Back to basics: implementing point-wise models with MFront

MFront Users Meeting

(1) CEA, DES, IRESNE, DEC, SESC, LMPC, Cadarache, France

T. Helfer(1), Maxence Wangermez(1) (and many others!)

Outline

1. Introduction

Point-wise models

- Point-wise models (swelling due to solid or gaseous fission products, phase transition, chemical reactions) describes the local evolution of a set of *internal state variables* $\left(y_{j}\right)_{j\in\left[1:n_{y}\right]}$ due to the evolution local thermodynamic environment described by a set of *external state variables* $\left(z_{j}\right)_{j\in\left[1:n_{z}\right]}$.

Point-wise models

- Point-wise models (swelling due to solid or gaseous fission products, phase transition, chemical reactions) describes the local evolution of a set of *internal state variables* $\left(y_j\right)_{j\in\left[1:n_y\right]}$ due to the evolution local thermodynamic environment described by a set of *external state variables* $\left(z_j\right)_{j\in\left[1:n_z\right]}$.
- Up to a certain extent, point-wise models may also be used to describe the evolution of a structure:
  - Oxidation of Zircaloy pipes.

Point-wise models

- Point-wise models (swelling due to solid or gaseous fission products, phase transition, chemical reactions) describes the local evolution of a set of *internal state variables* $\left(y_j\right)_{j\in\left[1:n_y\right]}$ due to the evolution local thermodynamic environment described by a set of *external state variables* $\left(z_j\right)_{j\in[1:n_z]}$.
- Up to a certain extent, point-wise models may also be used to describe the evolution of a structure:
  - Oxidation of Zircaloy pipes.
- Point-wise models can be seen as a simpliest case of generic behaviours:
  - No thermodynamic force nor consistent tangent operator.

Point-wise models

- Point-wise models (swelling due to solid or gaseous fission products, phase transition, chemical reactions) describes the local evolution of a set of *internal state variables* $\left(y_j\right)_{j\in\left[1:n_y\right]}$ due to the evolution local thermodynamic environment described by a set of *external state variables* $\left(z_j\right)_{j\in\left[1:n_z\right]}$.
- Up to a certain extent, point-wise models may also be used to describe the evolution of a structure:
  - Oxidation of Zircaloy pipes.
- Point-wise models can be seen as a simpliest case of generic behaviours:
  - No thermodynamic force nor consistent tangent operator.
- Point-wise models can be used:
  - directly in a `PLEIADES` application, in `MTest`, in `Cast3M`, in `Manta` or in any solver interfaced with `MGIS`.
  - as a building block for behaviours.

Evolution of internal state variables, notations

- The *internal state variables* $\left(y_j\right)_{j\in\left[1:n_y\right]}$ are packed in a vector $\vec{Y}$:

$$\vec{Y} = \begin{pmatrix} y_1 & \ldots & y_{n_y} \end{pmatrix}^T$$

  - Note that this notation does not make any assumption on tensorial nature of the state variable y_j that may a scalar, a symmetric tensor, an unsymmetric tensor, etc...

Evolution of internal state variables, notations

- The *internal state variables* $\left(y_j\right)_{j\in\left[1:n_y\right]}$ are packed in a vector $\vec{Y}$:

$$\vec{Y} = \begin{pmatrix} y_1 & \ldots & y_{n_y} \end{pmatrix}^T$$

  - Note that this notation does not make any assumption on tensorial nature of the state variable y_j that may a scalar, a symmetric tensor, an unsymmetric tensor, etc...

- The *external state variables* $\left(z_j\right)_{j\in[1:n_z]}$ packed in a vector $\vec{Z}$:

$$\vec{Z} = \begin{pmatrix} z_1 & \ldots & z_{n_z} \end{pmatrix}^T$$

Evolution of internal state variables, notations

- The *internal state variables* $\left(y_j\right)_{j\in\left[1:n_y\right]}$ are packed in a vector $\vec{Y}$:

$$\vec{Y} = \begin{pmatrix} y_1 & \ldots & y_{n_y} \end{pmatrix}^T$$

  - Note that this notation does not make any assumption on tensorial nature of the state variable y_j that may a scalar, a symmetric tensor, an unsymmetric tensor, etc...

- The *external state variables* $\left(z_j\right)_{j\in[1:n_z]}$ packed in a vector $\vec{Z}$:

$$\vec{Z} = \begin{pmatrix} z_1 & \ldots & z_{n_z} \end{pmatrix}^T$$

- The evolution of the internal state variables $\vec{Y}$ is assumed to obey a standard first-order ordinary differential equation (ODE) given by:

$$\dot{\vec{Y}} = \frac{\mathrm{d}\,\vec{Y}}{\mathrm{d}\tau} = \vec{G}\left(\vec{Y}\left(\tau\right),\vec{Z}\left(\tau\right)\right)$$

where τ denotes the time variable.

Time discretization, notations

- Time is discretized in intervals named time steps.
  - t denotes the time at the beginning of the (considered) time step.
  - $\Delta\, t$ is the time increment during the time step.
  - $t + \Delta\, t$ corresponds to the time at the end of the step.

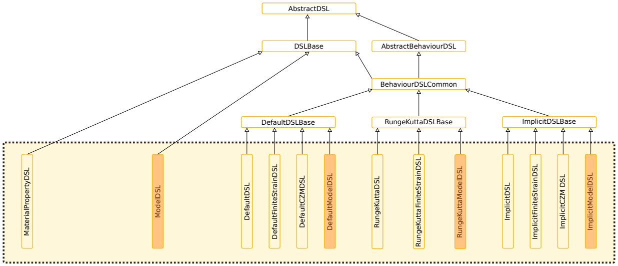
Time discretization, notations

- Time is discretized in intervals named time steps.
  - t denotes the time at the beginning of the (considered) time step.
  - $\Delta\, t$ is the time increment during the time step.
  - $t + \Delta\, t$ corresponds to the time at the end of the step.
- Models in `MFront` assumes that the values of the external state variables at the beginning of the time step, denoted $\left.\vec{Z}\right|_{t}$ and their increments during the time step, denoted $\Delta\,\vec{Z}$, are known.
  - Since only $\left.\vec{Z}\right|_{t}$ and $\Delta\,\vec{Z}$ are known, the evolution of $\vec{Z}$ during the time step can only be described using a linear interpolation:

$$\left.\vec{Z}\right|_{t+\theta\,\Delta\,t} = \left.\vec{Z}\right|_{t} + \theta\,\Delta\,\vec{Z}$$

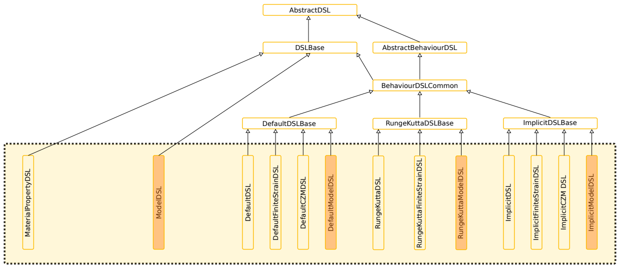
    where θ is a number in the range $[0, 1]$
  - This assumption generally implies that high order algorithms are generally useless.

Domain specific languages related to models



- The `Model` DSL is historically the first DSL dedicated to models.

Domain specific languages related to models



- The `Model` DSL is historically the first DSL dedicated to models.
- The `DefaultModel`, the `RungeKuttaModel` and the `ImplicitModel` have been introduced in recent versions of `MFront` as a consequence of the work on generic behaviours.
  - Those DSLs are much more powerful than the `Model` DSL.
  - Their conventions are more consistent with the rest of `MFront`.
  - Alas, there is no interface for the `PLEIADES` architecture nor `licos` yet.

Outline

2. A simple chemical reaction

Chemical reaction

- To illustrate the implementation of models with ‘MFront‘ , let us consider a system of two chemical species A and B whose evolution is given by the following reaction:

$$A \underset{k_2}{\overset{k_1}{\rightleftharpoons}} B$$

Chemical reaction

- To illustrate the implementation of models with ‘MFront‘ , let us consider a system of two chemical species A and B whose evolution is given by the following reaction:

$$A \underset{k_2}{\overset{k_1}{\rightleftharpoons}} B$$

- The molar concentrations $[A]$ and $[B]$ thus satisfies:

$$\left\{\begin{aligned} \frac{\mathrm{d}\,[A]}{\mathrm{d}\tau} &= k_2\;[B]\,(\tau) - k_1\;[A]\,(\tau) \\ \frac{\mathrm{d}\,[B]}{\mathrm{d}\tau} &= k_1\;[A]\,(\tau) - k_2\;[B]\,(\tau) \end{aligned}\right.$$

Chemical reaction

- To illustrate the implementation of models with ‘MFront‘ , let us consider a system of two chemical species A and B whose evolution is given by the following reaction:

$$A \overset{k_1}{\underset{k_2}{\rightleftharpoons}} B$$

- The molar concentrations $[A]$ and $[B]$ thus satisfies:

$$\left\{\begin{aligned} \frac{\mathrm{d}\,[A]}{\mathrm{d}\tau} &= k_2\ [B]\,(\tau) - k_1\ [A]\,(\tau) \\ \frac{\mathrm{d}\,[B]}{\mathrm{d}\tau} &= k_1\ [A]\,(\tau) - k_2\ [B]\,(\tau) \end{aligned}\right.$$

- $[A] + [B]$ is constant (*conservation of mass*).

Chemical reaction

- To illustrate the implementation of models with ‘MFront‘ , let us consider a system of two chemical species A and B whose evolution is given by the following reaction:

$$A \underset{k_2}{\overset{k_1}{\rightleftharpoons}} B$$

- The molar concentrations $[A]$ and $[B]$ thus satisfies:

$$\left\{\begin{aligned}\frac{\mathrm{d}\,[A]}{\mathrm{d}\tau} &= k_2\;[B]\,(\tau) - k_1\;[A]\,(\tau)\\ \frac{\mathrm{d}\,[B]}{\mathrm{d}\tau} &= k_1\;[A]\,(\tau) - k_2\;[B]\,(\tau)\end{aligned}\right.$$

- $[A] + [B]$ is constant (*conservation of mass*).
- Hence, the evolution of the system is driven by the following condensed equation:

$$\frac{\mathrm{d}\,[A]}{\mathrm{d}\tau} = k_2\left(\left.[A]\right|_t + \left.[B]\right|_t\right) - (k_1 + k_2)\;[A]\,(\tau)$$

Outline

3. Constant reaction rate coefficients. Implementation with the `Model` DSL. First tests

Constant reaction rate coefficients

Reaction rate coefficients	Value
k_1	$\dfrac{1}{60}\,s^{-1}$
k_2	$\dfrac{1}{120}\,s^{-1}$

- If the reaction rate coefficients k_1 and k_2 are assumed constant, a closed formed solution is given by:

$$\left.[A]\right|_{t+\Delta\,t} = \frac{B}{K} + \left(\frac{K\;\left.[A]\right|_{t} - B}{K}\right)\exp\left(-K\,\Delta\,t\right)$$

with: $B = k_2\,\left(\left.[A]\right|_{t} + \left.[B]\right|_{t}\right)$ and $K = k_1 + k_2$.

A first implementation

```
@DSL Model;
@Model ChemicalReaction1;
@Author Thomas Helfer;
@Date 09/07/2022;
@UseQt true;
@UnitSystem SI;

//! molar concentration of species A
@StateVariable quantity<real, 0, 0, 0, 0, 0, 0, 1> ca;
ca.setEntryName("MolarConcentrationOfSpeciesA");
ca.setDepth(1);
//! molar concentration of species B
@StateVariable quantity<real, 0, 0, 0, 0, 0, 0, 1> cb;
cb.setEntryName("MolarConcentrationOfSpeciesB");
cb.setDepth(1);

//! rate coefficient of the reaction transforming species A to species B
@Parameter frequency k1 = 0.016666666666666666;
k1.setEntryName("ReactionRateCoefficientAB");
//! rate coefficient of the reaction transforming species B to species A
@Parameter frequency k2 = 0.008333333333333333;
k2.setEntryName("ReactionRateCoefficientBA");

@Function ChemicalReaction {
  const auto B = k2 * (ca_1 + cb_1);
  const auto K = k1 + k2;
  const auto e = exp(-K * dt);
  ca = ca_1 * e + (B / K) * (1 - e);
  cb = ca_1 + cb_1 - ca;
}
```

Declaration of state variables

```
//! molar concentration of species A
@StateVariable quantity<real, 0, 0, 0, 0, 0, 0, 1> ca;
ca.setEntryName("MolarConcentrationOfSpeciesA");
ca.setDepth(1);
//! molar concentration of species B
@StateVariable quantity<real, 0, 0, 0, 0, 0, 0, 1> cb;
cb.setEntryName("MolarConcentrationOfSpeciesB");
cb.setDepth(1);
```

- `@StateVariable` (or equivalently `@Output`) declares new **scalar** state variables.

Declaration of state variables

```
//! molar concentration of species A
@StateVariable quantity<real, 0, 0, 0, 0, 0, 0, 1> ca;
ca.setEntryName("MolarConcentrationOfSpeciesA");
ca.setDepth(1);
//! molar concentration of species B
@StateVariable quantity<real, 0, 0, 0, 0, 0, 0, 1> cb;
cb.setEntryName("MolarConcentrationOfSpeciesB");
cb.setDepth(1);
```

- `@StateVariable` (or equivalently `@Output`) declares new **scalar** state variables.
- The depth of a state variable x has the following meaning:
  - 0: only $\left.x\right|_{t+\Delta\,t}$, is available and associated with a variable named `x`.
  - 1: $\left.x\right|_{t}$ is associated with the variable `x_1` and $\left.x\right|_{t+\Delta\,t}$ is associated with the variable `x`.
  - higher values are hardly ever been used and highly discouraged.

Declaration of state variables

```
//! molar concentration of species A
@StateVariable quantity<real, 0, 0, 0, 0, 0, 0, 1> ca;
ca.setEntryName("MolarConcentrationOfSpeciesA");
ca.setDepth(1);
//! molar concentration of species B
@StateVariable quantity<real, 0, 0, 0, 0, 0, 0, 1> cb;
cb.setEntryName("MolarConcentrationOfSpeciesB");
cb.setDepth(1);
```

- `@StateVariable` (or equivalently `@Output`) declares new **scalar** state variables.
- The depth of a state variable x has the following meaning:
  - 0: only $\left.x\right|_{t+\Delta\,t}$, is available and associated with a variable named `x`.
  - 1: $\left.x\right|_{t}$ is associated with the variable `x_1` and $\left.x\right|_{t+\Delta\,t}$ is associated with the variable `x`.
  - higher values are hardly ever been used and highly discouraged.
- The `quantity` type allows to define new quantities. Its integer template parameters have the following meaning in SI:

$$kg^{i_1}\,m^{i_2}\,s^{i_3}\,A^{i_4}\,K^{i_5}\,cd^{i_6}\,mol^{i_7}$$

Declaration of state variables

```
//! molar concentration of species A
@StateVariable quantity<real, 0, 0, 0, 0, 0, 0, 1> ca;
ca.setEntryName("MolarConcentrationOfSpeciesA");
ca.setDepth(1);
//! molar concentration of species B
@StateVariable quantity<real, 0, 0, 0, 0, 0, 0, 1> cb;
cb.setEntryName("MolarConcentrationOfSpeciesB");
cb.setDepth(1);
```

- `@StateVariable` (or equivalently `@Output`) declares new **scalar** state variables.
- The depth of a state variable x has the following meaning:
  - 0: only $\left.x\right|_{t+\Delta\,t}$, is available and associated with a variable named `x`.
  - 1: $\left.x\right|_{t}$ is associated with the variable `x_1` and $\left.x\right|_{t+\Delta\,t}$ is associated with the variable `x`.
  - higher values are hardly ever been used and highly discouraged.
- The `quantity` type allows to define new quantities. Its integer template parameters have the following meaning in SI:

$$kg^{i_1}\ m^{i_2}\ s^{i_3}\ A^{i_4}\ K^{i_5}\ cd^{i_6}\ mol^{i_7}$$

- Variables can be documented using a `doxygen`-like syntax.

The `@Function` keyword

```
@Function ChemicalReaction {
  const auto B = k2 * (ca_1 + cb_1);
  const auto K = k1 + k2;
  const auto e = exp(-K * dt);
  ca = ca_1 * e + (B / K) * (1 - e);
  cb = ca_1 + cb_1 - ca;
}
```

- Several functions can be defined, although this feature has hardly been used in pratice. It is highly recommended to define only one function per model.
  - The ‘@Integrator‘ keyword has been introducted in Version 4.1 for consistency with behaviours and does not expect a function name;
- The implementation readily translates the analytical formula in `C++`:

$$\left.[A]\right|_{t+\Delta\,t} = \frac{B}{K} + \left(\frac{K\ \left.[A]\right|_{t} - B}{K}\right)\ \exp\left(-K\,\Delta\,t\right)$$

with: $B = k_2\ \left(\left.[A]\right|_{t} + \left.[B]\right|_{t}\right)$ and $K = k_1 + k_2$.

Compilation and testing with `MTest`

```
// loading the model
@Model 'src/libModel.so' 'ChemicalReaction1';
// initial value of the molar concentration of species B
@Real 'B0' 0.1;
@StateVariable 'MolarConcentrationOfSpeciesB' 'B0';
// time discretization
@Times {
  0, 360 in 100
};
```

- `mfront --obuild --interface=generic ChemicalReaction1.mfront`

Compilation and testing with `MTest`

```
// loading the model
@Model 'src/libModel.so' 'ChemicalReaction1';
// initial value of the molar concentration of species B
@Real 'B0' 0.1;
@StateVariable 'MolarConcentrationOfSpeciesB' 'B0';
// time discretization
@Times {
  0, 360 in 100
};
```

- `mfront --obuild --interface=generic ChemicalReaction1.mfront`
- `mtest ChemicalReaction1.mtest`

Compilation and testing with `MTest`

```
// loading the model
@Model 'src/libModel.so' 'ChemicalReaction1';
// initial value of the molar concentration of species B
@Real 'B0' 0.1;
@StateVariable 'MolarConcentrationOfSpeciesB' 'B0';
// time discretization
@Times {
  0, 360 in 100
};
```

- `mfront --obuild --interface=generic ChemicalReaction1.mfront`
- `mtest ChemicalReaction1.mtest`
- `MTest` generates a file named `ChemicalReaction1.res` which contains the evolution of the state variables as a function of time.

Results of the first test

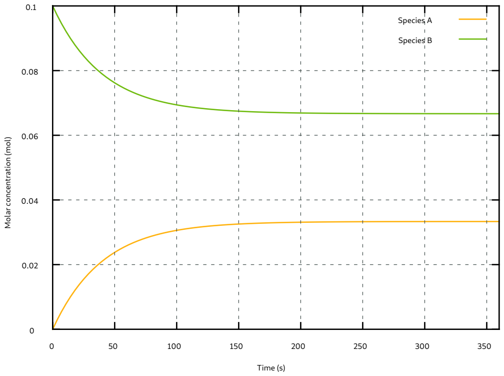


Figure 1: Evolution of the molar concentrations of species $[A]$ and $[B]$.

- `tplot "ChemicalReaction1.res" -u 1:2 -title="Species A"  "ChemicalReaction1.res" -u 1:3 -title="Species B" -with-grid -xlabel="Time (s)" -ylabel="Molar concentration (mol)"`

Adding unit tests

```
// loading the model
@Model 'src/libModel.so' 'ChemicalReaction1';
//  initial  value of the molar concentration of species B
@Real 'B0' 0.1;
@StateVariable 'MolarConcentrationOfSpeciesB' 'B0';
// time discretization
@Times {
  0, 360 in 100
};
// some useful variables
@Real 'k1' 'ChemicalReaction1::ReactionRateCoefficientAB';
@Real 'k2' 'ChemicalReaction1::ReactionRateCoefficientBA';
@Real 'K'  'k1 + k2';
@Real 'B'  'k2 * B0';
// unit tests
@Test<function> 'MolarConcentrationOfSpeciesA' '(B/K) * (1 - exp(-K * t))'       1e-14;
@Test<function> 'MolarConcentrationOfSpeciesB' 'B0 - (B/K) * (1 - exp(-K * t))' 1e-14;
```

- Unit tests can be defined using analytical solutions or reference files.

Using the MTest Python module

```
import std
import tfel.tests
import mtest

mtest.setVerboseMode(mtest.VerboseLevel.VERBOSE_QUIET)

m = mtest.MTest()
m.setAuthor("Thomas Helfer")
m.setDate("09/08/2022")
m.setModel('generic', 'src/libModel.so', 'ChemicalReaction1')
m.setStateVariableInitialValue('MolarConcentrationOfSpeciesB', 0.1)
m.setTimes([3.6 * i for i in range(0, 100)])
output_file = "ChemicalReaction1-python.res".format(k1)
m.setOutputFileName(output_file)
m.execute()
```

- The MTest Python module can be used to identify models using the LDC-0D component.

Parametric studies with the `MTest` `Python` module

```
import std
import  tfel . tests
import mtest

mtest.setVerboseMode(mtest.VerboseLevel.VERBOSE_QUIET)

 for  k1  in  [0.016666666666666666, 0.008333333333333333,
              0.004166666666666667, 0.03333333333333333,
              0.06666666666666667]:
    m = mtest.MTest()
    m.setAuthor("Thomas Helfer")
    m.setDate("09/08/2022")
    m.setModel('generic',  'src/libModel.so', 'ChemicalReaction1')
    m. setStateVariableInitialValue ( 'MolarConcentrationOfSpeciesB', 0.1)
    m.setParameter('ReactionRateCoefficientAB', float(k1))
    m.setTimes([3.6 * i  for  i  in  range(0, 100)])
     output_file  =  "ChemicalReaction1-python-parametric-{}.res".format(k1)
    m.setOutputFileName(output_file)
    m.execute()
```

- See the `MFront`' book to see how to perform parametric studies in `bash`

4. Reaction rate coefficients varying with temperature

Reaction rate coefficients as functions of the temperat

Material properties	Value
k_{10}	$0.018377505387559667\, s^{-1}$
k_{20}	$0.010131981128093540\, s^{-1}$
T_{a1}	$3000\, K^{-1}$
T_{a2}	$1500\, K^{-1}$

- The reaction rate coefficients k_1 and k_2 are now assumed to depend on the temperature following the Arrhenius law, as follows:

$$\begin{cases} k_1 = k_{10}\, \exp\left(-\dfrac{T}{T_{a1}}\right) \\ k_2 = k_{20}\, \exp\left(-\dfrac{T}{T_{a2}}\right) \end{cases}$$

where k_{10}, T_{a1}, k_{20} and T_{a2} are material coefficients.

Numerical schemes

- No closed-form solution exists.
- The ordinary differential equation can be integrated over the time step as follows:

$$\Delta\,\vec{Y} = \int_{t}^{t+\theta\,\Delta\,t} \vec{G}\left(\vec{Y}\left(\tau\right),\vec{Z}\left(\tau\right)\right)\,\mathrm{d}\,\tau$$

- Various numerical schemes can be built by approximating the integral on the right hand side. In this tutorial, we consider:
  - The trapezoidal rule
  - The generalized mid-point rule

Code factorization

- All the implementations based on the `Model` DSL will share the same internal state variables, the same external state variable (the temperature) and the same parameters.

Code factorization

- All the implementations based on the `Model` DSL will share the same internal state variables, the same external state variable (the temperature) and the same parameters.
- Moreover, the parameters' declaration can be shared between all DSLs.

Code factorization

- All the implementations based on the `Model` DSL will share the same internal state variables, the same external state variable (the temperature) and the same parameters.
- Moreover, the parameters' declaration can be shared between all DSLs.
- It is thus convenient to factorize their declaration in two auxiliary files:
  - `ChemicalReaction-parameters.mfront` which contains the declaration of the parameters. This file can be imported by all DSLs.
  - `ChemicalReaction-common.mfront` contains the declaration of the internal state variables and the external state variable and imports the `ChemicalReaction-parameters.mfront` file. This file is only compatible with the `Model` DSL.

Code factorization

- All the implementations based on the `Model` DSL will share the same internal state variables, the same external state variable (the temperature) and the same parameters.
- Moreover, the parameters' declaration can be shared between all DSLs.
- It is thus convenient to factorize their declaration in two auxiliary files:
  - `ChemicalReaction-parameters.mfront` which contains the declaration of the parameters. This file can be imported by all DSLs.
  - `ChemicalReaction-common.mfront` contains the declaration of the internal state variables and the external state variable and imports the `ChemicalReaction-parameters.mfront` file. This file is only compatible with the `Model` DSL.
- The `ChemicalReaction-common.mfront` file can be imported by the `@Import` keyword, as follows:

```
@Import "ChemicalReaction-common.mfront";
```

The ChemicalReaction-common.mfront file

```
@UseQt true;
@UnitSystem SI;

// ! molar concentration of species A
@StateVariable quantity<real, 0, 0, 0, 0, 0, 0, 1> ca;
ca.setEntryName("MolarConcentrationOfSpeciesA");
ca.setDepth(1);
// ! molar concentration of species B
@StateVariable quantity<real, 0, 0, 0, 0, 0, 0, 1> cb;
cb.setEntryName("MolarConcentrationOfSpeciesB");
cb.setDepth(1);

@ExternalStateVariable temperature T;
T.setGlossaryName("Temperature");
T.setDepth(1);

@Import "ChemicalReaction-parameters.mfront";
```

The `ChemicalReaction-parameters.mfront` file

```
//! reference rate coefficient of the reaction transforming species A to species B
@Parameter frequency k01 = 0.018377505387559667;
k01.setEntryName("ReferenceReactionRateCoefficientAB");
//! reference rate coefficient of the reaction transforming species B to species A
@Parameter frequency k02 = 0.010131981128093 54;
k02.setEntryName("ReferenceReactionRateCoefficientBA");
//! activation temperature reaction transforming species B to species A
@Parameter temperature Ta1 = 3000;
Ta1.setEntryName("ActivationTemperatureAB");
//! activation temperature reaction transforming species B to species A
@Parameter temperature Ta2 = 1500;
Ta2.setEntryName("ActivationTemperatureBA");
```

Trapezoidal rule

- The integral on the right hand size can be approximated by the trapezoidal rule as follows:

$$\Delta\,\vec{Y} \approx \frac{\Delta\,t}{2}\left(\vec{G}\left(\vec{Y}\,(t)\,,\vec{Z}\,(t)\right)+\vec{G}\left(\vec{Y}\,(t+\Delta\,t)\,,\vec{Z}\,(t+\Delta\,t)\right)\right)$$

Trapezoidal rule

- The integral on the right hand size can be approximated by the trapezoidal rule as follows:

$$\Delta\,\vec{Y} \approx \frac{\Delta\,t}{2}\left(\vec{G}\left(\vec{Y}\,(t)\,,\vec{Z}\,(t)\right)+\vec{G}\left(\vec{Y}\,(t+\Delta\,t)\,,\vec{Z}\,(t+\Delta\,t)\right)\right)$$

- Applied to the chemical reaction example, this scheme leads to the approximation of the increment $\Delta\;[A]$:

$$\Delta\;[A] = \frac{B}{1+K}$$

where:

- $B = \Delta\,t\,\left(\langle k_2\rangle\;\left.[B]\right|_t - \langle k_1\rangle\;\left.[A]\right|_t\right)$ with
$$\begin{cases}\langle k_1\rangle = \dfrac{k_1\left(\left.T\right|_{t+\Delta\,t}\right)+k_1\left(\left.T\right|_t\right)}{2}\\ \langle k_2\rangle = \dfrac{k_2\left(\left.T\right|_{t+\Delta\,t}\right)+k_2\left(\left.T\right|_t\right)}{2}\end{cases}$$
- $K = \dfrac{\Delta\,t}{2}\left(k_1\left(\left.T\right|_{t+\Delta\,t}\right)+k_2\left(\left.T\right|_{t+\Delta\,t}\right)\right).$

Trapezoidal rule

- The integral on the right hand size can be approximated by the trapezoidal rule as follows:

$$\Delta\,\vec{Y} \approx \frac{\Delta\,t}{2}\left(\vec{G}\left(\vec{Y}\,(t)\,,\vec{Z}\,(t)\right)+\vec{G}\left(\vec{Y}\,(t+\Delta\,t)\,,\vec{Z}\,(t+\Delta\,t)\right)\right)$$

- Applied to the chemical reaction example, this scheme leads to the approximation of the increment $\Delta\,[A]$:

$$\Delta\,[A] = \frac{B}{1+K}$$

where:

  - $B = \Delta\,t\,\left(\langle k_2\rangle\,\left.[B]\right|_t - \langle k_1\rangle\,\left.[A]\right|_t\right)$ with

$$\begin{cases}\langle k_1\rangle = \dfrac{k_1\left(\left.T\right|_{t+\Delta\,t}\right)+k_1\left(\left.T\right|_t\right)}{2}\\ \langle k_2\rangle = \dfrac{k_2\left(\left.T\right|_{t+\Delta\,t}\right)+k_2\left(\left.T\right|_t\right)}{2}\end{cases}$$

  - $K = \dfrac{\Delta\,t}{2}\,\left(k_1\left(\left.T\right|_{t+\Delta\,t}\right)+k_2\left(\left.T\right|_{t+\Delta\,t}\right)\right)$.

- This closed-form expression of $\Delta\,[A]$ is due to the linear nature of the function $\vec{G}$.

Implementation using the trapezoidal rule

```
@DSL Model;
@Model ChemicalReaction2;
@Author Thomas Helfer;
@Date 09/07/2022;

@Import "ChemicalReaction-common.mfront";

@Function ChemicalReaction {
  constexpr auto zero = quantity<real, 0, 0, 0, 0, 0, 0, 1>{};
  const auto k1_bts = k01 * exp(-T_1 / Ta1);
  const auto k1_ets = k01 * exp(-T / Ta1);
  const auto k2_bts = k02 * exp(-T_1 / Ta2);
  const auto k2_ets = k02 * exp(-T / Ta2);
  const auto mean_k1 = (k1_bts + k1_ets) / 2;
  const auto mean_k2 = (k2_bts + k2_ets) / 2;
  const auto B = dt * (mean_k2 * cb_1 - mean_k1 * ca_1);
  const auto K = dt * (k1_ets + k2_ets) / 2;
  ca = ca_1 + B / (1 + K);
  cb = ca_1 + cb_1 - ca;
  if (cb < zero) {
    cb = zero;
    ca = ca_1 + cb_1;
  }
  if (ca < zero) {
    ca = zero;
    cb = ca_1 + cb_1;
  }
}
```

The generalized midpoint rule

- The general ordinary differential equation can be integrated over the time step using the generalized midpoint rule as follows:

$$\Delta\,\vec{Y} \approx \Delta\,t\,\vec{G}\left(\left.\vec{Y}\right|_{t+\theta\,\Delta\,t},\left.\vec{Z}\right|_{t+\theta\,\Delta\,t}\right)$$

where θ is a numerical parameter between 0 and 1.

The generalized midpoint rule

- The general ordinary differential equation can be integrated over the time step using the generalized midpoint rule as follows:

$$\Delta\,\vec{Y} \approx \Delta\,t\,\vec{G}\left(\left.\vec{Y}\right|_{t+\theta\,\Delta\,t},\left.\vec{Z}\right|_{t+\theta\,\Delta\,t}\right)$$

  where θ is a numerical parameter between 0 and 1.

- Applied to the chemical reaction example, this scheme leads to the approximation of the increment $\Delta\,[A]$:

$$\Delta\,[A] = \frac{B}{1+\theta\,K}$$

  with $\begin{cases} B = \Delta\,t\,\left(k_2\left(\left.T\right|_{t+\theta\,\Delta\,t}\right)\,\left.[B]\right|_{t} - k_1\left(\left.T\right|_{t+\theta\,\Delta\,t}\right)\,\left.[A]\right|_{t}\right) \\ K = \Delta\,t\;k_1\left(\left.T\right|_{t+\theta\,\Delta\,t}\right) \end{cases}$

Implementation using the midpoint rule

```
@DSL Model;
@Model ChemicalReaction3;
@Author Thomas Helfer;
@Date 09/07/2022;

@Import "ChemicalReaction−common.mfront";

//! numerical parameter of the generalized mid−point rule
@Parameter real theta = 0.5;
theta.setEntryName("Theta");

@Function ChemicalReaction {
  constexpr auto zero = quantity<real, 0, 0, 0, 0, 0, 0, 1>{};
  const auto T_mts = T_1 * (1 − theta) + theta * T;
  const auto k1_mts = k01 * exp(−T_mts / Ta1);
  const auto k2_mts = k02 * exp(−T_mts / Ta2);
  const auto B = dt * (k2_mts * cb_1 − k1_mts * ca_1);
  const auto K = dt * (k1_mts + k2_mts);
  ca = ca_1 + B / (1 + K * theta);
  cb = ca_1 + cb_1 − ca;
  // imposing positivity of the molar concentrations
  if (cb < zero){
    cb = zero;
    ca = ca_1 + cb_1;
  }
  if (ca < zero){
    ca = zero;
    cb = ca_1 + cb_1;
  }
}
```

A third scheme

- If the temperature increment is assumed small over a time step, an interesting scheme is to resuse the first one and evaluate the rate coefficients k_1 and k_2 at the middle of the time step $t + \dfrac{\Delta\, t}{2}$.
- The advantage of this implementation is that the exact solution is retrieved if the temperature is constant.

Implementation of the third scheme

```
@DSL Model;
@Model ChemicalReaction4;
@Author Thomas Helfer;
@Date 09/07/2022;

@Import "ChemicalReaction−common.mfront";

@Function ChemicalReaction {
  const auto T_mts = (T_1 + T) / 2;
  const auto k1_mts = k01 * exp(−T_mts / Ta1);
  const auto k2_mts = k02 * exp(−T_mts / Ta2);
  const auto B = k2_mts * (ca_1 + cb_1);
  const auto K = k1_mts + k2_mts;
  const auto e = exp(−K * dt);
  ca = ca_1 * e + (B / K) * (1 − e);
  cb = ca_1 + cb_1 − ca;
}
```

Tests

```
@Author Thomas Helfer;
@Date   09/08/2022;
@Model 'src/libModel.so' 'ChemicalReaction2';

@Real 'B0' 0.1;

@StateVariable 'MolarConcentrationOfSpeciesB' 'B0';

@Real 'T0' 700;
@Real 'T1' 400;
@Real 'tau0' 30;
@ExternalStateVariable<function> 'Temperature' 'T0 + T1 * sin(t/tau0)' ;

@Times {
  0, 200 in 100, 720 in 20
};
```

- The temperature is a periodic function of time with a period shorter than the relaxation time of the chemical reaction.

Outline

5. The `DefaultModel` DSL

First implementation

```
@DSL DefaultModel;
@Model ChemicalReaction5;
@Author Thomas Helfer;
@Date 09/07/2022;
@UseQt true;
@UnitSystem SI;

//! molar concentration of species A
@AuxiliaryStateVariable quantity<real, 0, 0, 0, 0, 0, 0, 1> ca;
ca.setEntryName("MolarConcentrationOfSpeciesA");
//! molar concentration of species B
@AuxiliaryStateVariable quantity<real, 0, 0, 0, 0, 0, 0, 1> cb;
cb.setEntryName("MolarConcentrationOfSpeciesB");

@Import "ChemicalReaction-parameters.mfront";

@Integrator{
  const auto T_mts = T + dT / 2;
  const auto k1_mts = k01 * exp(-T_mts / Ta1);
  const auto k2_mts = k02 * exp(-T_mts / Ta2);
  const auto B = k2_mts * (ca + cb);
  const auto K = k1_mts + k2_mts;
  const auto e = exp(-K * dt);
  const auto sum = ca + cb;
  ca = ca * e + (B / K) * (1 - e);
  cb = sum - ca;
}
```

Outline

6. The `RungeKuttaModel` DSL

Implementation

```
@DSL RungeKuttaModel;
@Model ChemicalReaction6;
@Author Thomas Helfer;
@Date 09/07/2022;
@UseQt true;
@UnitSystem SI;

@Algorithm rk54;
@Epsilon 1e−14;

//! molar concentration of species A
@StateVariable quantity<real, 0, 0, 0, 0, 0, 0, 1> ca;
ca.setEntryName("MolarConcentrationOfSpeciesA");
//! molar concentration of species B
@AuxiliaryStateVariable quantity<real, 0, 0, 0, 0, 0, 0, 1> cb;
cb.setEntryName("MolarConcentrationOfSpeciesB");

@Import "ChemicalReaction−parameters.mfront";

//! sum of the molar concentrations of species A and B
@LocalVariable quantity<real, 0, 0, 0, 0, 0, 0, 1> sum;

@InitLocalVariables {
  sum = ca + cb;
}

@Derivative{
  const auto k1 = k01 * exp(−T / Ta1);
  const auto k2 = k02 * exp(−T / Ta2);
  dca = k2 * sum − (k1 + k2) * ca;
}
```

Outline

7. The `ImplicitModel` DSL

Implementation

```
@DSL ImplicitModel;
@Model ChemicalReaction7;
@Author Thomas Helfer;
@Date 09 / 07 / 2022;
@UseQt true;
@UnitSystem SI;

@Epsilon 1e−14;
@Theta 0.5;

//! molar concentration of species A
@StateVariable quantity<real, 0, 0, 0, 0, 0, 0, 1> ca;
ca.setEntryName("MolarConcentrationOfSpeciesA");
//! molar concentration of species B
@AuxiliaryStateVariable quantity<real, 0, 0, 0, 0, 0, 0, 1> cb;
cb.setEntryName("MolarConcentrationOfSpeciesB");

@Import "ChemicalReaction−parameters.mfront";

//! sum of the molar concentrations of species A and B
@LocalVariable quantity<real, 0, 0, 0, 0, 0, 0, 1> sum;
/*!
 * reaction rate coefficient of the reaction transforming species A to species
 * B at the middle of the time step
 */
@LocalVariable frequency k1_mts;
/*!
 * reaction rate coefficient of the reaction transforming species B to species
 * A at the middle of the time step
 */
@LocalVariable frequency k2_mts;

@InitLocalVariables {
  const auto T_mts = T + dT / 2;
  k1_mts = k01 * exp(−T_mts / Ta1);
  k2_mts = k02 * exp(−T_mts / Ta2);
  sum = ca + cb;
}

@Integrator {
  const auto vca = k2_mts * sum − (k1_mts + k2_mts) * (ca + theta * dca);
  fca −= dt * vca;
  dfca_ddca += theta * dt * (k1_mts + k2_mts);
}

@UpdateAuxiliaryStateVariables {
  cb = sum − ca;
}
```

Outline

8. Conclusions

Reference files

- The examples of this tutorial can be downloaded here:
  - `https://github.com/thelfer/MFrontBookExamples`