Désurchauffeur de vapeur

Un désurchauffeur de vapeur a pour fonction de réaliser la désurchauffe partielle ou totale d'une vapeur surchauffée. Cette opération, qui conduit généralement à une irréversibilité pénalisante, est réalisée seulement dans des circonstances spécifiques comme en sortie de compression mécanique de vapeur, ou pour réguler la production d'une vapeur surchauffée à des fins de cogénération.

Sur le plan technologique, la désurchauffe est réalisée en pulvérisant un flux à l'état liquide dans un écoulement de vapeur.

L'une des difficultés que présente l'utilisation d'un tel composant est que le débit de liquide injecté dépend de l'état thermodynamique de la vapeur, et qu'il est régulé en fonction de l'état désiré pour la vapeur désurchauffée. Il n'est donc pas connu a priori, de telle sorte qu'un simple mélangeur n'est pas bien approprié pour le modéliser dans Thermoptim.

Le composant DeSuperHeater que nous avons réalisé (figure 1) est un mélangeur externe qui reçoit en amont deux transfos, l'une représentant la vapeur et l'autre le liquide. On le paramètre en définissant l'état de sortie désiré pour le mélange (soit par un écart de température par rapport à l'état saturé, soit par sa température), en prenant éventuellement en compte une perte de charge. Le composant détermine le débit de liquide nécessaire, et met à jour récursivement les transfos qui lui sont connectées en amont.

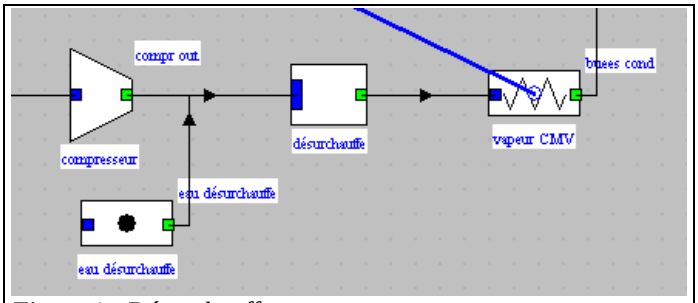

Figure 1 : Désurchauffeur

Interface graphique du désurchauffeur

Une interface graphique possible pour le désurchauffeur est donnée figure 2.

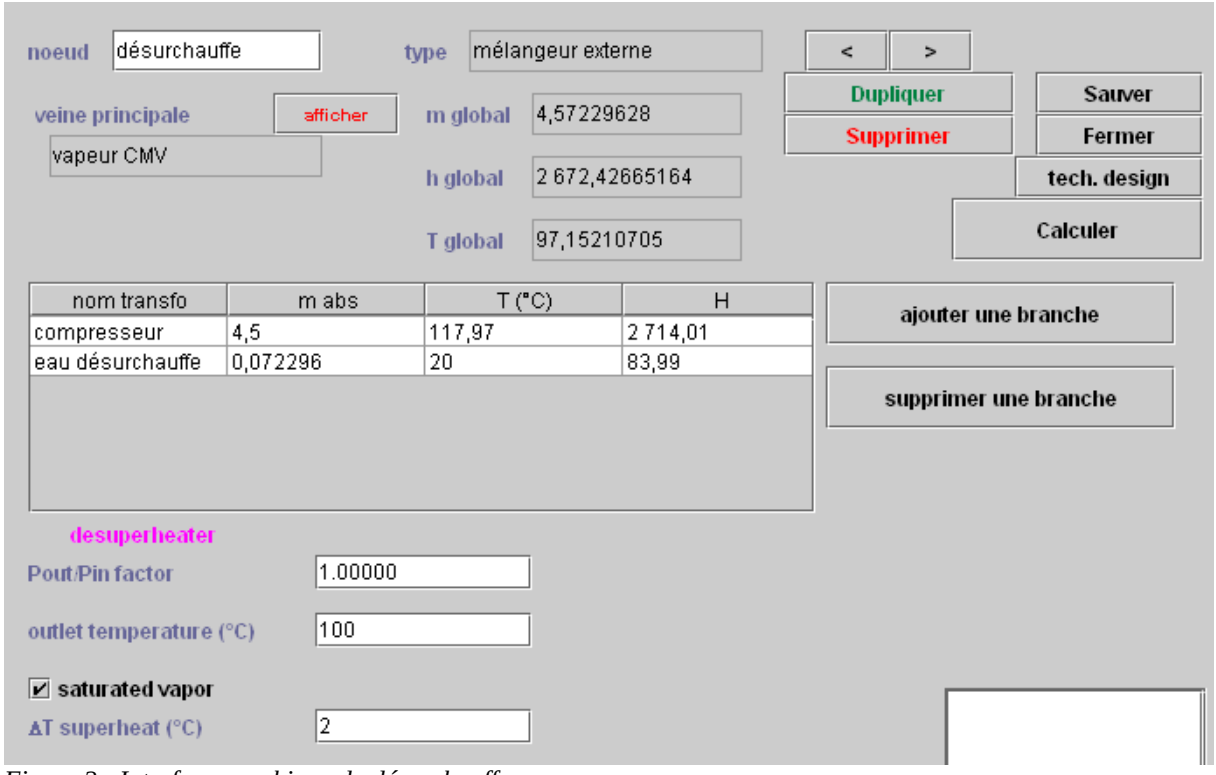

Figure 2 : Interface graphique du désurchauffeur

Le paramétrage retenu ici est le suivant : pas de perte de charge sur la vapeur (pression de sortie égale à la pression d'entrée), et température de sortie égale à celle de saturation plus 2 °C de surchauffe. Le débit d'eau de désurchauffe nécessaire est déterminé à partir des bilans enthalpique et massique, et mis à jour dans la transfo correspondante et celles en amont.

Modèle thermodynamique

Les paramètres du modèle sont d'une part un facteur de perte de charge relative à la pression de la vapeur, et d'autre part la température à atteindre en fin de désurchauffe, définie soit directement par sa valeur, soit par rapport à la saturation.

Les données d'entrée du modèle sont les suivantes (fournies par les autres composants du système) :
- l'état thermodynamique de la vapeur, et notamment sa pression et son enthalpie h_v
- l'état thermodynamique de l'eau de désurchauffe, et notamment son enthalpie h_l
- le débit de la vapeur m_v

Les sorties sont :
- l'état thermodynamique de la vapeur désurchauffée et notamment son enthalpie h_{vd}
- le débit de l'eau de désurchauffe m_l
- le débit de la vapeur désurchauffée m_{vd}

La pression et la température de sortie permettant de déterminer l'enthalpie de la vapeur désurchauffée h_{vd}, les équations du désurchauffeur correspondent à un système d'équations linéaires à deux inconnues, m_l et m_{vd} :

La conservation de la masse fournit : $m_{vd} = m_v + m_l$
La conservation de l'enthalpie donne : $m_v h_v + m_l h_l = m_{vd} h_{vd}$

Ce système se résout sans aucune difficulté et conduit aux résultats de la figure 3.

Implémentation informatique

Interface graphique

L'interface graphique du composant est donnée figure 2. Elle permet de construire le tiers inférieur de l'écran (en anglais sur la figure), le reste étant défini de manière standard dans Thermoptim.

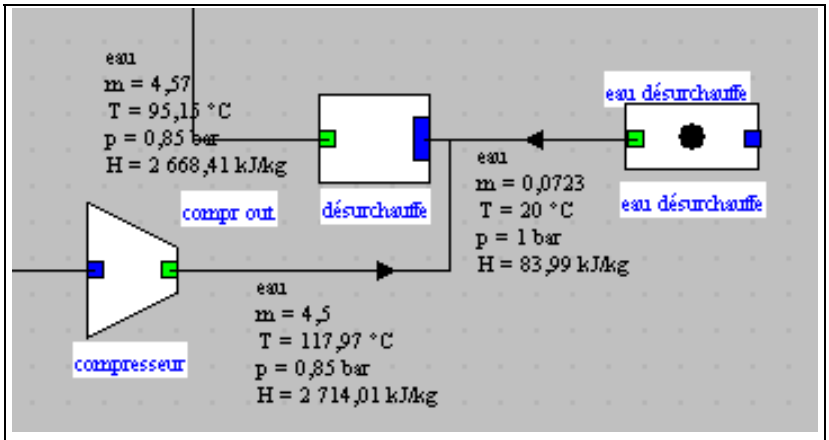

Figure 3 : Mélangeur externe représentant le désurchauffeur, avec ses connexions

Les paramètres correspondent aux quatre lignes ajoutées, les résultats des calculs apparaissant automatiquement dans le reste de l'écran. Les données d'entrée sont fournies par les transfos amont du système dans lequel le composant est inséré : débit de vapeur et états des points amont.

Modèle physique

Avec les notations précédentes, l'équation du modèle est la suivante, le calcul du débit de sortie étant trivial :

$m_l (h_{vd} - h_l) = m_v (h_v - h_{vd})$

Concrètement, l'enchaînement des calculs est le suivant :

1) vérification de la cohérence du nœud ;

2) mise à jour du composant avant calcul par chargement des valeurs des transfos et des points amont ;
3) lecture des paramètres sur l'écran du composant externe ;
4) calcul de l'état du point aval et du débit de liquide ;
5) mise à jour du nœud externe.

Présentation du code

Examinons maintenant les problèmes rencontrés en pratique à chacune de ces étapes.

1) vérification de la cohérence du nœud

Le composant recevant plusieurs transfos en amont, il importe tout d'abord de s'assurer que leur structure est conforme à ce qui est attendu. C'est le rôle des méthodes getMixerStructure() et checkConsistency

```
public void calculateProcess(){

    getMixerStructure();
    checkConsistency();
```

2) mise à jour du composant par chargement des valeurs des transfos et des points amont

Une première difficulté est ici qu'un composant externe n'a pas un accès direct aux variables du simulateur : ces grandeurs sont obtenues par des méthodes particulières génériques, qui construisent des Vector de structure différente selon l'objet désiré.

La marche à suivre n'est pas compliquée mais demande à être respectée :

```
private void checkConsistency(){
    String[] args=new String[2];
    Vector[] vBranch=new Vector[2];
    isBuilt=true;
    liquidProcess="";
    steamProcess="";

    desurchProcess=mainProcess;
    args[0]="process";//type of the element (see method getProperties(String[] args))
    args[1]=desurchProcess;//name of the process (see method getProperties(String[] args))
    Vector vPropMain=proj.getProperties(args);
    Double f=(Double)vPropMain.elementAt(3);
    m_desurch=f.doubleValue();
    String amont=(String)vPropMain.elementAt(1);//gets the upstream point name
    getPointProperties(amont);//direct parsing of point property vector
```

Cette dernière méthode charge l'état du point « amont » dans des variables qui peuvent être utilisées pour initialiser les valeurs dont nous aurons ensuite besoin pour les calculs, comme ici le corps « steam » représentant la vapeur,

```
    desurchPoint=amont;
    T_desurch=Tpoint;
    P_desurch=Ppoint;
/   X_desurch=Xpoint;
    H_desurch=Hpoint;
    String nom=nomCorps;
    args[0]="point";
    args[1]=amont;
    vPropMain=proj.getProperties(args);
    steam=(rg.corps.Corps)vPropMain.elementAt(0);
```

ou bien l'état des points amont et aval :

```
for(int j=0;j<nBranches;j++){
    args[0]="process";//type of the element (see method getProperties(String[] args))
    args[1]=nomBranche[j];//name of the process (see method getProperties(String[] args))
    vBranch[j]=proj.getProperties(args);
    String aval=(String)vBranch[j].elementAt(2);//gets the downstream point name
    getPointProperties(aval);//direct parsing of point property vector
    nom=nomCorps;
    //Check the substance at inlet
    System.out.println(" ligne "+j+" nomBranche[j] "+nomBranche[j]+" aval "+aval+" nomCorps "+nomCorps+" Ppoint "
    if((Xpoint==0)){//liquide
        liquidProcess=nomBranche[j];
        liquidPoint=aval;
        T_liquid=Tpoint;
        P_liquid=Ppoint;
        H_liquid=Hpoint;
        S_liquid=Spoint;
        X_liquid=Xpoint;
        f=(Double)vBranch[j].elementAt(3);
        mLiquid=f.doubleValue();
    }
    if((Xpoint==1)){//vapeur
        steamProcess=nomBranche[j];
        steamPoint=aval;
        P_steam=Ppoint;
        H_steam=Hpoint;
        S_steam=Spoint;
        T_steam=Tpoint;
        X_steam=Xpoint;
        f=(Double)vBranch[j].elementAt(3);
        mSteam=f.doubleValue();
    }
}
```

3) lecture des paramètres sur l'écran du composant externe

Le package extThopt fournit un certain nombre de méthodes simples mais robustes pour convertir en double les String affichés dans les champs JTextField utilisés dans l'interface graphique, et réciproquement pour afficher les double dans ces champs. Ils sont implémentés sous forme de méthodes statiques de la classe extThopt.Util :

```
//lecture du facteur de perte de charge
DeltaP_factor=Util.lit_d(deltaP_factor_value.getText());
//mise à jour de la pression aval
P_desurch=P_steam*DeltaP_factor;
//lecture de la valeur de la surchauffe à réaliser
double deltaT=Util.lit_d(deltaTsat_value.getText());
//lecture de la valeur de la température de désurchauffe
double outletT=Util.lit_d(outletT_value.getText())+273.15;
```

4) calcul de l'état du point aval et du débit de liquide

Selon que l'on a choisi d'imposer la valeur de la surchauffe ou celle de la température de désurchauffe, on calcule l'état de sortie :

```
//mise à jour de la température de désurchauffe
if(JCheckSaturation.isSelected())T_desurch=steam.getSatTemperature(P_desurch,1)+deltaT;
else T_desurch=outletT;
//calcul de l'état de sortie désiré
steam.CalcPropCorps(T_desurch,P_desurch,1) ;
```

On calcule alors le débit de liquide nécessaire, et le débit total qui en résulte :

```
getSubstProperties(nomCorps);
H_desurch=Hsubst;
//calcul du débit de liquide nécessaire
mLiquid=mSteam*(H_steam-H_desurch)/(H_desurch-H_liquid);
//calcul du débit aval
m_desurch=mLiquid+mSteam;
```

5) mise à jour du nœud externe

La méthode de Thermoptim updateMixer () permet de mettre à jour le nœud à partir des valeurs chargées dans un tableau de Vectors correspondant à chaque branche qui lui est connectée.

Une fois cette mise à jour effectuée, les transfos connectées en amont de la transfo correspondant au liquide, si elles existent, voient leur débit actualisé.

```
//Mise à jour du noeud externe
vTransfo= new Vector[nBranches+1];
vPoints= new Vector[nBranches+1];
setupDesurchProcess(m_desurch,T_desurch,P_desurch,1);
setupLiquidProcess(mLiquid,T_liquid,P_liquid,X_liquid);
setupSteamProcess(mSteam,T_steam,P_steam,X_steam);
updateMixer(vTransfo,vPoints,T_desurch,H_desurch);
proj.setUpstreamFlow(liquidProcess, mLiquid);//modifie récursivement les débits en amont
```

Sauvegarde et chargement des paramètres du modèle

Il est possible de sauvegarder dans les fichiers de projet normaux de Thermoptim (et ensuite de relire) les paramétrages des composants externes en utilisant deux méthodes spécifiques :

public String saveCompParameters()

public void readCompParameters(String ligne_data)

La seule contrainte est que l'ensemble du paramétrage du composant externe doit tenir sur une ligne, avec un formatage compatible avec celui qui est utilisé dans le noyau du progiciel : les différents champs de sauvegarde sont séparés par des tabulations.

ExtThopt.Util fournit une méthode générique pour pouvoir associer à la valeur d'un paramètre un code permettant de le repérer :

public static String extr_value(String ligne_data, String search)

La sauvegarde est effectuée sous la forme "paramètre= valeur", et la recherche se fait sous la forme :

valeur=Util.lit_d(Util.extr_value(ligne_data, "paramètre"));

Si le paramétrage du composant est trop complexe pour être sauvegardé de la sorte, rien n'empêche un utilisateur d'utiliser ce mécanisme pour sauvegarder le nom d'un fichier spécifique de paramétrage, et ensuite de relire comme il le désire ce fichier.

```
public void readCompParameters(String ligne_data){
    double P_fact=Util.lit_d(Util.extr_value(ligne_data, "deltaP_factor_value"));
    deltaP_factor_value.setText(Util.aff_d(P_fact,5));
    String val=Util.extr_value(ligne_data, "deltaTsat_value");
    if(val!=null)deltaTsat_value.setText(val);
    val=Util.extr_value(ligne_data, "saturationT");
    if(val!=null)JCheckSaturation.setSelected(Util.lit_b(val));
    val=Util.extr_value(ligne_data, "outletT_value");
    if(val!=null)outletT_value.setText(val);
    String valeur=Util.extr_value(ligne_data, "compRef");
    if(valeur!=null)compRef.setText(valeur);
}

public String saveCompParameters(){
    String h="deltaP_factor_value="+deltaP_factor_value.getText()+tab
    +"deltaTsat_value="+deltaTsat_value.getText()+tab
    +"saturationT="+Util.aff_b(JCheckSaturation.isSelected())+tab
    +"outletT_value="+outletT_value.getText()+tab
    +"compRef="+compRef.getText()+tab
    ;
    return h;
}
```