AZEP oxyfuel cycle for CO_2 capture

Description of the cycle

A MIEC (Mixed ionic-electronic conducting) membrane is a ceramic membrane permeable to oxygen.

At high temperatures (above 700 °C), the ceramic membrane (Figure 1) is a mixed ionic and electronic conductor, which passes simultaneously O_2-ions and electrons, oxygen being adsorbed on the surface.

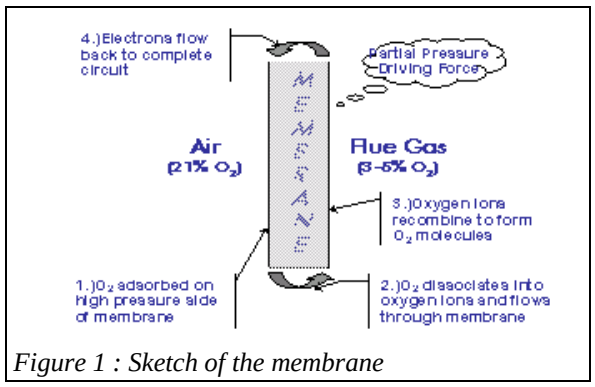

Figure 1 : Sketch of the membrane

Model of the membrane component

The membrane involves two separate streams which exchange matter through an interface: the oxygen depleted air and fumes out of the combustion chamber that are enriched in oxygen. It behaves like a quadrupole receiving two fluids in input, out of which come the other two.

To represent it in Thermoptim, the quadrupole is formed by combining the inlet Mixer (class MIEC_Inlet) and an outlet divider (class MIEC), the two being connected by a process-point playing a passive role. The classes references are "MIEC Inlet" and "MIEC".

For the model to be consistent, it synchronizes the calculations made by the two nodes. More specifically, the outlet divider takes control of the mixer, whose role is to perform an update of the coupling variables associated with the input stream.

The model structure is given in Figure 2. The membrane is represented by the three components "MIEC Inlet", "MIEC" and "MIEC outlet." The inlet mixer MIEC Inlet receives the compressed air heated by the combustion before entering the combustion chamber, and a fraction of the recirculated and cooled flue gas. At the output of the external divider, there is firstly the oxygen-depleted air, which is heated by the flue gases leaving the combustion chamber prior to expansion in the HTT turbine, and secondly the oxidizer formed by oxygen enriched fumes, which are cooled before entering the combustion chamber.

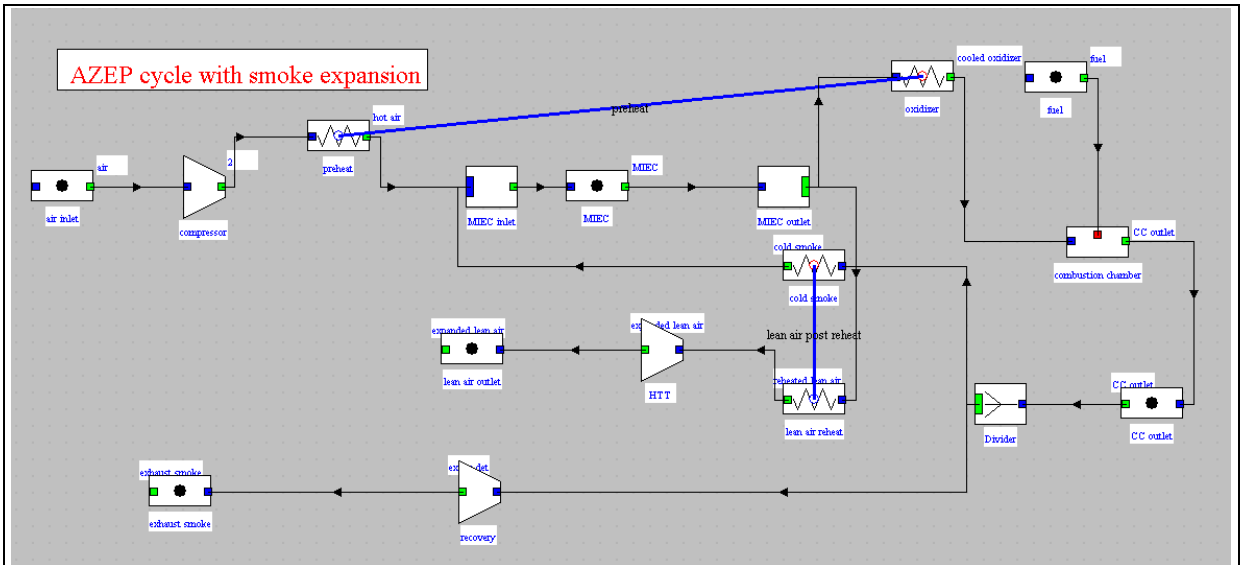

Figure 2: Diagram of the membrane component coupled to the combustion chamber

The membrane model we develop here is that used by the EPFL LENI. The surface molar flow of oxygen through the membrane is given by:

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

It is proportional to a reference flow, to exp(-Ea/RT), Ea being the activation energy and T the temperature of the membrane, and the difference in oxygen partial pressure between the two sides of the membrane. Since we can consider that the partial pressure of oxygen is zero combustion chamber side, this equation can be simplified further.

The model that can be retained is the following:

1) the membrane surface, its temperature and surface conductance parameters are read on the screen;
2) we first calculate using the equation above the molar flow of oxygen through the membrane;
3) we thus determine the composition of the depleted air then that of the oxidizer with which oxygen is mixed and their flow-rates
4) we estimate the thermal power transferred between the two sides of the membrane, which provides the enthalpy of combustion and that of the depleted air;

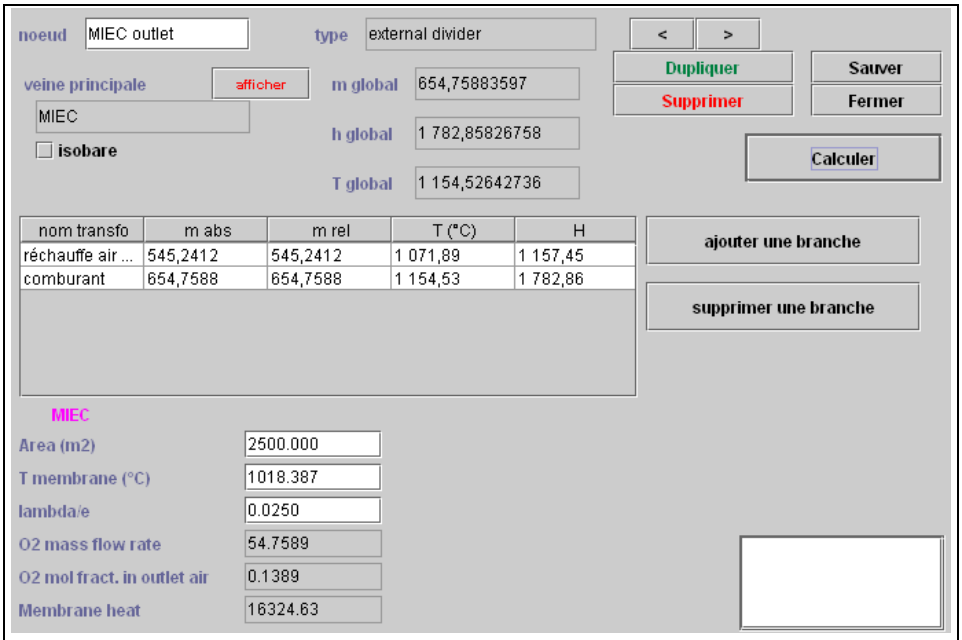

Figure 3 : Ecran du composant réacteur-membrane

In this model, available we do not know the pressure in the combustion chamber, a chamber separate from the air. To prevent damage to the membrane, the pressure difference between both sides must remain low.

Technologically, the fuel, here assumed to be natural gas at 70 bar, pressurizes the circuit, for example through an ejector. In our model, we do not represent this device, we simply set a slightly higher pressure than the air circuit.

In the combustion chamber, we assume that the reaction is stoichiometric and complete.

Study of the external class MIEC

To ensure consistency of the model (to avoid that the inlet mixer be connected to an inadequate outlet divider), each of the two nodes tries to instantiate the other seeking its class from among the external components of the project, and checks that both are well connected to the same connection process. If the operation fails, a message warns the user that the construction is incorrect. This check is performed by the methods setupOutlet() and setupInlet().

In addition, consistency tests are carried out for each node by the method checkConsistency() to check that the connected fluids are the right ones: in this case, a moist gas and water in and out. 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 MIEC shows how the model has been implemented. Five steps are used for this:

1) we begin by calculating the molar flow of oxygen through the 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
//calculation of partial pressures of oxygen on both sides of the membrane
//we assume the O2 transfer is counterflow, so that the O2 partial pressure gradient
//is almost constant

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

//analyse de la composition du comburant / oxidizer composition analysis
Vector comburComp=comburGasSubstance.getGasComposition();
double fractO2Combur=Util.molarComp(comburComp,"O2");
double P2=Pcombur*fractO2Combur;//pression partielle d'oxygène dans le comburant sortant
                                //O2 partial pressure in the exiting oxidizer
double Tmemb=Util.lit_d(Tmemb_value.getText())+273.15;//température de membrane lue à l'écran
                                                      //membrane temperature read on the screen

//débit d'oxygène dans le comburant / O2 flow rate in the oxidizer
double O2DepletedAirMolFlow=airMolFlow*fractO2-getO2TransferredMolFlow(Tmemb,PO2,P2);
```

```
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) we then calculate the composition and the flow of air depleted

```
double fractN2=Util.molarComp(airComp,"N2");//fraction molaire de N2 / N2 molar fraction
double fractAr=Util.molarComp(airComp,"Ar");//fraction molaire de Ar / Ar molar fraction
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 /depleted air composition
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);//composition de l'air / air composition

Vector vSubst=depletedAirSubstance.getSubstProperties();
Double z=(Double)vSubst.elementAt(7);
double airM=z.doubleValue();//masse molaire de l'air / air molar mass
double depletedAirFlow=totalDepletedAirMolFlow*airM;//débit massique d'air / air flow rate
O2Flow_value.setText(Util.aff_d(mciei.airFlow-depletedAirFlow,4));//débit massique d'oxygène transféré
                                                                  //O2 mass flow rate transferred
```

3) we then calculate the composition and flow of the oxidizer:

```
//détermination de la composition du comburant enrichi en oxygène
//composition of the O2 enriched oxidizer
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'O2 dans le comburant / O2 flow in oxidizer
        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();//change of the oxidizer composition
 z=(Double)vSubst.elementAt(7);
 double comburM=z.doubleValue();//masse molaire du combustible humide / moist fuel molar mass

 comburGasFlow=totalCombMolFlow*comburM;
```

4) we determine the temperature of the gases leaving the component

```
    //charge thermique de la membrane / membrane thermal load
    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) the node is updated using the generic methods described in the reference manual

```
//mise à jour du noeud aval en utilisant les méthodes génériques
//update of the downstream node by the generic methods
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
//update of the upstream node by the generic methods
vTransfo= new Vector[mciei.nBranches+1];
vPoints= new Vector[mciei.nBranches+1];
setupVector(mciei.recycleProcess, mciei.recyclePoint, 0, mciei.comburFlow, mciei.Tcombur, mciei.Pcombur,
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);
```