```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | www.openfoam.com
     \\/     M anipulation  |
-------------------------------------------------------------------------------
    Copyright (C) 2017-2020 OpenCFD Ltd.
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

\*---------------------------------------------------------------------------*/

#include "LeeCNT.H"
#include "addToRunTimeSelectionTable.H"
#include "mathematicalConstants.H"


// * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * //

template<class Thermo, class OtherThermo>
Foam::meltingEvaporationModels::LeeCNT<Thermo, OtherThermo>::LeeCNT
(
    const dictionary& dict,
    const phasePair& pair
)
:
    InterfaceCompositionModel<Thermo, OtherThermo>(dict, pair),
    C_("C", inv(dimTime), dict),
    Tactivate_("Tactivate", dimTemperature, dict),
    planck_("planck", dimEnergy*dimTime, dict),
    boltzmann_("boltzmann", dimEnergy/dimTemperature, dict),
    deltag_("deltag", dimEnergy, dict),
    nL_("nL", inv(dimVolume), dict),
    gammaYW_("gammaYW", dimEnergy/dimArea, dict),
    hLV_("hLV", dimEnergy/dimVolume, dict),
    alphaEY_("alphaEY", dict),
    alphaMin_(dict.getOrDefault<scalar>("alphaMin", 0)),
    interfaceVolume_
    (
        IOobject
        (
            "cellVolume",
            this->mesh_.time().timeName(),
            this->mesh_,
            IOobject::NO_READ,
            IOobject::NO_WRITE
        ),
        this->mesh_,
        dimensionedScalar(dimVolume, Zero)
    )
{}


// * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * * //


template<class Thermo, class OtherThermo>
Foam::tmp<Foam::volScalarField>
Foam::meltingEvaporationModels::LeeCNT<Thermo, OtherThermo>::Kexp
(
    const volScalarField& refValue
)
{

    {
        const fvMesh& mesh = this->mesh_;
        const volScalarField deltaG
        (
            (16.0 * (constant::mathematical::pi) * pow(gammaYW_,3.0) * pow(Tactivate_,2.0) * alphaEY_)/(3.0 *
pow(hLV_,2.0) * pow((Tactivate_ - refValue),2.0))
        );

        const volScalarField J
        (
            (boltzmann_*refValue)/(planck_) * (exp(-deltag_/(boltzmann_*refValue))  * nL_ *
exp(-deltaG/(boltzmann_*refValue)))
        );

        forAll(interfaceVolume_, celli)
        {
            interfaceVolume_[celli] = mesh.V()[celli];
        }

        const volScalarField lambda
        (
            J*interfaceVolume_
        );

        const volScalarField from
        (
            min(max(this->pair().from(), scalar(0)), scalar(1))
        );

        const volScalarField coeff
        (
            C_*from*this->pair().from().rho()*pos(from - alphaMin_)
           *(refValue - Tactivate_)
           /Tactivate_
        );

        const volScalarField coeff1
        (
            -lambda*from*this->pair().from().rho()*pos(from - alphaMin_)
           *(refValue - Tactivate_)
           /Tactivate_
        );

        if (sign(C_.value()) > 0)
        {
            return
            (
                coeff*pos(refValue - Tactivate_)
            );
        }
        else
        {
            return
            (
                coeff1*pos(Tactivate_ - refValue)
            );
        }
    }
}


template<class Thermo, class OtherThermo>
Foam::tmp<Foam::volScalarField>
Foam::meltingEvaporationModels::LeeCNT<Thermo, OtherThermo>::KSp
(
    label variable,
    const volScalarField& refValue
)
{

    if (this->modelVariable_ == variable)
    {
        const fvMesh& mesh = this->mesh_;

        const volScalarField deltaG
        (
            (16.0 * (constant::mathematical::pi) * pow(gammaYW_,3.0) * pow(Tactivate_,2.0) * alphaEY_)
            /(3.0 * pow(hLV_,2.0) * pow((Tactivate_ - refValue),2.0))
        );

        const volScalarField J
        (
            (boltzmann_*refValue)/(planck_) * (exp(-deltag_/(boltzmann_*refValue))  * nL_ *
exp(-deltaG/(boltzmann_*refValue)))
        );
        forAll(interfaceVolume_, celli)
        {
            interfaceVolume_[celli] = mesh.V()[celli];
        }
        const volScalarField lambda
        (
            J*interfaceVolume_
        );

        volScalarField from
        (
            min(max(this->pair().from(), scalar(0)), scalar(1))
        );

        const volScalarField coeff
        (
            C_*from*this->pair().from().rho()*pos(from - alphaMin_)
            /Tactivate_
        );

        const volScalarField coeff1
        (
            -lambda*from*this->pair().from().rho()*pos(from - alphaMin_)
            /Tactivate_
        );

        if (sign(C_.value()) > 0)
        {
            return
            (
                coeff*pos(refValue - Tactivate_)
            );
        }
        else
        {
            return
            (
                coeff1*pos(Tactivate_ - refValue)
            );
        }
    }
    else
    {
        return tmp<volScalarField> ();
    }
}


template<class Thermo, class OtherThermo>
Foam::tmp<Foam::volScalarField>
Foam::meltingEvaporationModels::LeeCNT<Thermo, OtherThermo>::KSu
(
    label variable,
    const volScalarField& refValue
)
```

```
{

    if (this->modelVariable_ == variable)
    {
        const fvMesh& mesh = this->mesh_;

        const volScalarField deltaG
        (
            (16.0 * (constant::mathematical::pi) * pow(gammaYW_,3.0) * pow(Tactivate_,2.0) * alphaEY_)
            /(3.0 * pow(hLV_,2.0) * pow((Tactivate_ - refValue),2.0))
        );

        const volScalarField J
        (
            (boltzmann_*refValue)/(planck_) * (exp(-deltag_/(boltzmann_*refValue))  * nL_ *
exp(-deltaG/(boltzmann_*refValue)))
        );

        forAll(interfaceVolume_, celli)
        {
            interfaceVolume_[celli] = mesh.V()[celli];
        }
        const volScalarField lambda
        (
            J*interfaceVolume_
        );

        volScalarField from
        (
            min(max(this->pair().from(), scalar(0)), scalar(1))
        );

        const volScalarField coeff
        (
            C_*from*this->pair().from().rho()*pos(from - alphaMin_)
        );

        const volScalarField coeff1
        (
            -lambda*from*this->pair().from().rho()*pos(from - alphaMin_)
        );

        if (sign(C_.value()) > 0)
        {
            return
            (
                -coeff*pos(refValue - Tactivate_)
            );
        }
        else
        {
            return
            (
                -coeff1*pos(Tactivate_ - refValue)
            );
            }
        }
        else
        {
            return tmp<volScalarField> ();
        }
    }


template<class Thermo, class OtherThermo>
const Foam::dimensionedScalar&
Foam::meltingEvaporationModels::LeeCNT<Thermo, OtherThermo>::Tactivate() const
{
    return Tactivate_;
}

template<class Thermo, class OtherThermo>
const Foam::dimensionedScalar&
Foam::meltingEvaporationModels::LeeCNT<Thermo, OtherThermo>::planck() const
{
    return planck_;
}

template<class Thermo, class OtherThermo>
const Foam::dimensionedScalar&
Foam::meltingEvaporationModels::LeeCNT<Thermo, OtherThermo>::boltzmann() const
{
    return boltzmann_;
}

template<class Thermo, class OtherThermo>
const Foam::dimensionedScalar&
Foam::meltingEvaporationModels::LeeCNT<Thermo, OtherThermo>::deltag() const
{
    return deltag_;
}

template<class Thermo, class OtherThermo>
const Foam::dimensionedScalar&
Foam::meltingEvaporationModels::LeeCNT<Thermo, OtherThermo>::nL() const
{
    return nL_;
}

template<class Thermo, class OtherThermo>
const Foam::dimensionedScalar&
Foam::meltingEvaporationModels::LeeCNT<Thermo, OtherThermo>::gammaYW() const
{
    return gammaYW_;
}

template<class Thermo, class OtherThermo>
const Foam::dimensionedScalar&
Foam::meltingEvaporationModels::LeeCNT<Thermo, OtherThermo>::hLV() const
```

```
{
    return hLV_;
}

template<class Thermo, class OtherThermo>
const Foam::dimensionedScalar&
Foam::meltingEvaporationModels::LeeCNT<Thermo, OtherThermo>::alphaEY() const
{
    return alphaEY_;
}

template<class Thermo, class OtherThermo>
bool
Foam::meltingEvaporationModels::LeeCNT<Thermo, OtherThermo>::includeDivU()
{
    return true;
}


// ************************************************************************* //
```