A detailed description of reactingTwoPhaseEulerFoam focussing on the link between heat and mass transfer at the interface

Darren Cappelli

Solid State Pharmaceutical Cluster,
University College Dublin,
Dublin, Ireland

2018-11-28

reactingTwoPhaseEulerFoam Solver

- A two-phase Eulerian model is employed to describe the system.
- Each phase is its own continuum and the phases are considered inter-penetrating.
- The solver can handle multi component phases and heat and mass transfer between phases.
- Each phase will have its own discrete set of equations which will require solving.

Equations

- Momentum Conservation Equation - 1 per phase

$$\frac{\partial \alpha_\varphi \bar{\mathbf{U}}_\varphi}{\partial t} + \nabla \cdot (\alpha_\varphi \bar{\mathbf{U}}_\varphi \bar{\mathbf{U}}_\varphi) + \nabla \cdot (\alpha_\varphi \bar{\mathbf{R}}_\varphi^{\;eff}) = -\frac{\alpha_\varphi}{\rho_\varphi}\nabla \bar{p} + \alpha_\varphi \mathbf{g} + \frac{\bar{\mathbf{M}}_\varphi}{\rho_\varphi}$$

  Where $\bar{\mathbf{M}}_\varphi$ is the averaged inter-phase momentum transfer term will also contain a momentum term due to mass transfer, $\sum \dot{\mathbf{m}}_{i\varphi} \bar{\mathbf{U}}_\mathbf{r}$.

- Bounded Phase Continuity Equation - 1 Equation

$$\frac{\partial \alpha_\varphi}{\partial t} + \nabla \cdot (\bar{\mathbf{U}} \alpha_\varphi) + \nabla \cdot (\bar{\mathbf{U}}_\mathbf{r} \alpha_\varphi (1 - \alpha_\varphi)) = 0$$

  Where $\bar{\mathbf{U}} = \alpha_\varphi \bar{\mathbf{U}}_\varphi + \alpha_{\varphi-1} \bar{\mathbf{U}}_{\varphi-\mathbf{1}}$ and $\bar{\mathbf{U}}_\mathbf{r} = \bar{\mathbf{U}}_\varphi - \bar{\mathbf{U}}_{\varphi-\mathbf{1}}$.

- Energy Conservation Equation - 1 per phase

$$\frac{\partial \alpha_\varphi \rho_\varphi (\mathbf{he}_\varphi + \mathbf{K}_\varphi)}{\partial t} + \nabla \cdot (\alpha_\varphi \rho_\varphi \mathbf{U}_\varphi (\mathbf{he}_\varphi + \mathbf{K}_\varphi)) - \nabla \cdot (\alpha_\varphi \bar{\alpha}_\varphi^{\;eff} \nabla \mathbf{he}_\varphi) =$$

$$\alpha_\varphi \frac{\partial \bar{p}}{\partial t} + \rho_\varphi \mathbf{g} \cdot \mathbf{U}_\varphi + \alpha_\varphi \dot{\mathbf{Q}}_R + \dot{\mathbf{Q}}_{KD} + \dot{\mathbf{Q}}_{TD}$$

Species Conservation Equations

Multicomponent phases will require species conservation equations.

$$\frac{\partial \alpha_\varphi C_i}{\partial t} + \nabla \cdot (\alpha_\varphi \bar{\mathbf{U}}_\varphi C_i) - \nabla \alpha_\varphi D_{i\varphi} \nabla (C_i) = \dot{\mathbf{R}} + \frac{dm_i}{dt}$$

Openfoam manipulates the equation so it is valid for compressible flow.

- Converts $C_i(kg/m^3)$ to $Y_i(kg/kg)$ $\quad Y_i = \frac{C_i}{\rho_\varphi}$
- $D_{i\varphi}$ is computed by the Schmidt Number $\quad D_{i\varphi} = \frac{\mu_\varphi}{\rho_\varphi Sc_{i\varphi}}$

The species conservation equations are in openfoam are represented as

$$\frac{\partial \alpha_\varphi \rho_\varphi Y_i}{\partial t} + \nabla \cdot (\alpha_\varphi \rho_\varphi \bar{\mathbf{U}}_\varphi Y_i) - \nabla \frac{\alpha_\varphi \mu_\varphi}{Sc_{i\varphi}} \nabla (Y_i) = \frac{dm_i}{dt}$$

massTransfer Therory

The rate of mass transport is typically described by

$$\frac{dm_i}{dt} = k_{i\varphi} a(C_i^* - C_i)$$

The manipulation $Y_i = \frac{C_i}{\rho_\varphi}$ is applied and yields

$$\frac{dm_i}{dt} = \rho_\varphi k_{i\varphi} a(Y_i^* - Y_i)$$

$(Y_i^* - Y_i)$ is the driving force for mass transfer, where, Y_i^* is the saturation concentration at the interface.

Saturation Concentration

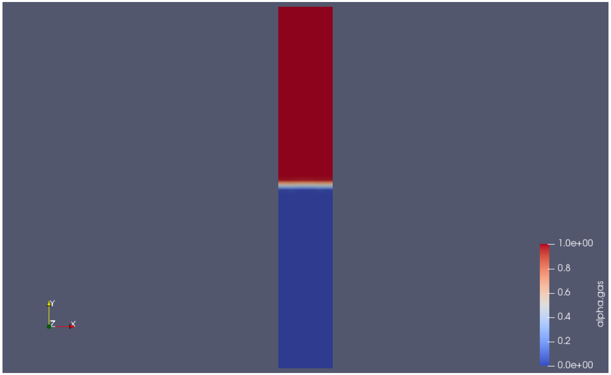
- The saturation concentration Y_i^* is dependent on the temperature at the interface.
- Predicted using saturation models which predict saturation pressure p_i^*.
- Conversion to concentration is achieved using ideal gas equation.

$$C_i^* = \frac{MW_i}{RT} p_i^* \qquad \rho_\varphi = \frac{MW_\varphi}{RT} p$$

$$Y_i = \frac{C_i}{\rho_\varphi} \qquad Y_i^* = \frac{MW_i}{MW_\varphi} \frac{p_i^*}{p}$$

bubbleColumnEvaporatingDissolving

The mass transfer and heat transfer at the interface will be analysed for the bubbleColumnEvaporatingDissolvingCase.



`Bubble column at time 0`

Structure of case directory

At initialization the case directory will consist of three directories.

- 0
- constant
- system

File name	File description
blockMeshDict	Dictionary which contains instructions on how to generate the mesh structure.
controlDict	Dictionary which contains the run control parameters for the solution procedure.
fvSchemes	Dictionary which contains the discretization schemes used in the solution. This dictionary is run-time selectable.
fvSolution	Dictionary where the equation solvers, tolerances and other algorithm controls are set.
setFieldsDict	Dictionary which contains instructions on generating a non uniform initial conditions in a case.

Description of files in the system directory

File name	File description
g	Dictionary which defines the gravitational constant
phaseProperties	Dictionary which defines the phase system model, and the interfacial models for the system.
thermophysicalProperties	Dictionary which defines the phase model for an individual phase. The phase thermodynamic and transport properties are defined and each phase will require an individual thermophysicalProperties dictionary
turbulenceProperties	Dictionary which defines the turbulance within each individual phase. Each phase will require an individual turbulancePropertiesDictionary.

Description of files in the constant directory

Boundary and Initial conditions

Variable Notation	Variable meaning	Value
air.gas	Mass fraction of air in the gas phase	Uniform 1
air.liquid	Mass fraction of air in the liquid phase	Uniform 0
alpha.gas	phase fraction of gas	Uniform 1
alpha.liquid	phase fraction of liquid	Uniform 1
p	Pressure	Uniform 1e5
p_rgh	Pressure without hydrostatic pressure	Uniform 1e5
T.gas	Temperature of the gas phase	350
T.liquid	Temperature of the liquid phase	350
U.gas	Velocity of gas phase	Uniform [0 0.1 0]
U.liquid	Velocity of liquid phase	Uniform [0 0 0]
water.gas	Mass fraction of water in the gas phase	Uniform 0
water.liquid	Mass fraction of water in the liquid phase	Uniform 1

Boundary condition variables to be set in the /0 directory

Setfields

Set fields is used to add liquid to the bubble column.

```
defaultFieldValues
(
    volScalarFieldValue alpha.gas 1
    volScalarFieldValue alpha.liquid 0
);

regions
(
    boxToCell
    {
        box (0 0 0) (0.15 0.501 0.1);
        fieldValues
        (
            volScalarFieldValue alpha.gas 0
            volScalarFieldValue alpha.liquid 1
        );
    }
);
```

phaseProperties

```
type    interfaceCompositionPhaseChangeTwoPhaseSystem;

phases (gas liquid);

gas
{
    type            multiComponentPhaseModel;
    diameterModel   isothermal;
    isothermalCoeffs
    {
        d0              3e-3;
        p0              1e5;
    }
    Sc              0.7;

    residualAlpha   1e-6;
}

    liquid
{
    type            multiComponentPhaseModel;
    diameterModel constant;
    constantCoeffs
    {
        d               1e-4;
    }
    Sc              0.7;

    residualAlpha   1e-6;
}
```

Two-phase systems

Two Phase System	Description
heatAndMomentumTransfer TwoPhaseSystem	This system is the most basic system available. It models momentum and heat transfer in a twoPhaseSystem. Drag, virtual mass, lift, wall lubrication and turbulent dispersion are all modelled in this two phase system.
interfaceCompositionPhaseChange TwoPhaseSystem	This model adds the effect of mass transfer to the basic model. The mass transfer between the phases is calculated according to an interface composition model. The driving force for mass transfer is concentration gradient.
thermalPhaseChangeTwoPhase System	This model adds the effect of mass transfer to the basic model. The mass transfer between the phases is calculated according to a thermal model. The driving force for mass transfer is a temperature gradient. This model should be used for cases involving evaporation and condensation.

Types of two phase systems available in reactingTwoPhaseEulerFoam

Phase Model	Description
multiComponentPhaseModel	This phase model represents a phase with multiple species.
pureIsothermalPhaseModel	This model represents a pure phase model for which the temperature (strictly energy) remains constant.
purePhaseModel	This model represents a pure phase.
reactingPhaseModel	This model represents a phase with multiple species and volumetric reactions.

Types of phase models available in reactingTwoPhaseEulerFoam

Interface Composition and Mass Transfer

```
    interfaceComposition
(
    (gas in liquid)
    {
        type Saturated;
        species ( water );
        Le 1.0;
        saturationPressure
        {
            type ArdenBuck;
        }
    }

    (liquid in gas)
    {
        type Henry;
        species ( air );
        k ( 1.492e-2 );
        Le 1.0;
    }
);
```

```
    massTransfer.gas
(
    (gas in liquid)
    {
        type spherical;
        Le 1.0;
    }

    (liquid in gas)
    {
        type Frossling;
        Le 1.0;
    }
);

massTransfer.liquid
(
    (gas in liquid)
    {
        type Frossling;
        Le 1.0;
    }

    (liquid in gas)
    {
        type spherical;
        Le 1.0;
    }
);
```

Thermophysical properties of the gas phase

```
thermoType
{
    type            heRhoThermo;
    mixture         multiComponentMixture;
    transport       const;
    thermo          hConst;
    equationOfState perfectGas;
    specie          specie;
    energy          sensibleInternalEnergy;
}

species
(
    air
    water
);

inertSpecie air;

"(mixture|air)"
{
    specie
    {
        molWeight   28.9;
    }
    thermodynamics
    {
        Hf          0;
        Cp          1012.5;
    }
    transport
    {
        mu          1.84e-05;
        Pr          0.7;
    }
}
water
{
    specie
    {
        molWeight   18.0153;
    }
    thermodynamics
    {
        Hf          -1.3435e+07;
        Cp          1857.8;
    }
    transport
    {
        mu          1.84e-05;
        Pr          0.7;
    }
}
```

reactingTwoPhaseEulerFoam

- The executable file `reactingTwoPhaseEulerFoam.C` is located in the directory
  `$FOAM_SOLVERS/multiphase/reactingEulerFoam/reactingTwoPhaseEulerFoam`
- The `reactingEulerFoam` directory contains all of the definitions of the phase systems, phase models and the interface compositions.
- The `reactingTwoPhaseEulerFoam` directory contains the two-phase system class definitions which inherit from the classes defined in the `reactingEulerFoam` directory.

reactingTwoPhaseEulerFoam.C

```
#include "fvCFD.H"
#include "twoPhaseSystem.H"
#include "phaseCompressibleTurbulenceModel.H"
#include "pimpleControl.H"
#include "localEulerDdtScheme.H"
#include "fvcSmooth.H"
```

- The file starts by declaring header files required to perform the finite volume analysis and pimple algorithm.
- The file `"twoPhaseSystem.H"` is the declaration file of the twoPhaseSystem class which is the main class in the reactingTwoPhaseEulerFoam solver.

reactingTwoPhaseEulerFoam execution

```
#include "postProcess.H"

#include "addCheckCaseOptions.H"
#include "setRootCase.H"
#include "createTime.H"
#include "createMesh.H"
#include "createControl.H"
#include "createTimeControls.H"
#include "createFields.H"
#include "createFieldRefs.H"
```

- The above files not to confused with declaration files. They contain instructions to create the case, mesh, runtime and control objects.
- `"createFields.H"` contains the instructions to create the two-phase system for the case.

createFields.H

```
Info<< "Creating phaseSystem\n" << endl;

autoPtr<twoPhaseSystem> fluidPtr
(
    twoPhaseSystem::New(mesh)
);
twoPhaseSystem& fluid = fluidPtr();
```

- The two-phase system is constructed and assigned the name fluid.
- For the case described the fluid object will be a two-phase system of type interfaceCompositionPhaseChangeTwoPhaseSystem.
- The individual phase models for the two-phase system will be of type multiComponentPhaseModel.

twoPhaseSystem Construction

```
    Foam::twoPhaseSystem::twoPhaseSystem
(
    const fvMesh& mesh
)
:
    phaseSystem(mesh),
    phase1_(phaseModels_[0]),
    phase2_(phaseModels_[1])
{
    phase2_.volScalarField::operator=(scalar(1) - phase1_);

    volScalarField& alpha1 = phase1_;
    mesh.setFluxRequired(alpha1.name());
}
```

- mesh argument is required for construction
- Inheritance from phaseSystem
- phase1 and phase2 member data are initialized at construction
- Notice that phase2 is calculated from phase1

New() Function

```
Foam::autoPtr<Foam::twoPhaseSystem> Foam::twoPhaseSystem::New
(
    const fvMesh& mesh
)
{
    const word systemType
    (
        IOdictionary
        (
            IOobject
            (
                propertiesName,
                mesh.time().constant(),
                mesh,
                IOobject::MUST_READ_IF_MODIFIED,
                IOobject::NO_WRITE,
                false
            )
        ).lookup("type")
    );
    Info<< "Selecting twoPhaseSystem " << systemType << endl;

    auto cstrIter = dictionaryConstructorTablePtr_->cfind(systemType);

    }

    return cstrIter()(mesh);
}
```

- Mesh argument passed to the New function.
- Two phase system is read from the propertiesName dictionary by looking up the word "type".
- The object systemType is assigned to the two-phase system.
- systemType is then searched for in the runtime selectable table.
- If found the two-phase system is returned with "mesh" as an argument.

Typedef of interfaceCompositionPhaseChangeTwoPhaseSystem

```
typedef
InterfaceCompositionPhaseChangePhaseSystem
<
    MomentumTransferPhaseSystem<twoPhaseSystem>
    >
interfaceCompositionPhaseChangeTwoPhaseSystem;
```

```
typedef
AnisothermalPhaseModel
<
    MultiComponentPhaseModel
    <
        InertPhaseModel
        <
            MovingPhaseModel
            <
            ThermoPhaseModel<phaseModel, rhoReactionThermo>
            >
        >
    >
>
multiComponentPhaseModel;
```

- A number of equations are solved in the reactingTwoPhaseEulerFoam Algorithm.
- Equations to be solved depend on the two-phase system and phase models chosen at runtime.
- Continuity equations are member data of the phase models and phase transfer terms are member data of the phase systems

Variable Locations based on typedefs

Object	Location
UEqn()	phaseSystems/phaseModel/MovingPhaseModel/ MovingPhaseModel.C
heEqn()	phaseSystems/phaseModel/AnisothermalPhaseModel/ AnisothermalPhaseModel.C
YiEqn(Yi)	phaseSystems/phaseModel/MultiComponentPhaseModel/ MultiComponentPhaseModel.C
momentumTransfer()	phaseSystems/PhaseSystems/MomentumTransferPhaseSystem/ MomentumTransferPhaseSystem.C
heatTransfer()	phaseSystems/PhaseSystems/ HeatAndMassTransferPhaseSystem/ HeatAndMassTransferPhaseSystem.C
massTransfer()	phaseSystems /PhaseSystems/ InterfaceCompositionPhaseChangePhaseSystem/ InterfaceCompositionPhaseChangePhaseSystem.C

createFields.H (Cont'd)

```
dimensionedScalar pMin
(
    "pMin",
    dimPressure,
    fluid
);

#include "gh.H"

volScalarField& p = fluid.phase1().thermo().p();

Info<< "Reading field p_rgh\n" << endl;
volScalarField p_rgh
(
    IOobject
    (
        "p_rgh",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);
```

- createFields the initializes the thermodymanic pressure and the pressure without hydrostatic effects.

createFieldRefs.H

```
phaseModel& phase1 = fluid.phase1();
phaseModel& phase2 = fluid.phase2();

volScalarField& alpha1 = phase1;
volScalarField& alpha2 = phase2;

volVectorField& U1 = phase1.U();
surfaceScalarField& phi1 = phase1.phi();
surfaceScalarField& alphaPhi1 = phase1.alphaPhi();
surfaceScalarField& alphaRhoPhi1 = phase1.alphaRhoPhi();

volVectorField& U2 = phase2.U();
surfaceScalarField& phi2 = phase2.phi();
surfaceScalarField& alphaPhi2 = phase2.alphaPhi();
surfaceScalarField& alphaRhoPhi2 = phase2.alphaRhoPhi();
```

```
surfaceScalarField& phi = fluid.phi();

rhoThermo& thermo1 = phase1.thermo();
rhoThermo& thermo2 = phase2.thermo();

volScalarField& rho1 = thermo1.rho();
const volScalarField& psi1 = thermo1.psi();

volScalarField& rho2 = thermo2.rho();
const volScalarField& psi2 = thermo2.psi();

const IOMRFZoneList& MRF = fluid.MRF();
fv::options& fvOptions = fluid.fvOptions();
```

- This file creates variable references from the fluid object just created.
- Notice how the phaseModel class is referanced and how the alpha field is created form it.
- The object thermo contains all the thermodynamic properties of the phases.

reactingTwoPhaseEulerFoam.C (Cont'd)

```
if (!LTS)
{
    #include "CourantNo.H"
    #include "setInitialDeltaT.H"
}

bool faceMomentum
(
    pimple.dict().lookupOrDefault("faceMomentum", false)
);

bool implicitPhasePressure
(
    mesh.solverDict(alpha1.name()).lookupOrDefault
    (
        "implicitPhasePressure", false
    )
);
```

- If a local time step is not declared it will be determined via Courant number
- Searches pimple dictionary to see if face momentum was selected
- Searches the solverDictionary to see if phase pressure should be treated implicitly

reactingTwoPhaseEulerFoam.C (Cont'd)

```
Info<< "\nStarting time loop\n" << endl;

while (runTime.run())
{
    #include "readTimeControls.H"

    int nEnergyCorrectors
    (
        pimple.dict().lookupOrDefault<int>("nEnergyCorrectors", 1)
    );

    if (LTS)
    {
        #include "setRDeltaT.H"
        if (faceMomentum)
        {
            #include "setRDeltaTf.H"
        }
    }
    else
    {
        #include "CourantNos.H"
        #include "setDeltaT.H"
    }

    runTime++;
    Info<< "Time = " << runTime.timeName() << nl << endl;
```

- Time loop begins
- Time step is selected
- Run time settings can be changed while the solver is running. Therefore, the method used to determine the time step must be checked at every time iteration

reactingTwoPhaseEulerFoam.C (Cont'd)

```
// --- Pressure-velocity PIMPLE corrector loop
while (pimple.loop())
{
    fluid.solve();
    fluid.correct();

    #include "YEqns.H"
```

- pimple loop begins
- solve member function of the fluid object is called. This function solves the bounded phase continuity equation for the fluid object. It is defined in the twoPhaseSystem class.
- correct member function of the fluid object is called. This function corrects the properties of the fluid which are not dependent on thermodynamics and turbulence. It is defined in the phaseSystem class and is inherited by the twoPhaseSystem class.

YEqns.H

- The mass transfer in the system is calculated.
- The species fraction equation within each phase is solved.

```
autoPtr<phaseSystem::massTransferTable>
    massTransferPtr(fluid.massTransfer());

phaseSystem::massTransferTable&
    massTransfer(massTransferPtr());

PtrList<volScalarField>& Y1 = phase1.Y();
PtrList<volScalarField>& Y2 = phase2.Y();
```

- A pointer to a mass transfer table is defined.
- The pointer is then initialized with the mass transfer function of the fluid object.
- The object massTransfer is then initialized as a reference to the pointer.
- phase.Y() returns the species in the phase. For the tutorial case this function will return two species for each of the phases, air and water.

YEqns.H (Cont'd)

```
forAll(Y1, i)
{
    tmp<fvScalarMatrix> Y1iEqn(phase1.YiEqn(Y1[i]));

    if (Y1iEqn.valid())
    {
        Y1iEqn =
        (
            Y1iEqn
            ==
              *massTransfer[Y1[i].name()]
              + fvOptions(alpha1, rho1, Y1[i])
        );

        Y1iEqn->relax();
        Y1iEqn->solve(mesh.solver("Yi"));
    }
}
```

- The application continues and executes a for loop which will loop through all of the species in phase 1.
- An fvScalarMatrix called Y1iEqn is then initialized with the species conservation equation for the species in the current iteration of the for loop.
- YiEqn function is defined in the phase model of each phase.
- Y1[i] is referring to the species in the current iteration of the for loop.
- The YiEqn is then rewritten to include the mass transfer term in the species conservation equation.
- The equation is then relaxed and the species concentration equation is solved.

massTransfer Function

The function massTransfer is a member of the `interfaceCompositionPhaseChagePhaseSystem` class

```
//Mass transfer coefficient
const volScalarField K
(
    this->massTransferModels_[key][phase.name()]->K()
);

//Mass transfer coefficient including diffusivity
const volScalarField KD
(
    K*compositionModel.D(member)
);
//Saturation concentration
const volScalarField Yf
(
    compositionModel.Yf(member, Tf)
);

// Implicit transport through the phase
*eqns[name] +=
    phase.rho()*KD*Yf
        - fvm::Sp(phase.rho()*KD, eqns[name]->psi());
```

- mass transfer model is chosen from runtime
- Diffusivity function of interface model chosen at runtime
- Saturation concentration function of the interface model chosen at runtime

$$\frac{dm_i}{dt} = \rho_\varphi k_{i\varphi} a(Y_i^* - Y_i)$$

Saturation Concentration

The interface composition model chosen for this case at runtime was the saturated model.

```
template<class Thermo, class OtherThermo>
Foam::tmp<Foam::volScalarField>
Foam::interfaceCompositionModels::Saturated<Thermo, OtherThermo>::wRatioByP() const
{
    const dimensionedScalar Wi
    (
        "W",
        dimMass/dimMoles,
        this->thermo_.composition().W(saturatedIndex_)
    );

    return Wi/this->thermo_.W()/this->thermo_.p();

template<class Thermo, class OtherThermo>
Foam::tmp<Foam::volScalarField>
Foam::interfaceCompositionModels::Saturated<Thermo, OtherThermo>::Yf
(
    const word& speciesName,
    const volScalarField& Tf
) const
{
    if (saturatedName_ == speciesName)
    {
        return wRatioByP()*saturationModel_->pSat(Tf);
    }
```

$$Y_i^* = \frac{MW_i}{MW_\varphi}\frac{p_i^*}{p}$$

YiEqn Function

The YiEqn member functions for both phases are inherited from the multiphaseComponentPhaseModel class.

```
template<class BasePhaseModel>
Foam::tmp<Foam::fvScalarMatrix>
Foam::MultiComponentPhaseModel<BasePhaseModel>::YiEqn
(
    volScalarField& Yi
)
{

return
(
    fvm::ddt(alpha, rho, Yi)
     + fvm::div(alphaRhoPhi, Yi, "div(" + alphaRhoPhi.name() + ",Yi)")
     - fvm::Sp(this->continuityError(), Yi)

     - fvm::laplacian
    (
        fvc::interpolate(alpha)
          *fvc::interpolate(this->turbulence().muEff()/Sc_),
            Yi
    )
```

$$\frac{\partial \alpha_\varphi \rho_\varphi Y_i}{\partial t} + \nabla \cdot (\alpha_\varphi \rho_\varphi \bar{\mathbf{U}}_\varphi Y_i) - \nabla \frac{\alpha_\varphi \mu_\varphi}{Sc_{i\varphi}} \nabla (Y_i) = \frac{dm_i}{dt}$$

reactingTwoPhaseEulerFoam.C (Cont'd)

```
{
    #include "pU/UEqns.H"
    #include "EEqns.H"
    #include "pU/pEqn.H"
}

fluid.correctKinematics();

if (pimple.turbCorr())
{
    fluid.correctTurbulence();
}
}

runTime.write();

runTime.printExecutionTime(Info);
}
```

- The application then solves the momentum equations described the introduction without the effects of pressure, turbulent drag and the explicit part of the drag term.
- Next the energy equation for the system is solved
- The pressure equation for the system is then solved. This is where the pressure-velocity linking for the system occurs.
- The kinematic and turbulent properties of the fluid system are then corrected and specified data is written to runtime.