par recurence. chromo est le chromosome courant.

6) La classe GenerateCodeEvalFunc

Cette classe génère un vecteur de chaine de caractères, un élément par ligne qui correspond au code du génome. Elle génère le code du fichier eoDataEvalFunc.h

a) les méthodes

- public GeneratorCodeEvalFunc() constructeur

- public Vector getCode(String name, ChromoNod chromo, String evalFunc, Vector userEvalFile) : retourne le code généré. Appel le préprocesseur addEvalFunction(). name est le nom des données. chromo le chromosome racine. evalFunc le code saisi de la fonction d'évaluation. Et userEvalFile est du code inséré par un fichier externe.

- private void addEvalFunction(Vector vect, ChromoNod chromo, String evalFunc): retourne une liste de lignes de code pour la fonction d'évaluation appel addEvalFunctionRec() sur le chromosome racine chromo. evalFunc est la fonction d'évaluation avant préprocesseur. Le résultat est stocké dans vect.

- private String addEvalFunctionRec(ChromoNod chromo, String evalFunc) : partie recursive sur le chromosome chromo. La fonction d'evaluation evalFunc sera modifié. Une variable du nom de chromo sera changé en une méthode "get" si un point ou un '->' se trouve directement devant. Un espace devant cette variable inhibe l'action.

7) La classe GenerateCodeStat

Génère le code du fichier statistiques.

a) les méthodes

- public GenerateCodeStat() : constructeur

- public Vector getCode(String name) : retourne le code du fichier "eoDataStat.h" . name est le nom des données (Data).

) La classe GenerateCodeStat

Génère le code du fichier statistiques.

8) La classe GenerateMakefile

Génère le code du fichier Makefile.

a) les méthodes

- public GenerateMakefile() : constructeur

- public Vector getCode(String name, String eoPath) : retourne le code du fichier "Makefile" . name est le nom des données (Data) et eoPath est le chemin de la librairie EO.

VI) La sauvegarde et le chargement de données.

Les données de travail sont sauvegardées dans un fichier XML. La classe IOGenome gère la sauvegarde et le chargement de ces fichiers. Le fichier est découpé en trois parties. Il liste tous les opérateurs attaché à au moins un chromosome. Il donne la fonction d'évaluation puis il donne le génome.

1) La DTD du fichier XML

```
<!ELEMENT REPRESENTATION (INITIALISATOR*,CHROMOSOME)>
    <!ELEMENT INITIALISATOR (#PCDATA)>
    <!ATTLIST INITIALISATOR id ID #REQUIRED
                    type NMTOKEN #REQUIRED
                    name NMTOKEN #REQUIRED
                    varname NMTOKEN #REQUIRED
                    userdefined (yes|no) #REQUIRED
                    uniquetype PCDATA #IMPLIED>
    <!ELEMENT MUTATOR (#PCDATA)>
    <!ATTLIST MUTATOR id ID #REQUIRED
                    type NMTOKEN #REQUIRED
```

```
                    name NMTOKEN #REQUIRED
                    varname NMTOKEN #REQUIRED
                    userdefined (yes|no) #REQUIRED
                    uniquetype PCDATA #IMPLIED>
    <!ELEMENT CROSSOVER (#PCDATA)>
    <!ATTLIST CROSSOVER id ID #REQUIRED
                    type NMTOKEN #REQUIRED
                    name NMTOKEN #REQUIRED
                    varname NMTOKEN #REQUIRED
                    userdefined (yes|no) #REQUIRED
                    uniquetype PCDATA #IMPLIED>
    <!ELEMENT EVAL_FUNCTION (#PCDATA)>
    <!ATTLIST EVAL_FUNCTION nbHelpLines PCDATA #IMPLIED>
    <!ELEMENT CHROMOSOME (PARAMETERS*,
                                CHROMO_INITIALISATOR*,
                                CHROMO_MUTATOR*,
                                 CHROMO_CROSSOVER*,
                                 CHROMOSOME*)>
    <!ATTLIST CHROMOSOME type NMTOKEN #REQUIRED
                    name NMTOKEN #REQUIRED>
    <!ELEMENT CHROMO_INITIALISATOR (PARAMETERS*)>
    <!ATTLIST CHROMO_INITIALISATOR id IDREF #REQUIRED weight
PCDATA #REQUIRED>
    <!ELEMENT CHROMO_MUTATOR (PARAMETERS*)>
    <!ATTLIST CHROMO_MUTATOR id IDREF #REQUIRED weight PCDATA
#REQUIRED>
    <!ELEMENT CHROMO_CROSSOVER (PARAMETERS*)>
```

<!ATTLIST CHROMO_CROSSOVER id IDREF #REQUIRED weight PCDATA #REQUIRED>

<!ELEMENT PARAMETERS>

<!ATTLIST PARAMETERS name NMTOKEN #REQUIRED

value PCDATA #REQUIRED >

La balise REPRESENTATION est le corps du fichier.

Les balises INITIALISATOR, MUTATOR et CROSSOVER définissent tous les opérateurs utilisées par le génome. Le code inséré entre les balises ouvrantes et fermantes est l'implémentation des opérateurs défini par l'utilisateur (rien dans le cas des opérateurs prédéfini). Les attributs id est l'identifiant de l'opérateur, utilisé plus loin. Le tpe est répresenté par type, et son type unique par uniquetype. Le nom de l'opérateur est "name", son varName est varname. L'attribut userdefined prends les valeur "yes" pour opérateur défini par l'utilisateur et "no" pour prédéfini.

La balise EVAL_FUNCTION est la fonction d'évaluation. L'attribut nbHelpLines est ne nombre de lignes d'aide mis en commentaire dans le fonction d'évaluation.

La balise CHROMOSOME désigne un chromosome. Les chromosomes peuvent s'encastrer les un dans les autres. Il peuvent avoir des paramètres et des opérateurs. Un chromosome a un type "type" et un nom "name".

Les balises CHROMO_INITIALISATOR, CHROMO_MUTATOR et CHROMO_ CROSSOVER sont les objet opérateur attaché au chromosome. Ils ont un id utilisé plus haut dans la liste des opérateurs et un poids "weight". Il peuvent contenir des paramètres.

La balise PARAMETERS représente un paramètre qu'il soit attaché à un chromosome ou un opérateur.

2) La classe IOGenome

a) les attributs et classes privées

- private class Initialisator : contient tous les informations sur une opération d'initialisation (avant de la créer)

- private class Mutator : contient tous les informations sur une opération de mutation

(avant de la créer)

- private class Crossover : contient tous les informations sur une opération de croisement (avant de la créer)

-private int nbHelpVarNames=0 : est le nombre de lignes d'aide mis en commentaire dans la fonction d'évaluation.

- private String _evalFunc=null; la fonction d'évaluation

- private Vector savedInitList= new Vector() : objets initialisations

- private Vector savedInitIDList= new Vector() : id des objets initialisation

- private Vector savedMutList= new Vector() : objets mutations

- private Vector savedMutIDList= new Vector() : id des objets mutation

- private Vector savedCrossList= new Vector() : objets croisements

- private Vector savedCrossIDList= new Vector() : id des objets croisement

- private SelectorCreate_Types selectorTypes : sélecteur de types.

b) les méthodes

- public IOGenome() : constructeur

- public void save(String fileName,ChromoNod chromo,String evalFunc,int nb) : sauvegarde dans le fichier fileName (sans l'extension ".xml"), dont la racine du génome est chromo et la fonction d'évaluation evalFunc. nb est le nombre de lignes d'aide mis en commentaire dans la fonction d'évaluation.

- private void saveChromoRec(ChromoNod chromo,Vector vect,String space) : sauvegarde les données d'un chromosome, ses opérateurs, ses paramètres, les paramètres des opérateur et poursuit par récurrence sur les chromosomes enfants. chromo est le chromosome courant, vect les lignes xml générées et space la tabulation de debut de ligne.

- private void saveInitRec(ChromoNod chromo,Vector vect) : sauvegarde par récurrence tous les opérateurs d'initialisation trouvées. chromo est le chromosome courrant et vect les lignes xml générées.

- private void saveMutRec(ChromoNod chromo,Vector vect) : sauvegarde par récurrence tous les opérateurs de mutation trouvées. chromo est le chromosome courrant et vect les lignes xml générées.

- private void saveCrossRec(ChromoNod chromo,Vector vect) : sauvegarde par récurrence tous les opérateurs de croisement trouvées. chromo est le chromosome courrant et vect les lignes xml générées.

- private void saveEvalFunc(String evalFunc, int nbHelpVarNamesVector vect) : sauvegarde la fonction d'évaluation. Le paramètre vect contient les lignes xml générées. nbHelpVarNamesVector est le nombre de lignes d'aide mis en commentaire dans la fonction d'évaluation.

- public ChromoNod load(String fileName) : lire le fichier fileName (avec extension ".xml").

- private void loadInitialisor(Node aNode) : lit la liste des opérateurs d'initialisation et les stocke en mémoire. Le noeud aNode est le noeud courant dans le fichier XML.

- private void loadMutator(Node aNode) : lit la liste des opérateurs de mutation et les stocke en mémoire. Le noeud aNode est le noeud courant dans le fichier XML.

- private void loadCrossover(Node aNode) : lit la liste des opérateurs de croisement et les stocke en mémoire. Le noeud aNode est le noeud courant dans le fichier XML.

- private String loadEvalFunc(Node aNode) : lit la liste la fonction d'évaluation. Le noeud aNode est le noeud courant dans le fichier XML.

- private ChromoNod getChromosome(Node aNode) : lit un chromosome et crée l'objet correspondant avec ses opérateurs. Le noeud aNode est le noeud courant dans le fichier XML.

- private ChromoInit_Root getInitialisor(Node aNode) : lit les attribut d'un opérateur pour créer un objet opérateur d'initialisation.

- private ChromoMut_Root getMutator(Node aNode) : lit les attribut d'un opérateur pour créer un objet opérateur de mutation.

- private ChromoCross_Root getCrossover(Node aNode) : lit les attribut d'un opérateur pour créer un objet opérateur de croisement.

- private String[] getParamSaveNameValue(ChromoNod chromo) : retourne une suite de

ligne. Une ligne correspond à un paramètre du chromosome chromo.

- private String[] getParamSaveNameValue(ChromoOperator operator): retourne une suite de ligne. Une ligne correspond à un paramètre de l'opérateur operator.

- private void loadParameter(Node aNode,Vector paraNames,Vector paraValues) : lit une balise paramètre du noeud aNode. Le nom du paramètre sera mis dans paraNames et sa valeur dans paraValues (ils ont meme indice).

-public int getNbHelpVarNames(): retourne le nombre de lignes d'aide mis en commentaire dans la fonction d'évaluation.

VII) Les exceptions

Cinq classes représentent les exceptions du programme.

1) ExpBadParameters

Cette exception est levée quant les paramètres saisis dans un opérateur ou un chromosome sont incorrect. Comme par exemple une probabilité négative ou supérieur à un.

2) ExpCancel

Cette exception est levée pour signaler que l'utilisateur à annulé son action.

3) ExpContainerNoChild

Lors de la génération de code, cette exception est généré pour indiquer un conteneur sans enfants.

4) ExpNameInDouble

Lors de la génération de code, cette exception est généré pour indiquer que deux chromosomes ont le meme nom.

5) ExpNameIncorrect

Lors de la construction d'un chromosome, cette exception signale un nom incorrect.

VIII) Conclusion

La nouvelle version de GUIDE permet aux utilisateurs de créer facilement des algorithmes évolutionnaires. L'interface se veut intuitive et ergonomique. Le code à été écrit dans un soucis de clarté et de logique. J'espère que les futurs développeurs pourrons la saisir facilement et réussiront à faire évoluer GUIDE sans trop de problèmes.