Cooling coil with condensation

To cool a moist mixture, it is passed through a special heat exchanger called a cooling coil, which can be cooled by ice water or by direct evaporation of a refrigerant (Figure 1). The mixture being in contact with the cold surfaces sees its temperature decrease. Depending on circumstances, there may be condensation or not. If there is no condensation, specific humidity remains constant and cooling can be represented by a horizontal segment oriented to the left in the Carrier chart, and vertically oriented downwards in the Mollier chart. If condensation occurs, which is very often the case, the segment is oriented to the bottom left in both charts (Figure 2).

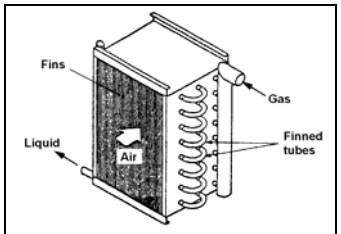

Figure 1 (Extract from Techniques de l'Ingénieur - Génie énergétique)

A perfect theoretical cooling in a cooling coil of infinite surface would lead to cool the moist mixture at the coil temperature saturated state. It is customary to characterize a real process taking this cooling as theoretical reference, introducing the cooling coil effectiveness ε.

ε = **Erreur !**

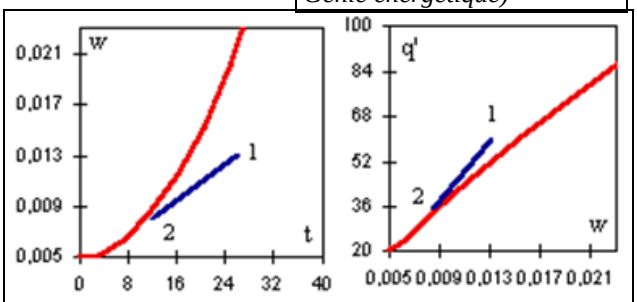

Figure 2

Modeling a cooling coil with condensation in Thermoptim

A cooling coil with condensation acts as a divider receiving the humid air coming in and out of which exit two fluids: water and drier air. It is cooled by a fluid, and the thermal coupling can be represented by a thermocoupler. To simplify writing, we talk about air, but this component can cool and condense any moist mixture.

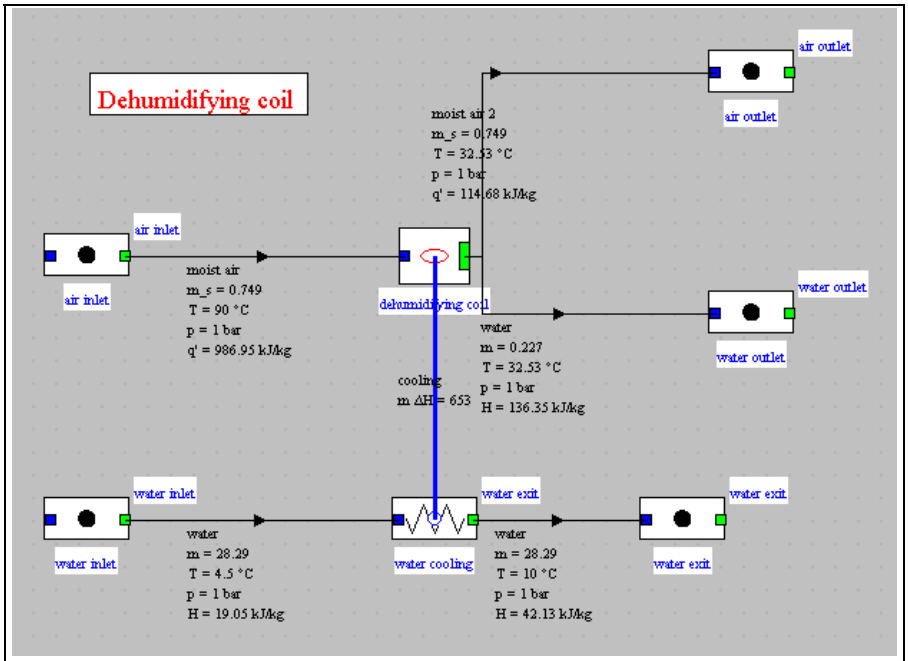

Figure 1.5.1: Diagram of component cooling coil with condensation

The model structure is given Figure 1. Here we have chosen in Thermoptim Global settings screen, the optional display in specific units for moist gas. For air, the flow shown is that of dry gas, which is invariant, and enthalpy h is replaced by the specific enthalpy q'. For water, which is not a moist gas, nothing is changed compared to the usual display.

Model of the cooling coil with condensation

Knowing the cooling coil effectiveness ε and its average surface temperature t_S, the calculation of the air outlet state is performed as follows.

We begin by looking for the saturation conditions $w_{sat}(t_S)$ and $q'(t_S,w_{sat})$.

We then calculate the moist end conditions in view of effectiveness:

$w_2 = \varepsilon w_{sat} + (1 - \varepsilon) w_1$

$q'_2 = \varepsilon q'(t_s, w_{sat}) + (1 - \varepsilon) q'_1$

We then look for t_2 such as $q'_2 = q'(t_2, w_2)$.

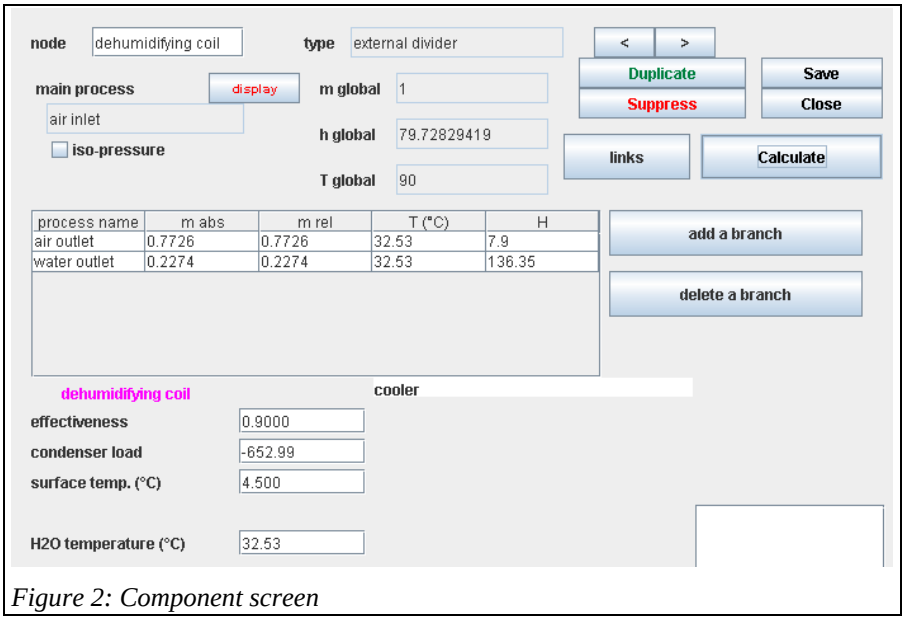

Figure 2: Component screen

A problem may arise when, in the psychrometric chart, the line from point 1 intersects the saturation curve in two points. Indeed, in this case the point calculated from effectiveness may be in the saturated zone. We speak of early condensation. Two possibilities exist: if we set the downstream point specific humidity w_2, the end point is then on the saturation curve for $w_{sat} = w_2$, if we set effectiveness, it is on the saturation curve, for $q'_{sat} = q'_2$.

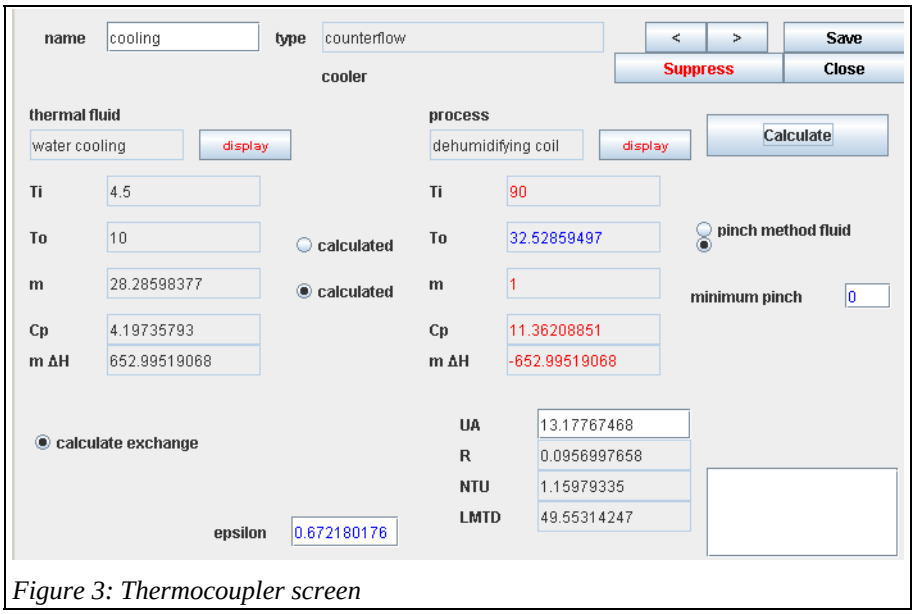

Figure 3: Thermocoupler screen

If we want to make an exact calculation, the coordinates of the second point of intersection of the line from the upstream point with the saturation curve can be obtained by eliminating ε between the two previous equations, with w_2 and q'_2 equal to their saturation values.

One approximate way is to determine temperature $t_{app} = \varepsilon t_{sat} + (1 - \varepsilon) t_1$, and consider it as an estimate of t_2, then to calculate $wsat(t_{app})$. If $w_2 > w_{sat}(t_{app})$, the point is in the saturated zone. This is the test implemented in the class presented here.

Model implementation

The functions for calculating properties of Thermoptim moist gases and points have been made available from external classes. We advise you to refer to the note: "Calculations of wet gas from external classes[1]"for a detailed presentation of the available methods.

The model that we can choose is:

1) the only parameters are firstly the coil effectiveness value ε, read on the screen, and secondly the surface temperature, which we assume to be equal to that of the coolant at the coil entrance;
2) we begin by calculating the inlet moist air properties, and determining the mass flow of dry gas;
3) we calculate the outlet air state at saturation and the final state ("2" in the above equations);
4) the coil outlet temperature is set, and the outlet moist air properties are calculated, which gives the specific and total enthalpy extracted from air;
5) the flow of water transferred by air is determined and the outlet moist air composition is changed;
6) the enthalpy balance provides the thermocoupler load;
7) values downstream of the node are updated.

component name	molar fraction	mass fraction
Ar	0.005845102	0.009295136
O2	0.1363857	0.1737198
N2	0.507225	0.5656051
H2O	0.3505442	0.2513799

Figure 4: Composition of the inlet moist air

The component screen is given in figure 2 and that of the thermocoupler in Figure 3.

component name	molar fraction	mass fraction
Ar	0.008559244	0.01203094
O2	0.1997157	0.2248501
N2	0.7427522	0.7320773
H2O	0.04897291	0.0310416

Figure 1.5.5: Composition of the outlet moist air

The mole fraction of water has been divided by about 7 (Figures 4 and 5).

Study of the external class DehumidifyingCoil

Consistency tests are performed by the node method checkConsistency() to check that the fluids are well connected: in this case, moist air at the inlet and humid air and water at the outlet.

Please refer to Volume 3 of the reference manual for explanations on this point, valid for all external nodes.

The study of the external class DehumidifyingCoil allows one to understand how the model has been implemented. As can be seen, six steps are enough to make the calculations:
1) we begin by calculating the inlet moist air properties thanks to generic method updatepoint(), then initializing the dry air flow-rate and the upstream specific enthalpy:

[1] http://www.thermoptim.org/sections/base-methodologique/extensions-thermoptim/calculs-gaz-humides

```
//Initialisations relatives à l'état du gaz entrant
//Initializations concerning entering gas
getPointProperties(wetGasPoint);
wInlet=Wpoint;
Tinlet=Tpoint;
double test=(inlet_w-wInlet)*(inlet_w-wInlet)/wInlet/wInlet;
if(test>0.0001){
    String msg = "Watch out: the composition of the inlet humid air is not consisten
    JOptionPane.showMessageDialog(de, msg);
    isBuilt=false;
}
double flow_as=wetGasFlow/(1+Wpoint);//débit massique de gaz sec

//calcul des propriétés humides en entrée du composant
//calculation of moist properties at the component inlet
updatepoint(wetGasPoint, false, 0,  //T
    false, 0, false, 0,//P,x
    true, "setW and calculate q'", wInlet);
getPointProperties(wetGasPoint);//direct parsing of point property vector
double qprimeInlet=QPrimepoint;
```

2) we then initialize the surface temperature:

```
//récupération du nom de la transfo couplée au composant par le thermocoupleu
//gives the name of the process coupled to the component bu the thermocoupler
String thcoupl=(String)de.de.getThermoCouplerData("cooler").elementAt(0);
    System.out.println(" thermocoupler : "+thcoupl);
if((thcoupl!=null)&&(!thcoupl.equals("null"))){
    String[] args=new String[2];
    args[0]="heatEx";
    args[1]=thcoupl;
    Vector vProp=proj.getProperties(args);
    String trsf=(String)vProp.elementAt(0);
    System.out.println("hot : "+trsf+" cold : "+(String)vProp.elementAt(1));
    args[0]="process";
    args[1]=trsf;
    vProp=proj.getProperties(args);
    String amont=(String)vProp.elementAt(1);
    getPointProperties(amont);
    //Tpoint contient la valeur de la température d'entrée du fluide de refroidi
    //Tpoint stores the value of thecooling fluid inlet temperature
    ts_value.setText(Util.aff_d(Tpoint-273.15,3));
}
double ts=Util.lit_d(ts_value.getText());
```

3) we then calculate the outlet air saturation conditions:

```
//calcul des conditions de saturation pour le gaz sortant
//calculation of saturation conditions for the exiting gas
updatepoint(dryGasPoint, true, ts+273.15,  //T
    false, 0, false, 0,//P,x
    false, "", 0);

updatepoint(dryGasPoint, false, 0,  //T
        false, 0, false, 0,//P,x
        true, "setEpsi and calculate", 1);
getPointProperties(dryGasPoint);//propriétés à la saturation

double wsat_ts=Wpoint;
double qprime_ts=QPrimepoint;
```

4) we calculate the desired end state:

```
//calcul de l'état final recherché
//calculation of the desired final state
w2=epsi*wsat_ts+(1-epsi)*wInlet;
qprime2=epsi*qprime_ts+(1-epsi)*qprimeInlet;
```

5) we seek to know whether the point so determined may be in the saturated zone:

```
//préparation du test sur la zone saturée
//preparation of the test on the saturated zone
double tapp=epsi*ts+(1-epsi)*(Tinlet-273.15);
updatepoint(dryGasPoint, true, tapp+273.15,  //T
    false, 0, false, 0,//P,x
    false, "", 0);

updatepoint(dryGasPoint, false, 0,  //T
        false, 0, false, 0,//P,x
        true, "setEpsi and calculate", 1);
getPointProperties(dryGasPoint);//propriétés à la saturation / saturation pr
double wsat_tapp=Wpoint;
```

6) we search the corresponding dry bulb temperature with the generic reverse function Util.dicho (which uses f_dicho):

```
double T=0;
if(w2>wsat_tapp){//Si le point est dans la zone saturée / if the point is in the saturated
    T=Util.dicho_T(this, 0, Pinlet, "qprime2sat", ts+273.15, Tinlet, 0.01);
}
else{//sinon / otherwise
    T=Util.dicho_T(this, 0, Pinlet, "qprime2", ts+273.15, Tinlet, 0.01);
}


public double f_dicho(double T, double P, String fonc){

    if (fonc.equals("qprime2")){//point final hors zone saturée / final point outside satu
        double diff;
        updatepoint(dryGasPoint, true, T,  //T
            false, 0, false, 0,//P,x
            false, "", 0);
        updatepoint(dryGasPoint, false, 0,  //T
            false, 0, false, 0,//P,x
            true, "setW and calculate q'", w2);

        getPointProperties(dryGasPoint);//propriétés humides / moist properties
        diff=qprime2-QPrimepoint;
        return diff;
    }
    if (fonc.equals("qprime2sat")){//point final saturé / saturated final point
        double diff;
        updatepoint(dryGasPoint, true, T,  //T
            false, 0, false, 0,//P,x
            false, "", 0);
        updatepoint(dryGasPoint, false, 0,  //T
                false, 0, false, 0,//P,x
                true, "setEpsi and calculate", 1);

        getPointProperties(dryGasPoint);//
        diff=qprime2-QPrimepoint;
        return diff;
    }
    return 0;
}
```

7) we modify the outlet moist air composition, and calculate the thermocoupler load and the mass flow of water exiting:

```
//modification de la composition du gaz
//modification of the gas composition
updatepoint(dryGasPoint, false, 0,  //T
        false, 0, false, 0,//P,x
        true, "modHum", 0);

//charge du condenseur / condenser load Qprime_inlet
double DeltaQprime=flow_as*qprime2-Qprime_inlet;//enthalpie to
double condLoad=DeltaQprime;

//eau mise en jeu / water put into play
 double H2Oflow=(inlet_w-Wpoint)*flow_as;
```

8) the node is updated using generic methods.

```
//affichages à l'écran / screen displays
condenserLoad_value.setText(Util.aff_d(condLoad,2));
Tout_value.setText(Util.aff_d(T-273.15,2));
epsi_value.setText(Util.aff_d(epsi,4));

//mise à jour des thermocoupleurs, comme pinchFluid, DTmin=10
//thermocoupler update, as pinchFluid, DTmin=10
updateThermoCoupler("cooler", Tinlet, T, condLoad, wetGasFlow,true,10);

//mise à jour du noeud en utilisant les méthodes génériques
//node update by generic methods
vTransfo= new Vector[nBranches+1];
vPoints= new Vector[nBranches+1];
setupVector(H2OProcess, H2OPoint, 0, H2Oflow, T, Pinlet, 0);
setupVector(dryGasProcess, dryGasPoint, 1, wetGasFlow-H2Oflow, T, Pinlet, 0);
setupVector(wetGasProcess, wetGasPoint, 2, wetGasFlow, Tinlet, Pinlet, 1);
updateDivider(vTransfo,vPoints,Tinlet,Hinlet*wetGasFlow);

de.updateProcess(setEnergyTypes(wetGasProcess,0,0,0));
```