Etude d'un cycle AZEP pour la capture du CO2 par oxy-combustion

Description du cycle

Une membrane MIEC (Mixed ionic-electronic conducting) est une membrane en céramique perméable à l'oxygène.

A haute température (supérieure à 700 °C), la membrane céramique (figure 1) est un conducteur mixte ionique et électronique, qui laisse passer simultanément des ions O2- et des électrons, l'oxygène étant adsorbé en surface.

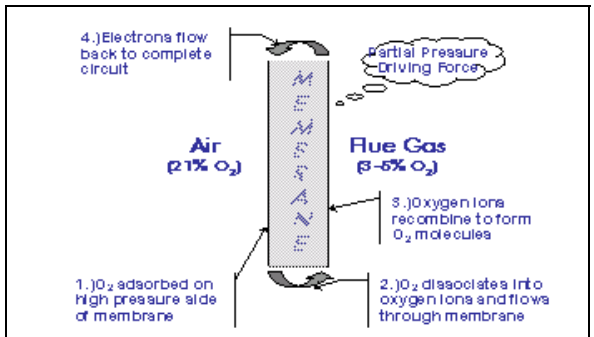

Figure 1 : Schéma de la membrane

Modèle du composant membrane

La membrane présente la particularité de mettre en jeu deux flux séparés : l'air qui s'appauvrit en oxygène et les fumées sortant de la chambre de combustion qui s'enrichissent en oxygène, qui échangent de la matière par l'intermédiaire d'une interface. Elle se comporte donc comme un quadripôle recevant deux fluides en entrée, et dont en sortent deux autres.

Pour la représenter dans Thermoptim, on forme ce quadripôle en associant un mélangeur en entrée (classe MIEC_Inlet) et un diviseur en sortie (classe MIEC), les deux étant reliés par une transfo-point qui joue un rôle purement passif. Les intitulés des classes sont "MIEC inlet" et "MIEC".

Pour que le modèle soit bien cohérent, on synchronise les calculs effectués par les deux nœuds. Plus précisément le diviseur de sortie prend le contrôle du mélangeur, dont le rôle se limite à effectuer une mise à jour des variables de couplage associées aux flux d'entrée.

La structure du modèle est donnée figure 2. La membrane est représentée par les 3 composants "MIEC inlet", "MIEC" et "MIEC outlet". Le mélangeur d'entrée MIEC inlet reçoit d'une part l'air comprimé réchauffé par le comburant avant entrée dans la chambre de combustion, et d'autre part une fraction des fumées recirculées et refroidies. En sortie du diviseur externe, on retrouve d'une part l'air appauvri , qui est réchauffé par les fumées sortant de la chambre de combustion avant détente dans la turbine HTT, et d'autre part le comburant formé par les fumées enrichies en oxygène, qui sont refroidies avant d'entrer dans la chambre de combustion.

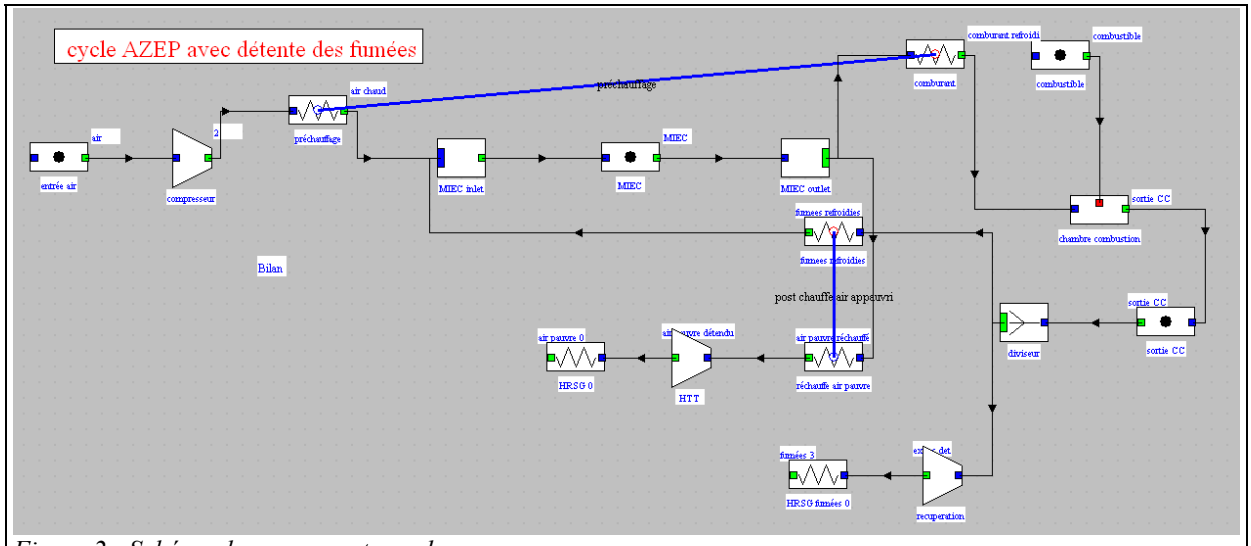

Figure 2 : Schéma du composant membrane

Le modèle de membrane que nous développons ici est celui retenu par le LENI de l'EPFL[1]. Le débit molaire surfacique d'oxygène traversant la membrane y est donné par :

$$j_{O2} = j_{O2,o} \cdot e^{-\frac{E_A}{RT}} \left[p^n{}_{O2,feedside} - p^n{}_{O2,permeateside} \right]$$

Il est proportionnel à un débit de référence, à exp(-Ea/RT), Ea étant l'énergie d'activation et T la température de la membrane, et à la différence de pression partielle d'oxygène entre les deux faces de la membrane. Etant donné que du côté chambre de combustion, on peut considérer que la pression partielle d'oxygène est nulle, cette équation se simplifie encore.

Le modèle que l'on peut retenir est alors le suivant :

- la surface de la membrane, sa température et sa conductance surfacique sont des paramètres lus à l'écran
- on commence par calculer grâce à l'équation ci-dessus le débit molaire d'oxygène qui traverse la membrane
- on détermine ainsi la composition de l'air appauvri puis celle du comburant auquel se mélange l'oxygène, ainsi que leurs débits
- on estime la puissance thermique transférée entre les deux faces de la membrane, ce qui fournit l'enthalpie du comburant et celle de l'air appauvri

Dans ce modèle, on ne connaît pas la pression dans la chambre de combustion, qui est une enceinte distincte de celle de l'air. Pour éviter d'endommager la membrane, la différence de pression entre ses deux faces doit rester faible.

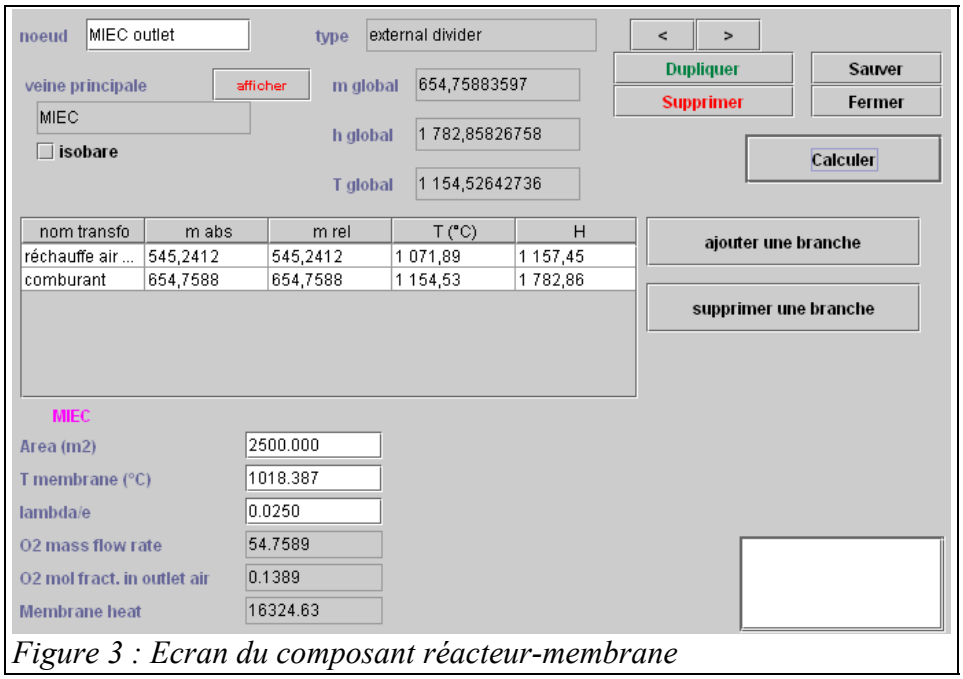

Figure 3 : Ecran du composant réacteur-membrane

Sur le plan technologique, c'est le combustible qui pressurise ce circuit, par exemple grâce à un éjecteur. Dans notre modèle, nous ne représenterons pas ce dispositif, nous contentant d'imposer une pression légèrement plus élevée que celle du circuit de l'air.

Dans la chambre de combustion, nous supposerons que la réaction est stœchiométrique et complète.

Etude de la classe externe MIEC

Pour assurer la cohérence du modèle (éviter que l'on connecte le mélangeur d'entrée à un diviseur de sortie inadéquat), chacun des deux nœuds essaie d'instancier l'autre en recherchant sa classe parmi les composants externes du projet, et vérifie que tous deux sont bien connectés à la même transfo de liaison. Si l'opération échoue, un message avertit l'utilisateur que la construction est incorrecte. Cette vérification est effectuée par les méthodes setupOutlet() et setupInlet().

De surcroît, des tests de cohérence de chaque nœud sont effectués par la méthode checkConsistency() pour vérifier que les fluides connectés sont les bons : dans ce cas, un gaz humide et de l'eau en entrée et en sortie. On

[1] A. Maestro, Thermo-economic design of hydrogen production systems using oxygen, LENI-EPFL, juin 2005

se reportera au tome 3 du manuel de référence pour les explications sur ce point, valables pour tous les nœuds externes.

L'étude de la classe externe MIEC permet de voir comment le modèle a été implémenté. Cinq étapes suffisent pour cela :

1) on on commence par calculer le débit molaire d'oxygène qui traverse la membrane :

```
//Calcul des pressions partielles de chaque côté de la membrane
//on considère que le transfert d'oxygène se fait à contre-courant, de telle sorte que le
//gradient de pression partielle d'oxygène est à peu près constant

//analyse de la composition de l'air
Vector airComp=mciei.airSubstance.getGasComposition();
double fractO2=Util.molarComp(airComp,"O2");//fraction molaire de O2
double airMolFlow=mciei.airFlow/mciei.airM;//débit molaire d'air
double PO2=mciei.Pair*fractO2;//pression partielle d'oxygène dans l'air entrant

//analyse de la composition du comburant
Vector comburComp=comburGasSubstance.getGasComposition();
double fractO2Combur=Util.molarComp(comburComp,"O2");
double P2=Pcombur*fractO2Combur;//pression partielle d'oxygène dans le comburant sortant

double Tmemb=Util.lit_d(Tmemb_value.getText())+273.15;//température de membrane lue à l'écran

double O2DepletedAirMolFlow=airMolFlow*fractO2-getO2TransferredMolFlow(Tmemb,PO2,P2);//débit d'oxygène dans le comburant

double getO2TransferredMolFlow(double T, double P, double P2){//retourne des kmol/s
    double area=Util.lit_d(Area_value.getText());
    return area*2.2e-3*Math.exp(73.2/8.314/T)*(Math.pow(P, 0.25)-Math.pow(P2, 0.25));
}
```

2) on calcule ensuite la composition et le débit de l'air appauvri

```
double fractN2=Util.molarComp(airComp,"N2");//fraction molaire de N2
double fractAr=Util.molarComp(airComp,"Ar");//fraction molaire de Ar
double totalDepletedAirMolFlow=O2DepletedAirMolFlow+(fractN2+fractAr)*airMolFlow;
fractN2=(fractN2*airMolFlow)/totalDepletedAirMolFlow;
fractAr=fractAr*airMolFlow/totalDepletedAirMolFlow;
fractO2=O2DepletedAirMolFlow/totalDepletedAirMolFlow;
//détermination de la composition de l'air appauvri
Vector vComp=new Vector();
Double[]fracMol= new Double[3],molarMass= new Double[3];
String[] nom_gaz_comp = new String[3];
vComp.addElement(new Integer(3));
nom_gaz_comp[0]="N2";
nom_gaz_comp[1]="O2";
nom_gaz_comp[2]="Ar";
fracMol[0]=new Double(fractN2);
fracMol[1]=new Double(fractO2);
fracMol[2]=new Double(fractAr);
molarMass[0]=new Double(28.02);
molarMass[1]=new Double(32);
molarMass[2]=new Double(39.95);
vComp.addElement(nom_gaz_comp);
vComp.addElement(fracMol);
vComp.addElement(fracMol);
vComp.addElement(molarMass);
O2outFract_value.setText(Util.aff_d(fractO2,4));

depletedAirSubstance.updateGasComposition(vComp);//modification de la composition de l'air appauvri

Vector vSubst=depletedAirSubstance.getSubstProperties();
Double z=(Double)vSubst.elementAt(7);
double airM=z.doubleValue();//masse molaire de l'air appauvri
double depletedAirFlow=totalDepletedAirMolFlow*airM;//débit massique d'air appauvri
O2Flow_value.setText(Util.aff_d(mciei.airFlow-depletedAirFlow,4));//débit massique d'oxygène transféré
```

3) on calcule ensuite la composition et le débit du comburant :

```
//analyse de la composition du comburant
comburComp=mciei.comburSubstance.getGasComposition();
fractO2=Util.molarComp(comburComp,"O2");//fraction molaire de O2
vComp=new Vector();

//détermination de la composition du comburant enrichi en oxygène
if(fractO2==0){
    double comburMolFlow=mciei.comburFlow/mciei.comburM;
    Integer i=(Integer)comburComp.elementAt(0);
    int nComp=i.intValue();
    if(nComp>=0){
        double O2MolFlow=airMolFlow-totalDepletedAirMolFlow;//débit d'oxygène dans le comburant
        totalCombMolFlow=O2MolFlow+comburMolFlow;
        String[] newComp= new String[nComp+1];
        Double[]newFractmol= new Double[nComp+1];
        Double[]newMolarMass= new Double[nComp+1];
        String[] Comp= new String[nComp];
        double[]fractmol= new double[nComp],fractmass= new double[nComp];
        fracMol= new Double[nComp];
        Comp=(String[])comburComp.elementAt(1);
        fracMol=(Double[])comburComp.elementAt(2);
        molarMass=(Double[])comburComp.elementAt(4);
        for(int j=0;j<nComp;j++){
            newComp[j]=Comp[j];
            Double f=(Double)fracMol[j];
            fractmol[j]=f.doubleValue();
            double fract=fractmol[j]*comburMolFlow/totalCombMolFlow;
            newFractmol[j]=new Double(fract);
            newMolarMass[j]=molarMass[j];
        }
        newComp[nComp]="O2";
        newFractmol[nComp]=new Double(O2MolFlow/totalCombMolFlow);
        newMolarMass[nComp]=new Double(32.);
        vComp.addElement(new Integer(nComp+1));
        vComp.addElement(newComp);
        vComp.addElement(newFractmol);
        vComp.addElement(newFractmol);
        vComp.addElement(newMolarMass);
    }
}


comburGasSubstance.updateGasComposition(vComp);//modification de la composition du comburant
vSubst=comburGasSubstance.getSubstProperties();
z=(Double)vSubst.elementAt(7);
double comburM=z.doubleValue();//masse molaire du combustible humide

comburGasFlow=totalCombMolFlow*comburM;
```

4) on détermine la température des gaz en sortie du composant

```
//charge thermique de la membrane
double Q=Util.lit_d(Area_value.getText())*Util.lit_d(lambda_value.getText())*(mciei.Tcombur-mciei.Tair);
Q_value.setText(Util.aff_d(Q,2));
double HoutCO2=(mciei.Hcombur*mciei.comburFlow-Q)/comburGasFlow;
TcomburGas=comburGasSubstance.getT_from_hP(HoutCO2,1.);
double Houtlet=Hsubst;

double HairPauvre=(mciei.Hair*mciei.airFlow+Q)/depletedAirFlow;
double TairPauvre=depletedAirSubstance.getT_from_hP(HairPauvre,1.);
Tmemb_value.setText(Util.aff_d((TairPauvre+mciei.Tair)/2.-273.15,3));
```

5) le nœud est mis à jour en utilisant les méthodes génériques décrites dans le manuel de référence

```
//mise à jour du noeud aval en utilisant les méthodes génériques
vTransfo= new Vector[nBranches+1];
vPoints= new Vector[nBranches+1];
setupVector(depletedAirProcess, depletedAirPoint, 0, depletedAirFlow, TairPauvre, mciei.Pair, 1);
setupVector(comburProcess, comburPoint, 1, comburGasFlow, TcomburGas, Pcombur, 1);
setupVector(mainProcess, linkPoint, 2, comburGasFlow, TcomburGas, Pcombur, 1);
updateDivider(vTransfo,vPoints,TcomburGas,Houtlet);


//mise à jour du noeud amont en utilisant les méthodes génériques
vTransfo= new Vector[mciei.nBranches+1];
vPoints= new Vector[mciei.nBranches+1];
setupVector(mciei.recycleProcess, mciei.recyclePoint, 0, mciei.comburFlow, mciei.Tcombur, mciei.Pcombur, 1);
setupVector(mciei.airProcess, mciei.airProcess, 1, mciei.airFlow, mciei.Tair, mciei.Pair, 1);
setupVector(mainProcess, linkPoint, 2, comburGasFlow, TcomburGas, Pcombur, 1);
mciei.updateMixer(vTransfo,vPoints,TcomburGas,Houtlet);
```