Ajout d'un chromosome

I) Introduction

L'ajout d'un chromosome se fait en cinq phases. Il faut créer la classe qui correspond au chromosomes. Pour pouvoir être sélectionné et pour pouvoir créer un exemplaire soit par une action utilisateur, soit par une action de chargement, il faut un sélecteur pour ce chromosome. Pour être visible ce sélecteur doit être inscrit dans une liste. A partir de là le chromosome existe mais il est impossible de créer des opérateur sur lui. Il faut trois nouvelles classes qui généralisent le comportement des opérateurs pour ce chromosomes. Il faut encore trois classes "sélecteur de types" qui donnent la liste des opérateurs disponibles pour ce chromosomes. A partir de là le chromosome existe dans Guide.

II) Le chromosome coté modèle

Le chromosome s'appelle soit ChromoNodXXX ou ChromoNodLeafXXX avec XXX le nom du chromosome. Si le chromosome est de type feuille alors il faut l'appeller ChromoNodLeafXXX. Si au contraire il est de type conteneur, il faut l'appeler ChromoNodXXX.

Il peut hériter de plusieurs classes abstraites. Si le chromosome est un type feuille sans descendants, il doit hériter de ChromoNodTerminal. S'il est un conteneur à enfants simple, il hérite de ChromoNodChildSimple. Si au contraire il peut avoir plusieurs enfants, il doit hériter de ChromoChildMult.

Les nom des types ainsi que des paramètres doivent se trouver en un seul endroit. Il peuvent être modifié facilement à cette condition.

```
//déclaration du chromosome réel.
public class ChromoNodLeafReal extends ChromoNodLeafTerminal{

//liste des paramètres mis sou forme static public.
   static public String min ="min";
   static public String max ="max";

//type de base, forme abrégé va servir à écrire des nom de variables en EO. Il ne peut
```

```
//donc prendre que des lettres, des chiffres et le caractère souligné.
    static public String type(){
        return "real";
    }


//type long, sert pour l'affichage à l'écran, peut prendre tous les caractères.
    static public String longType(){
        return "real";
    }


//aide mémoire sur le chromosome. Il est intéressant de faire apparaitre les paramètres avec leur type C++ du chromosomes dans l'aide mémoire, l'utilisateur peut les manipuler dans les opérateurs qu'il définit.
    static public String[] explanations(){
        String[] stg = new String[4];
        stg[0]= new String("Real :  floating point approximation of real values");
        stg[1]= new String("Parameters:");
        stg[2]= new String("    min(double) : minimum value");
        stg[3]= new String("    max(double) : maximum value");
        return stg;
    }

//paramètres du chromosome
    private double _min;
    private double _max;
```

```
//constructeur prev est le chromosome parent et name est son nom.
     public ChromoNodLeafReal(ChromoNod prev, String name, double min, double max) throws ExpNameIncorrect{
        super(prev,name);
        if(min<max) {
           this._min=min;
           this._max=max;
        }else{
           this._min=max;
           this._max=min;
        }
    }


//retourne le type de base
    public String getType(){
        return type();
    }


    public String getLongType(){
        return longType();
    }


//retourne le type complet (type long + valeur des paramètres)
    public String getFullType(){
        return longType()+" ["+_min+" , "+_max+"]";
    }
```

```
//clonnage
    public Object clone() throws CloneNotSupportedException{
        return super.clone();
    }

//accesseur aux paramètres
    public double getMax(){
        return _max;
    }
//accesseur aux paramètres
    public double getMin(){
        return _min;
    }

//retourne un vecteur de lignes de codes.
//Il s'agit de la définition du chromosome. Ce peut etre une classe ou un typedef
//comme ici. Un constructeur sans paramètres doit toujours exister.
    public Vector getGenomeCodeLines(){
        Vector vect = new Vector();
        vect.add("typedef double real;");
        return vect;
    }

//initialisateur par défaut (existe de préférence)
    public ChromoInit_Root getDefaultInitialisator(){
        return new ChromoInitRealUniform();
    }
```

```
//mutation par défaut (existe de préférence)
    public ChromoMut_Root getDefaultMutator(){
        return new ChromoMutRealFixedNormal();
    }


//croisement par défaut (existe de préférence)
    public ChromoCross_Root getDefaultCrossover(){
        return new ChromoCrossRealExchange();
    }

//retourne la liste des type (en C++) des paramètres
    public String[] getParamType(){
        String [] stg = new String[2];
        stg[0]="double";
        stg[1]="double";
        return stg;
    }

//retourne la liste des nom des paramètres
    public String[] getParamName(){
        String [] stg = new String[2];
        stg[0]=min;
        stg[1]=max;
        return stg;
    }
```

```
//retourne la liste des valeurs des paramètres
    public String[] getParamValue(){
        String [] stg = new String[2];
        stg[0]=""+_min;
        stg[1]=""+_max;
        return stg;
    }

//retourne vrais si le type est scalaire (par opposition à une classe)
    public boolean isScalarType(){
        return true;
    }
}

//autres méthodes redéfini par ChromoNodLeafTerminal
//retourne vrais si le chromosomes est de type template
    public boolean isTemplateType(){
        return false;
    }

//retourne vrais si le type du chromosome est un conteneur d'éléments à enfant
//simple
    public boolean isSingleVectorTypeChild(){
        return false;
    }

//retourne vrai si le chromosome est de type racine du génome
```

```
    public boolean isRoot(){
        return false;
    }
```

III) Le chromosome coté vue (sélecteur)

Cet classe sert à sélectionner un chromosome. Le sélecteur une fois trouvé va permettre de créer un nouveau chromosome, soit après une action utilisateur, soit après une action chargement.

```
//Les sélecteurs héritent de SelectorCreate_Root.
public class SelectorCreateReal extends SelectorCreate_Root{

//constructeur
    public SelectorCreateReal() {
    }


//retourne l'aide mémoire
    public String[] getExplanations(){
        return ChromoNodLeafReal.explanations();
    }


//retourne le type
    public String getType(){
        return ChromoNodLeafReal.type();
    }
```

```
//retourne le type long
    public String getLongType(){
        return ChromoNodLeafReal.longType();
    }

//retourne un chromosome du type voulu. Dans notre cas des paramètres sont
//demandées. La variable oldNod est l'ancien chromosome qui va être remplacé
//par le nouveau (si absent = null).
    public ChromoNod create(JDialog owner,String name,ChromoNod oldNod)
throws ExpNameIncorrect, ExpCancel, ExpBadParameters{
        Dialog_RealMinMax dialogParam;
        if(oldNod!=null&&ChromoNodLeafReal.class.isInstance(oldNod)){
            ChromoNodLeafReal oldNod2 = (ChromoNodLeafReal)oldNod;
            dialogParam = new Dialog_RealMinMax(owner, oldNod2.getMin(),
oldNod2.getMax());
        }else{
            dialogParam = new Dialog_RealMinMax(owner);
        }
        return new ChromoNodLeafReal(null,name,dialogParam.getMin(),
dialogParam.getMax());
    }

//charge à partir d'un fichier le chromosome. la variable paraNames et paraValues
//sont les liste des noms et valeurs des paramètres lus sur le disque. Pour obtenir
//un paramètre la méthode getParam(nom, paraNames, paraValues) peut être utilisé.
//Elle retourne la valeur de nom trouvé dans paraNames et paraValues. Si une
//exception est levée, la méthode l'intercepte et lève une exception IOException
//(exception de chargement)
    public ChromoNod load(String name,Vector paraNames,Vector paraValues)
```

```
throws IOException{
      try{
         float min = (new Float( getParam
(ChromoNodLeafReal.min,paraNames,paraValues))).floatValue();
         float max = (new Float( getParam
(ChromoNodLeafReal.max,paraNames,paraValues))).floatValue();
         return new ChromoNodLeafReal(null,name,min,max);
      }catch(ExpNameIncorrect exp){
         throw new IOException();
      }
   }
}
```

IV) La mise à jour de la liste des chromosomes

La classe *SelectorCreate_Types* contient plusieurs méthodes pour gérer l'ensembles des sélecteurs de chromosomes. C'est dans cette classe qu'il faut ajouter le sélecteur du nouveau chromosome pour qu'il soit visible. Il faut l'ajouter au tableau *selectors*. Cet attribut contient tous les sélecteurs de tous les chromosomes, à l'exception du chromosome racine qui est à part. Il est impossible à l'utilisateur de créer un chromosome racine mais il doit pouvoir être chargé du disque.

V) Classes mères des opérateur (modèle)

Pour un chromosome nommé XXX, il faut trois classes dans le modèle, pour construire les opérateurs. Il faut ChromoInit_TypeXXX, ChromoMut_TypeXXX et ChromoCross_TypeXXX. Chacune de ces classes généralise le type des opérateurs (égal au type du chromosome).

```
//L'opérateur est prédéfini, il hérite de ChromoYYY_PreDefined
public abstract class ChromoInit_TypeReal extends ChromoInit_PreDefined{

//type de l'opérateur
   static public String type = ChromoNodLeafReal.type();
```

```
//constructeur
    public ChromoInit_TypeReal() {
    }

//type de l'opérateur
    public String getType(){
        return type;
    }

//clonnage
    protected Object clone()  throws CloneNotSupportedException{
        return super.clone();
    }
}
```

VI) Classes mères des opérateur (sélecteurs)

Pour tout nouveau chromosome créé, il faut trois sélecteurs de types, une de chaque sorte. Chaque sélecteur de type fourni la liste des opérateur de la sorte qui agissent sur le chromosome. Pour un sélecteur de type de sorte YYY et de type XXX, il faut qu'il soit inscrit dans la liste des sélecteurs de type, l'attribut selectors de la classe SelectorYYY_Root. Il faut donc inscrire les trois sélecteurs dans les trois listes.

Le sélecteur de type prends la forme suivante:

```
//hérite de SelectorYYY_Type (avec YYY= Init et XXX=Real)
public class SelectorInit_TypeReal extends SelectorInit_Type{
```

```
//type du sélecteur
    static String type = ChromoInit_TypeReal.type;

//liste des opérateur d'initialisation sur les réels (YYY= Init et XXX=Real)
    private SelectorInit_Root[] preDef = {new SelectorInitRealUniform()};



//constructeur
    public SelectorInit_TypeReal() {
    }

//retourne le type
    public String getType(){
        return type;
    }

//retourne la liste des opérateurs de la sorte pour ce type de chromosomes.
    protected SelectorInit_Root[] getPredefSel(){
        return preDef;
    }
}
```

VII) Conclusion

Par ces cinq actions nous avons créer ce qui est nécessaire pour un chromosome. Mais pour être utilisable, il lui faut des opérateurs. Le code généré dans EO sera bugé s'il manque un opérateur sur un chromosome. Je vous conseil de voir la section "ajout d'un opérateur". Il est important de donner des opérateurs par défaut aux chromosomes. De cette manière, le génome sera toujours correct. Si tous les opérateurs sont retirés d'un chromosome, l'opérateur par défaut est ajouté.