CLC reactor model

The set composed by the two reactors has the distinction of bringing into play two separate streams: air and gases that combine in the combustion reaction, and exchange matter and energy through the metal oxide. It behaves like a quadrupole receiving two input fluids, and come out with two others.

To represent it in Thermoptim, the quadrupole is formed by combining an inlet mixer (CLCReactorInlet class) and an outlet divider (class CLCReactor), the two being connected by a process-point playing a passive role. The class types are "CLC reactor inlet" and "CLC reactor".

For the model to be consistent, we synchronize the calculations made by the two nodes. Specifically the outlet divider takes control of the mixer, whose role is limited to conduct an update of the coupling variables associated with the input stream.

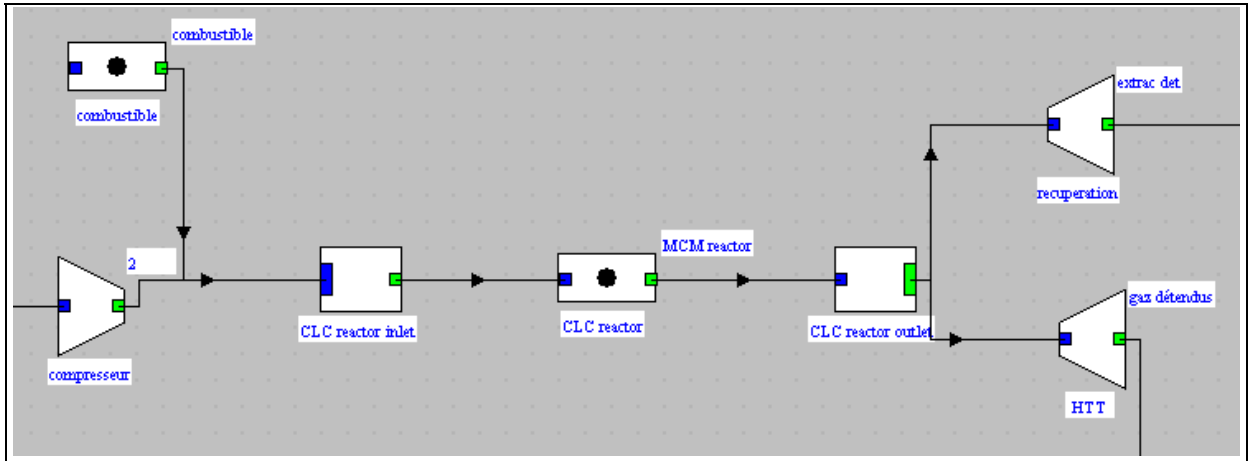

Figure 1: Diagram of reactor component

The model structure is given in Figure 1. The reactors are represented by three components "CLC reactor inlet", "CLC reactor" and "CLC reactor outlet". The inlet mixer CLC reactor inlet receives on the one hand the compressed air, and secondly the fuel. At the external divider outlet, there are on the one hand depleted air, which is expanded in the HTT turbine, and on the other hand the exhaust gases.

The model we built is purely of the depleted air are known, it is the two compartments of the

In the combustion chamber, we As the temperature reached the the calculation of gas properties in of the LHV, which provides a balance of the reactor, by writing incoming enthalpies plus 80% of exit temperature of flue gases,

Clearly this model is inadequate, flow of metal oxide. It may be that cycle values close to those

However, we lack precise data on

The model we choose is the

1) temperature and the molar
2) we determine the composition
3) we calculate the combustion temperature and the fuel flow rate
4) the end of combustion
5) we adjust accordingly the inlet processes

Study of the external

To ensure consistency of the outlet divider), each of the two the project external components, connection process-point. If the construction is incorrect. This is

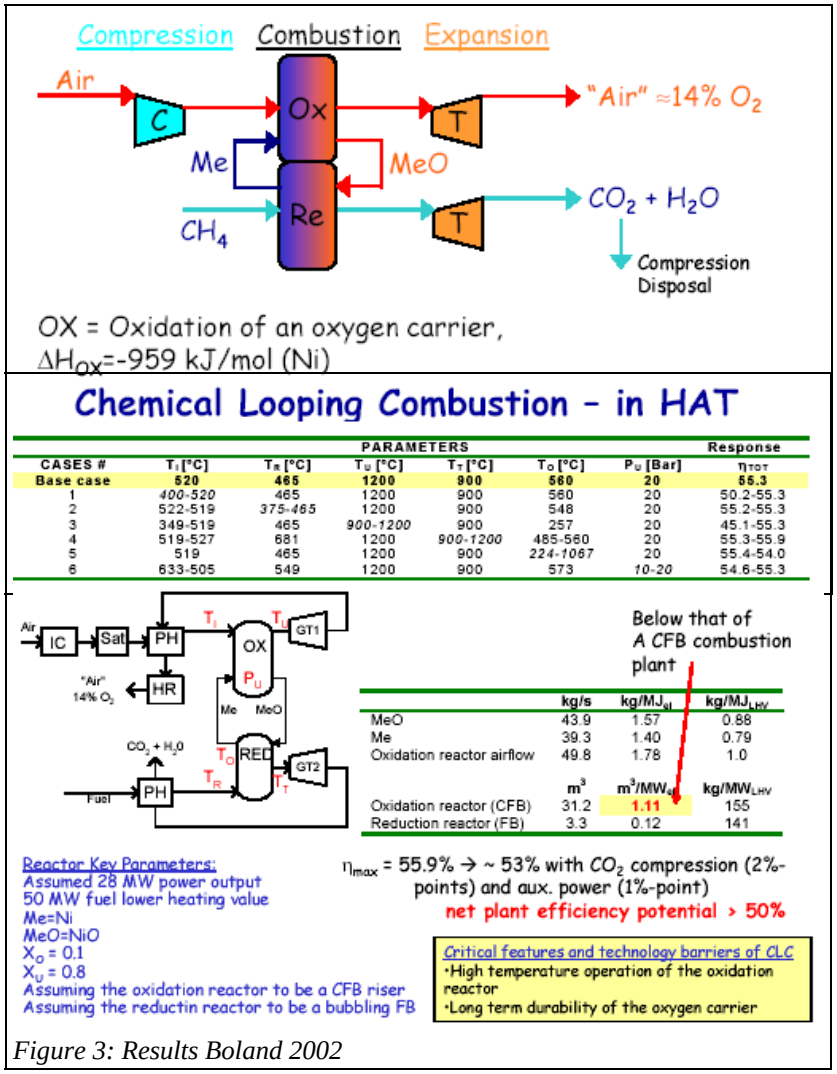

Figure 3: Results Boland 2002

global. Considering that the temperature and composition possible to determine the rate of oxygen transfer between reactor.

assume that the reaction is stoichiometric and complete. end of combustion exceeds the limit of 2600 K used for Thermoptim, we set a fictitious thermal load equal to 80% fictive temperature. Then we perform a comprehensive that the sum of the outgoing enthalpies is equal to that of LHV not taken into account, which allows us to find the which includes only CO2 and H2O.

especially because it does not account for losses due to the supplemented later. By varying the parameters, we get for announced by Bolland[1] (Figures 2 and 3).

the cycle in order to validate our model precisely.

following:

fraction of depleted air parameters are read on the screen;
of the depleted air;
reaction, which determines the end of combustion

temperature is recalculated as explained above;
fuel flow and the setting values of the component outlet

class CLCReactor

model (avoid that the inlet mixer connects to an inadequate nodes tries to instantiate the other seeking its class among and verifies that both are well connected to the same operation fails, a message warns the user that the checked by the methods setupOutlet () and setupInlet ().

[1] O. Bolland, Options for oxy-fuels & pre-combustion decarbonisation cycles, Presentation for Alstom Cross Segment CO2 Mitigation Group, October 11th 2002

Moreover, consistency tests are performed on each node by the method checkConsistency () to check that the fluids are rightly connected: in this case, a wet 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 CLCReactor shows how the model was implemented. Six steps allow us to perform the calculations:
1) we first calculate the molar flow of oxygen into play:

```
//analyse de la composition de l'air
Vector airComp=mcmi.airSubstance.getGasComposition();
double fractO2=Util.molarComp(airComp,"O2");//fraction molaire de O2
double airMolFlow=mcmi.airFlow/mcmi.airM;//débit molaire d'air
double PO2=mcmi.Pair*fractO2;//pression partielle d'oxygène dans l'air
double Tmemb=Util.lit_d(Tmemb_value.getText())+273.15;//température de membrane lue à l'écran
double fractO2out=Util.lit_d(O2outFract_value.getText());
double totalDepletedAirMolFlow=airMolFlow*(1.-fractO2)/(1.-fractO2out);

double O2DepletedAirMolFlow=airMolFlow-totalDepletedAirMolFlow;//débit d'oxygène pour la combustion
```

2) the composition and the flow of depleted air are then calculated

```
//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(mcmi.airFlow-depletedAirFlow,4));//débit massique d'oxygène transféré
```

3) we then calculate the partial combustion reaction by following the steps above:

```
//on impose une charge artificielle pour éviter d'avoir une température trop élevée
double heatLoad=-PCI*mcmi.fuelM*0.8;//ramené à 1 kmol de combustible
System.out.println("heatLoad : "+heatLoad);

//Initialisation de la combustion
//on fait l'hypothèse d'une combustion stoechiométrique complète
//Sinon, il faut prévoir une conversion shift pour transformer le CO
Vector vSettings=new Vector();
vSettings.addElement(NewComburSubstance);
vSettings.addElement(mcmi.fuelSubstance);
vSettings.addElement(new Double(totalCombFlow));//oxidizer flow rate
vSettings.addElement(new Double(mcmi.fuelFlow));//fuel flow rate
vSettings.addElement(new Double(1.));//lambda (combustion stoechiométrique)
vSettings.addElement(new Double(Tmemb));//Tcomb
vSettings.addElement(new Double(0.));//dissociation rate (inutilisé ici)
vSettings.addElement("false");//dissociation boolean (pas de dissociation)
vSettings.addElement(new Double(273.15));//Tfigeage (inutilisé ici)
vSettings.addElement(new Double(0.));//a (inutilisé ici)
vSettings.addElement(new Double(0.));//hfOcarb (inutilisé ici)
vSettings.addElement("false");//comb CHa
vSettings.addElement(new Double(Hcarb));//hc
vSettings.addElement(new Double(Hcarb));//uc
vSettings.addElement("false");//setFuelFlow
vSettings.addElement(new Double(heatLoad));//heat load
vSettings.addElement("true");//open system
vSettings.addElement("true");//IcalcDirect //si true on calcule Tcomb à partir de lambda
vSettings.addElement(burntGasSubstance);
vSettings.addElement(new Double(mcmi.Tair));//T amont
vSettings.addElement(new Double(mcmi.Hair));//H amont
vSettings.addElement(new Double(1.));//rendement chambre

mcmi.calcExternalCombustion(vSettings);//lancement des calculs de combustion

Vector results=mcmi.getExternalCombustionResults();//Vector des résultats
```

4) finally, we determine the temperature of exhaust gases by the overall reactor enthalpy balance and by feeding back 80% of LHV artificially removed

```
depletedAirSubstance.CalcPropCorps(Tmemb,Pamont, 1); //recalcul de l'état de sortie (fournit Hsubst)
getSubstProperties(depletedAirSubstanceName);
double HdepletedAir=Hsubst;

burntGasSubstance.CalcPropCorps(T_burntGas,Pamont, 1); //recalcul de l'état de sortie (fournit Hsubst)
getSubstProperties(burntGasSubstanceName);

double HcombArtificiel=Hsubst;

double HoutCO2=HcombArtificiel+0.8*PCI*fuelFlow/burntGasFlow-(depletedAirFlow)/burntGasFlow*(HdepletedAir-mcmi.Hair);
T_burntGas=burntGasSubstance.getT_from_hP(HoutCO2,1.);
double Houtlet=Hsubst;
```

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
vTransfo= new Vector[nBranches+1];
vPoints= new Vector[nBranches+1];
setupVector(depletedAirProcess, depletedAirPoint, 0, depletedAirFlow, Tmemb, mcmi.Pair, 1);
setupVector(flueGasProcess, flueGasPoint, 1, burntGasFlow, T_burntGas, gasP, 1);
setupVector(mainProcess, linkPoint, 2, burntGasFlow, T_burntGas, gasP, 1);
updateDivider(vTransfo,vPoints,T_burntGas,Houtlet);

de.updateProcess(setEnergyTypes(mainProcess,0,PCI*fuelFlow,0));//DeltaH énergie payante


//mise à jour du noeud amont en utilisant les méthodes génériques
vTransfo= new Vector[mcmi.nBranches+1];
vPoints= new Vector[mcmi.nBranches+1];
setupVector(mcmi.combustProcess, mcmi.combustPoint, 0, fuelFlow, mcmi.Tfuel, mcmi.Pfuel, 1);
setupVector(mcmi.airProcess, mcmi.airProcess, 1, mcmi.airFlow, mcmi.Tair, mcmi.Pair, 1);
setupVector(mainProcess, linkPoint, 2, burntGasFlow, T_burntGas, gasP, 1);
mcmi.updateMixer(vTransfo,vPoints,T_burntGas,Houtlet);
```