Biomass combustion model

This note presents briefly the external class BiomassCombustion. This is a simplified model that can simulate different types of biomass combustion, and in which it is possible to vary with some flexibility the composition and fuel moisture and combustion conditions. BiomassCombustion class can be used to simulate both a boiler and a downdraft gasifier.
The main thermodynamic parameters that influence biomass combustion are:

- first, of course, fuel composition;
- second, moisture, which determines the enthalpy required for drying, plays on the gas composition, and finally influences CO_2 dissociation;
- finally, quenching temperature and CO_2 dissociation rate.

To separate as much as possible the influence of the first two parameters, composition is that of dry fuel, and moisture is taken into account by adding water. Given that overall moisture is not necessarily the one that governs the thermodynamic equilibrium, as part of the steam may not react, for reasons of kinetics or geometry, we introduced an additional parameter, equal to the fraction of water involved in combustion. Physically, this means that a fraction of water is not dissociated: it must be vaporized but its influence on the gas composition is the same as if it were inert.

Fuel definition and combustion progress

Fuel definition is made in Thermoptim as follows (figure 1) :

- dry gas composition, excluding species not included in Thermoptim core, is estimated, for example in a spreadsheet, and entered in the form of a compound gas (process-point "dry fuel");
- species not considered by Thermoptim core are considered separately: they are entered in the component screen, and are subject to pre-combustion;
- the dry gas is mixed with water (substance “water”) to form the moist fuel (process-point "humidity");
- final combustion is calculated by core Thermoptim functions, which are emulated from the external class (external mixer "biomass combustion "), taking into account pre-combustion and the enthalpy that would have been required to evaporate the water.

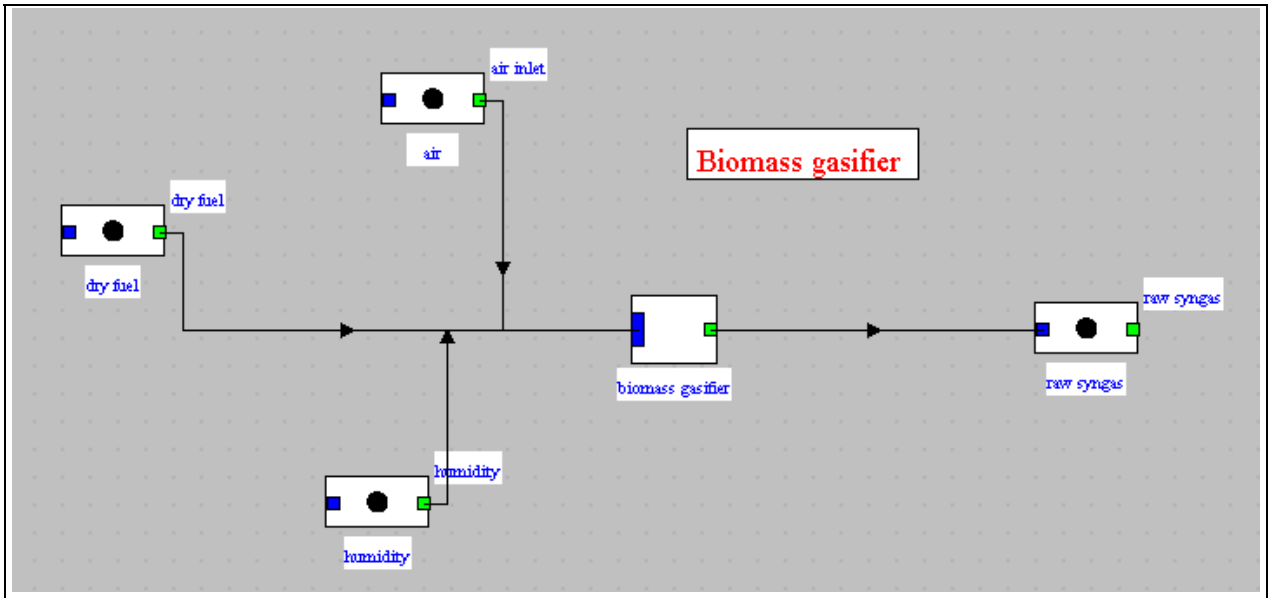

Figure 1: Diagram of the combustion structure

The oxidizer (usually air) is itself defined in process-point "air inlet". Flows involved are also specified in each of the three processes-points entering the external mixer. Their respective ratios allow one to play on fuel moisture and excess air.

Warning: the external mixer "biomass combustion" must be **connected to processes-points** and not to other process types, otherwise point states are not updated correctly. In addition, in the case where the fuel and oxidizer are at different temperatures, the temperature selected for the process-point bringing water must be selected on the basis of an adequate physical reasoning in order to reflect the distribution of moisture between these two inlet flows.

Example of lack of air combustion (gasifier)

Biomass gasification is partial oxidation of an organic resource, mainly composed of cellulose ($C_6H_{10}O_5$) to produce synthesis gas. Biomass being generally very wet, oxidation takes place in four stages:

- fuel drying;
- pyrolysis or carbonization (in the absence of oxygen), producing tars and carbon;
- combustion of carbon and oxygen, exothermic reaction drying fuel among others;
- reduction of CO_2, H_2 and water by carbon and tar.

By performing lack of air combustion, we can model a downdraft gasifier (Figure 1.4.1) using class BiomassCombustion. We assumed here that only 50% of fuel moisture takes part in combustion, the rest not participating.

Figure 2 shows the component screen allowing one to model biomass combustion. In the case presented, it is assumed that fuel comprises 0.634% ammonia and 10.5% carbon referred to dry mass, that the quenching temperature equals 900 °C and that 50% of the moisture is involved in combustion.

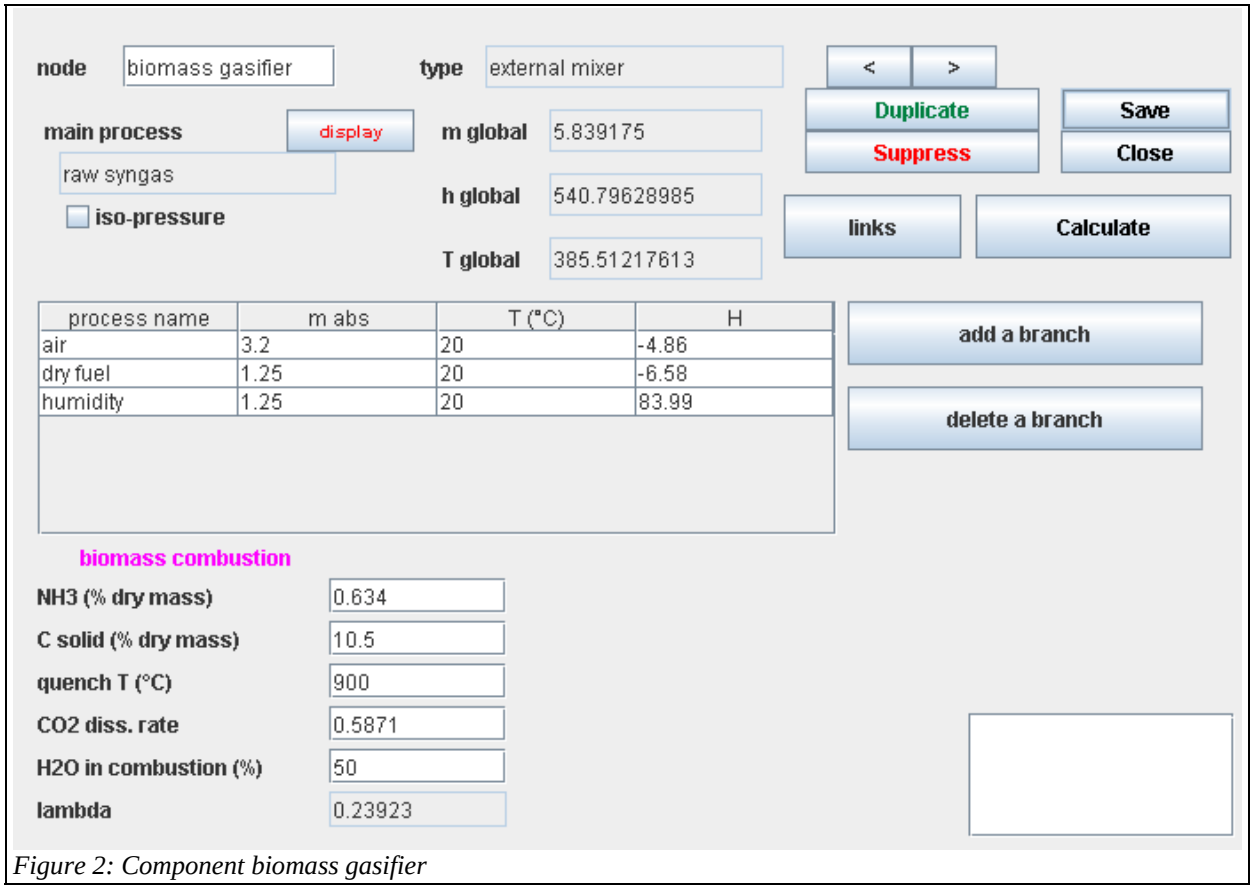

Figure 2: Component biomass gasifier

As we have said, the input flow-rates of fuel, water and oxidizer obviously play a fundamental role, as their ratios directly influence the synthesis gas composition (Figure 3).

In this example, fuel moisture is 50% by mass, the flows of the two processes "dry oil" and "humidity" being equal. Air flow being less than that which would have ensured a stoichiometric combustion (about 7.8 kg/s), combustion occurs in lack of air and the CO_2 dissociation rate is recalculated.

component name	molar fraction	mass fraction
CO2	0.09095625	0.1780174
H2O	0.2800306	0.2243506
N2	0.3332404	0.4151491
CO	0.1293294	0.1611006
H2	0.1626134	0.01457812
Ar	0.003829836	0.006804206

Figure 3: Composition of raw synthesis gas (LHV: 3,4 MJ/kg)

The synthesis gas temperature depends a lot on the heat of reaction, itself a function of oxygen available: if there is very little, it is mainly CO that is produced with few CO_2.

The raw synthesis gas, high humidity, can be washed and partially dried, using class WaterQuench presented in the Thermoptim-UNIT portal model library.

Access to combustion calculations from external classes

So that external classes can access combustion calculations available in Thermoptim, various modifications were made to the package. In particular, a class called ExternalCombustion, which inherits directly from Combustion, was added. Accessible from MixerExterne, it allows for combustion calculations in the following manner:

- we start by building Vector vSettings, which includes the three ideal gases involved (oxidizer, fuel, exhaust gases) and all the necessary settings to specify the required calculations;
- the method public void calcExternalCombustion(Vector vSettings) initiates the combustion of MixerExterne, then performs calculations required;
- results are then retrieved by the method public Vector getExternalCombustionResults() of MixerExterne, which returns a Vector comprising the flue gas composition, end of combustion temperature, lambda, energies involved etc.

From external classes (only in external mixers, a combustion chamber being structurally similar to a mixer), access is through the methods of the same name in ExtMixer, with calls like:

```
calcExternalCombustion(vSettings);
Vector results=getExternalCombustionResults();
```

Setting the combustion is in every way the same as that required for combustion carried out in the Thermoptim core, with the proviso that it is possible to set a heat load to be taken into account in calculating the end of combustion temperature.

Presentation of the external class

The class is an external mixer (Figure 1), which receives as input on the one hand water, representing fuel moisture, and on the other hand two gases: the oxidizer, identified by the fact that it contains both oxygen and nitrogen, and fuel, which does not contain both these gases. The main outlet vein is the syngas. Consistency checks are performed on these bases.

The model is somewhat complex given its nature. We limit ourselves here to explain the overall progress of calculations, leaving the reader who wishes further information to refer to the explanations provided in Thermoptim model library.

The calculation procedure is as follows:
1) calculation of ammonia and solid carbon combustion (to simplify things, we have just changed the composition of the oxidizer, making the implicit assumption that it was dry air; we could also have depleted oxygen in the oxidizer and increased fuel H_2O and CO_2 mole fractions);

```
//initialisation des débits molaires de NH3 et de carbone
double NH3Flow=fuelFlow*Util.lit_d(NH3_value.getText())/100;//mass flow
double CFlow=fuelFlow*Util.lit_d(C_solid_value.getText())/100;//mass flow
double additionalFlow=NH3Flow+CFlow;//pour correction débit comburant
NH3Flow=NH3Flow/17.034;//molar flow
CFlow=CFlow/12;//molar flow

//analyse de la composition du comburant
Vector comburComp=comburSubstance.getGasComposition();
double fractO2=Util.molarComp(comburComp,"O2");//fraction molaire de O2
double comburMolFlow=comburFlow/comburM;
double O2MolFlow=comburMolFlow*fractO2-0.75*NH3Flow-CFlow;//débit d'oxygène restant dans le comburant
if(O2MolFlow<0){
    String msg = "Not enough oxygen in oxidizer for NH3 and C";
    JOptionPane.showMessageDialog(me, msg);
}
double fractN2=Util.molarComp(comburComp,"N2");//fraction molaire de N2
double fractAr=Util.molarComp(comburComp,"Ar");//fraction molaire de Ar
double totalCombFlow=O2MolFlow+(fractN2+fractAr)*comburMolFlow
+NH3Flow*(0.5// 1/2 N2
          +1.5// 3/2 H2O
          )+CFlow;// CO2
fractN2=(fractN2*comburMolFlow+0.5*NH3Flow)/totalCombFlow;
fractAr=fractAr*comburMolFlow/totalCombFlow;
double fractH2O=(1.5*NH3Flow)/totalCombFlow;
double fractCO2=CFlow/totalCombFlow;
fractO2=O2MolFlow/totalCombFlow;
```

```
//détermination de la composition du comburant modifié
Vector vComp=new Vector();
Double[]fracMol= new Double[5],molarMass= new Double[5];
String[] nom_gaz_comp = new String[5];
vComp.addElement(new Integer(5));
nom_gaz_comp[0]="N2";
nom_gaz_comp[1]="O2";
nom_gaz_comp[2]="Ar";
nom_gaz_comp[3]="H2O";
nom_gaz_comp[4]="CO2";
fracMol[0]=new Double(fractN2);
fracMol[1]=new Double(fractO2);
fracMol[2]=new Double(fractAr);
fracMol[3]=new Double(fractH2O);
fracMol[4]=new Double(fractCO2);
molarMass[0]=new Double(28.02);
molarMass[1]=new Double(32);
molarMass[2]=new Double(39.95);
molarMass[3]=new Double(18.02);
molarMass[4]=new Double(44.01);
vComp.addElement(nom_gaz_comp);
vComp.addElement(fracMol);
vComp.addElement(fracMol);
vComp.addElement(molarMass);

NewComburSubstance=Util.createIdealGas("comburant");//création d'un nouveau gaz composé
NewComburSubstance.updateGasComposition(vComp);//modification de la composition du comburant
```

2) humidification of fuel and calculation of the heat load to be taken into account in the rest of the combustion (referred to 1 kmol of fuel);

```
double waterFlow=massWaterFlow*Util.lit_d(H2O_comb_value.getText())/100.;//partie de l'eau prise en compte dans la combusti

double fractH2OFuel=1./(1.+fuelFlow/fuelM*18.01528/waterFlow);//fraction molaire de H2O

Vector fuelComp=fuelSubstance.getGasComposition();
double inlet_w=18.01528*fractH2OFuel/fuelM/(1-fractH2OFuel);//(M_H2O)(x_H2O)/(M_gs)/(x_gs)

//modification de la composition du combustible
NewFuelSubstance=Util.createIdealGas("wetFuel");//création d'un nouveau gaz composé

//dans un premier temps on charge la composition du gaz sec
NewFuelSubstance.updateGasComposition(fuelComp);

//puis on impose l'humidité
vComp=new Vector();
vComp.addElement(new Integer(-1));
vComp.addElement("setGasHum");
vComp.addElement(Util.aff_d(inlet_w));
NewFuelSubstance.updateGasComposition(vComp);

Vector vSubst=NewFuelSubstance.getSubstProperties();
Double z=(Double)vSubst.elementAt(7);
fuelM=z.doubleValue();//masse molaire du combustible humide

//la charge est égale à la chaleur libérée par la combustion de NH3 et C, moins la chaleur de vaporisation de l'eau
double Q=46190*NH3Flow+393520*CFlow-2444*massWaterFlow;//charge totale (kJ)
double heatLoad=Q*fuelM/(fuelFlow+waterFlow);//ramené à 1 kmol de combustible
```

3) initialization and calculation of combustion, then recovery of results;

```
//Initialisation de la combustion
Vector vSettings=new Vector();
vSettings.addElement(NewComburSubstance);
vSettings.addElement(NewFuelSubstance);
vSettings.addElement(new Double(comburFlow+additionalFlow));//oxidizer flow rate
vSettings.addElement(new Double(fuelFlow+waterFlow));//fuel flow rate
vSettings.addElement(new Double(3.));//lambda (inutilisé ici)
vSettings.addElement(new Double(1000.));//Tcomb (inutilisé ici)
vSettings.addElement(new Double(Util.lit_d(k_CO_value.getText())));//dissociation rate
vSettings.addElement("true");//dissociation boolean
vSettings.addElement(new Double(Util.lit_d(Tfig_value.getText())+273.15));//Tfigeage
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("true");//setFuelFlow
vSettings.addElement(new Double(heatLoad));//heat load
vSettings.addElement("true");//open system
vSettings.addElement("true");//IcalcDirect //si true on calcule Tad à partir de lambda
vSettings.addElement(synGasSubstance);
vSettings.addElement(new Double(Tcombur));//T amont
vSettings.addElement(new Double(Hcombur));//H amont
vSettings.addElement(new Double(1.));//rendement chambre

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

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

4) update of the outlet gas humidity;

```
//mise à jour de la composition du gaz pour tenir compte de l'eau non prise en compte dans la combustion
double waterExcessFlow=massWaterFlow*(1.-Util.lit_d(H2O_comb_value.getText())/100.);//partie de l'eau non prise en compte

synGasFlow=comburFlow+fuelFlow+additionalFlow+waterFlow;
double fractExcessH2O=1./(1.+synGasFlow/Msubst*18.01528/waterExcessFlow);//fraction molaire de H2O

fuelComp=synGasSubstance.getGasComposition();
Integer taille=(Integer)fuelComp.elementAt(0);
int nbComp=taille.intValue();

vComp=new Vector();
vComp.addElement(new Integer(nbComp));
fracMol= new Double[nbComp];molarMass= new Double[nbComp];
nom_gaz_comp = new String[nbComp];

for(int i=0;i<nbComp;i++){
    String[]noms=(String[])(fuelComp.elementAt(1));
    nom_gaz_comp[i]=noms[i];
    if(!nom_gaz_comp[i].equals("H2O")){
        Double[]frac=(Double[])(fuelComp.elementAt(2));
        double x=frac[i].doubleValue();
        fracMol[i]=new Double(x*(1-fractExcessH2O));
        Double[]M=(Double[])(fuelComp.elementAt(4));
        molarMass[i]=M[i];
    }
    else{
        Double[]frac=(Double[])(fuelComp.elementAt(2));
        double x=frac[i].doubleValue();
        fracMol[i]=new Double(x*(1-fractExcessH2O)+fractExcessH2O);
        Double[]M=(Double[])(fuelComp.elementAt(4));
        molarMass[i]=M[i];
    }
}
vComp.addElement(nom_gaz_comp);
vComp.addElement(fracMol);
vComp.addElement(fracMol);
vComp.addElement(molarMass);

synGasSubstance.updateGasComposition(vComp);//modification de la composition du gaz en sortie
```

5) update of the mixer (note: in the class, the Houtlet calculation is actually done before step 4 so that the change in composition of synGasSubstance has no influence on its value).

```
Double y=(Double)results.elementAt(2);
double lambda=y.doubleValue();
System.out.println("lambda : "+lambda);
lambda_value.setText(Util.aff_d(lambda,5));

synGasSubstance=(Corps)results.elementAt(0);
y=(Double)results.elementAt(1);
T_syngas=y.doubleValue();
y=(Double)results.elementAt(5);
k_CO_value.setText(Util.aff_d(y.doubleValue(),4));

y=(Double)results.elementAt(6);
double DeltaH=y.doubleValue();
System.out.println("DeltaH : "+DeltaH);

y=(Double)results.elementAt(4);
double eta_comb=y.doubleValue();
System.out.println("eta_comb : "+eta_comb);

synGasSubstance.CalcPropCorps(T_syngas,Pamont, 1); //recalcul de l'état de sortie (fournit Hsubst)
getSubstProperties(synGasSubstanceName);
double Houtlet=Hsubst;

//recherche de la température de sortie (on suppose que l'enthalpie de l'eau est égale à celle du carburant)
double Hout=Houtlet*synGasFlow+Hcarb*waterExcessFlow;
DeltaH=Hout-Hcarb*(massWaterFlow+fuelFlow)-Hcombur*comburFlow;
synGasFlow=synGasFlow+waterExcessFlow;
Hout=Hout/synGasFlow;
double Tout=synGasSubstance.getT_from_hP(Hout,Pamont);

//mise à jour du noeud en utilisant les méthodes génériques
vTransfo= new Vector[nBranches+1];
vPoints= new Vector[nBranches+1];
setupVector(fuelProcess, fuelPoint, 0, fuelFlow, Tcombur, Pamont, 1);
setupVector(comburProcess, comburPoint, 1, comburFlow, Tcombur, Pamont, 1);
setupVector(waterProcess, waterPoint, 2, massWaterFlow, Tcombur, Pamont, 1);
setupVector(synGasProcess, synGasPoint, 3, synGasFlow, Tout, Pamont, 1);
updateMixer(vTransfo,vPoints,Tout,Hout);

me.updateProcess(setEnergyTypes(synGasProcess,0,DeltaH,0));//DeltaH énergie payante
```