pilotes externes seront utilisés, car ce sont eux qui permettent de trouver les solutions lors des études de régime non-nominal.

1.7.1 PILOTE DE MOTEUR STIRLING

Dans cette section, nous présenterons le pilote qui a servi à calculer le bilan énergétique du moteur Stirling solaire modélisé chapitre 4 du tome 2. Il s'agit en effet d'un exemple très simple qui permet d'introduire ce type de classe externe sans qu'il soit nécessaire de se plonger dans le détail d'un modèle thermodynamique complexe.

Un moteur Stirling est un moteur d'un type particulier, qui travaille en système fermé, et met en œuvre une compression refroidie et une détente réchauffée, de telle sorte que les indicateurs de performance usuels de Thermoptim ne peuvent pas être directement utilisés : l'énergie payante est la somme de la chaleur fournie à la source chaude (dans cet exemple un concentrateur solaire) et de celle apportée pendant la détente.

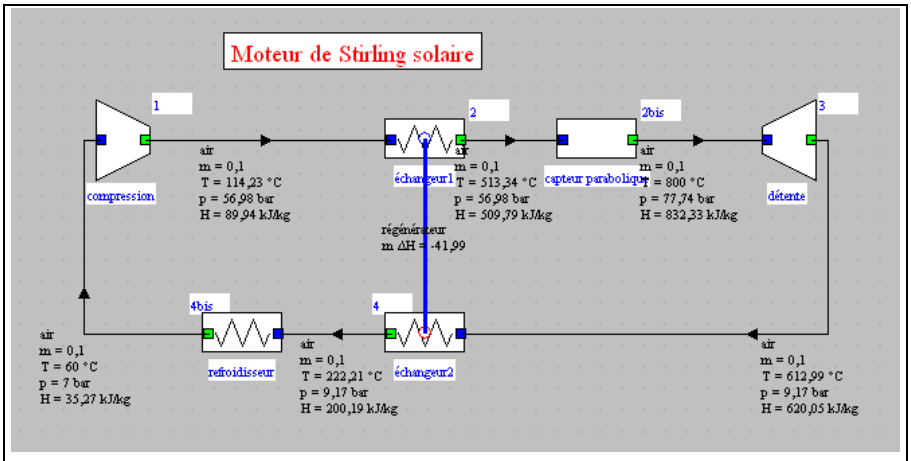

Figure 1.7.1 : Moteur Stirling solaire

L'objectif que nous assignons au pilote est de dresser un tableau récapitulatif des énergies mises en jeu dans le moteur, sous forme de chaleur ou de puissance mécanique, et de calculer le rendement du cycle. La figure 1.7.2 montre le type d'écran que l'on peut imaginer.

1.7.2 CREATION DE LA CLASSE, INTERFACE VISUELLE

Pour créer un pilote externe, il suffit de sous-classer extThopt. ExtPilot.

La réalisation de l'interface visuelle ne pose pas de problème particulier et nous ne la commenterons pas ici.

Le constructeur doit se terminer par un String précisant le code qui désignera le pilote dans la liste de ceux qui sont disponibles : type="stirling";

Il est par ailleurs recommandé de documenter la classe :

```
public String getClassDescription(){
    return "pilot for a simple Stirling motor\n\nauthor : R. Gicquel february 2008";}
```

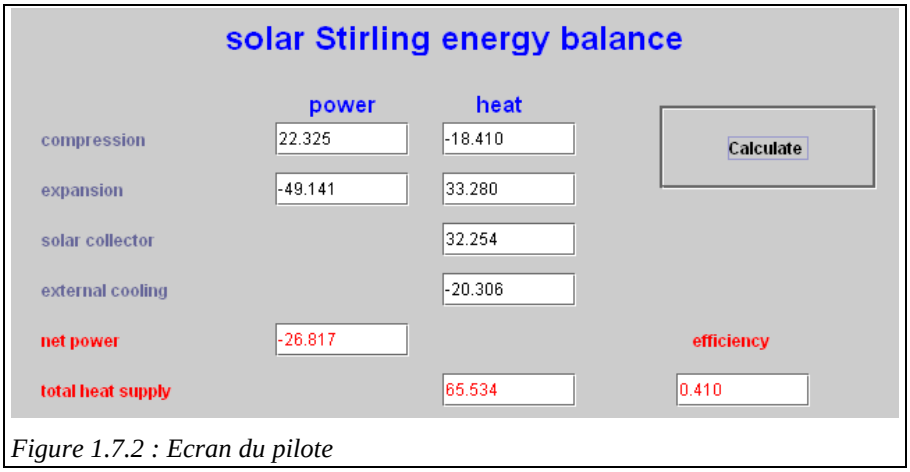

Figure 1.7.2 : Ecran du pilote

1.7.3 RECONNAISSANCE DES NOMS DES COMPOSANTS

Il est possible de reconnaître automatiquement les noms des divers composants constituant le modèle, en les triant par type, ce qui confère au pilote une plus grande généricité que si ces noms sont entrés sous forme de String dans le code. C'est l'objet des méthodes init() et setupProject(), qui utilisent les méthodes permettant d'accéder aux noms des composants de l'éditeur de schémas et à la liste des classes externes disponibles (pour la transfo externe représentant le capteur à concentration).

```
    public void init(){
    isInitialized=true;
    proj=getProjet();
    setupProject();
    //On récupère la liste et le type des composants présents dans l'éditeur de schémas
    String[]listComp=proj.getEditorComponentList();
    composant=new String[listComp.length];
    nomComposant=new String[listComp.length];

    //on en extrait les noms des transfos du noyau dont on a besoin
    //à savoir le compresseur et la chambre de combustion
    for(int i=0;i<listComp.length;i++){
        composant[i]=Util.extr_value(listComp[i], "type");
        nomComposant[i]=Util.extr_value(listComp[i], "name");
        if(composant[i].equals("Compression"))compressorName=nomComposant[i];
        if(composant[i].equals("Expansion"))expansionName=nomComposant[i];
    }
    //test de cohérence (des messages d'erreur plus précis seraient souhaitables)
    if((!expansionName.equals(""))&&(!compressorName.equals(""))&& isBuilt)isBuilt=true;
    if(isBuilt)show();//on n'ouvre le pilote que si sa structure est correcte
    //initialisations pour la simulation (calculs pour 10 rapports de compression)
  }

  void setupProject(){
      //on récupère ici les instances des transfos externes dont on a besoin
      //afin d'accéder à leurs différents paramètres pour les calculs ultérieurs
      Vector vExt=proj.getExternalClassInstances();//Vector contenant les classes externes
      int j=0;
      for(int i=0;i<vExt.size();i++){
          Object[] obj=new Object[6];
          obj=(Object[])vExt.elementAt(i);
          ExtProcess ep=(ExtProcess)obj[1];
          if(ep instanceof SolarConcentrator){
              collector=(SolarConcentrator)ep;
              collectorName=collector.getName();
              j++;
          }
      }
      if(j==1)isBuilt=true;//test de cohérence du pilote par rapport au modèle
}
```

1.7.4 CALCULS EFFECTUES ET AFFICHAGE

Une fois les noms des divers composants identifiés, on accède à leurs propriétés grâce à la méthode getProperties() de Projet, ce qui permet d'obtenir toutes les valeurs dont on a besoin.

```
void bCalculate_actionPerformed(java.awt.event.ActionEvent event){
    if(!isInitialized)init();//la première fois, on initialise, car il faut un constructeur sans argument
                             //pour instancier la classe par le RMI

    String[] args=new String[2];
    args[0]="process";
    args[1]=compressorName;
    Vector vProp=proj.getProperties(args);
    String amont=(String)vProp.elementAt(1);//point amont (non utilisé ici)
    String aval=(String)vProp.elementAt(2);//point aval (non utilisé ici)
    Double f=(Double)vProp.elementAt(4);
    double deltaUcompr=f.doubleValue();//puissance compresseur
    f=(Double)vProp.elementAt(12);
    double Qcompr=f.doubleValue();//chaleur compresseur
    args[1]=expansionName;
    vProp=proj.getProperties(args);
    amont=(String)vProp.elementAt(1);
    aval=(String)vProp.elementAt(2);
    f=(Double)vProp.elementAt(4);
    double deltaUexpan=f.doubleValue();//puissance détente
    f=(Double)vProp.elementAt(12);
    double Qexpan=f.doubleValue();//chaleur détente
    args[0]="process";
    args[1]=collectorName;
    vProp=proj.getProperties(args);
    f=(Double)vProp.elementAt(4);
    double solarHeat=f.doubleValue();//chaleur solaire

    //calcul des performances globales du moteur et affichages
    tauExpan_value.setText(Util.aff_d(deltaUexpan, 3));
    netPower_value.setText(Util.aff_d(deltaUexpan+deltaUcompr, 3));
    tauCompr_value.setText(Util.aff_d(deltaUcompr, 3));
    Q_value.setText(Util.aff_d(Qexpan+solarHeat, 3));
    eta_value.setText(Util.aff_d((-deltaUexpan-deltaUcompr)/(Qexpan+solarHeat), 3));
    expanHeat_value.setText(Util.aff_d(Qexpan, 3));
    comprHeat_value.setText(Util.aff_d(Qcompr, 3));
    solarHeat_value.setText(Util.aff_d(solarHeat, 3));
    extCooling_value.setText(Util.aff_d(-(Qexpan+solarHeat+deltaUexpan+deltaUcompr+Qcompr), 3));
}
```

Comme on le voit, la réalisation d'un pilote externe ne pose pas de problème particulier. Celui-ci est particulièrement simple et se contente d'effectuer les bilans que Thermoptim calculerait de manière erronée compte tenu des spécificités de ce modèle. Il serait par exemple tout à fait possible de le compliquer légèrement, pour qu'il mette à jour le simulateur avant d'effectuer des recalculs du modèle et de dresser le bilan recherché.

1.8 GESTIONNAIRE DE CLASSES EXTERNES

Une fois que des classes externes sont mises au point, il faut les intégrer dans la bibliothèque extUser.zip pour qu'elles puissent être automatiquement reconnues dans la version purement exécutable de Thermoptim.

Un utilitaire de Thermoptim permet d'effectuer cette opération. Appelé "Gestionnaire de classes externes", il présente dans sa partie gauche (figure 1.7.1) l'ensemble des classes disponibles dans le répertoire de la bibliothèque des classes en cours de développement. Ces classes sont rangées par type (corps, transfos externes, mélangeurs externes, diviseurs externes, pilotes). Dans la partie droite, le contenu de l'archive "extUser.zip" est affiché selon le même principe. Il est ainsi possible de facilement comparer les deux séries de classes.

Si vous sélectionnez une classe, dans l'une ou l'autre des listes, son descriptif apparaît dans la fenêtre située en dessous, comme celui de la classe Nozzle sur la figure 1.7.1. Si vous faites une sélection multiple, rien n'est affiché.