



Point-wise models in MFront

DE LA RECHERCHE À L'INDUSTRIE

MFront training course

Septembre 2022

Thomas Helfer, Maxence Wangermez

Introduction

Commissariat à l'énergie atomique et aux énergies alternatives - www.cea.fr

- 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* $(y_j)_{j \in [1:n_y]}$ due to the evolution local thermodynamic environment described by a set of *external state variables* $(z_j)_{j \in [1:n_z]}$.

- ▶ 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* $(y_j)_{j \in [1:n_y]}$ due to the evolution local thermodynamic environment described by a set of *external state variables* $(z_j)_{j \in [1:n_z]}$.
- ▶ 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 (swelling due to solid or gaseous fission products, phase transition, chemical reactions) describes the local evolution of a set of *internal state variables* $(y_j)_{j \in [1:n_y]}$ due to the evolution local thermodynamic environment described by a set of *external state variables* $(z_j)_{j \in [1:n_z]}$.
- ▶ 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 simplest case of generic behaviours :
 - No thermodynamic force nor consistent tangent operator.

- ▶ 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* $(y_j)_{j \in [1:n_y]}$ due to the evolution local thermodynamic environment described by a set of *external state variables* $(z_j)_{j \in [1:n_z]}$.
- ▶ 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 simplest case of generic behaviours :
 - No thermodynamic force nor consistent tangent operator.
- ▶ Point-wise models can be used :
 - directly (in a PLEIADES application, in MTest or in MGIS) or used as a building block for models
 - as a building block for behaviours.

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

$$\vec{Y} = (y_1 \quad \dots \quad y_{n_y})^T$$

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

$$\vec{Y} = (y_1 \quad \dots \quad y_{n_y})^T$$

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

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

$$\vec{Z} = (z_1 \quad \dots \quad z_{n_z})^T$$

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

$$\vec{Y} = (y_1 \quad \dots \quad y_{n_y})^T$$

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

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

$$\vec{Z} = (z_1 \quad \dots \quad z_{n_z})^T$$

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

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

where τ denotes the time variable.

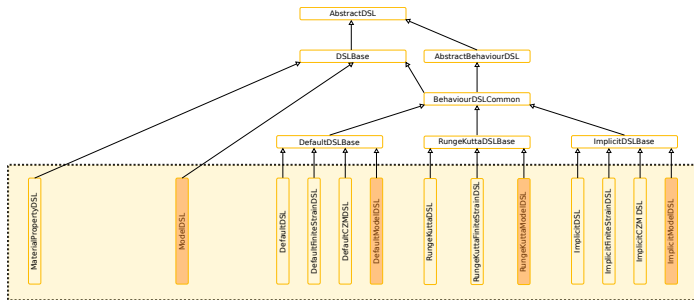
- ▶ Time is discretized in intervals named time steps.
 - t denotes the time at the beginning of the (considered) time step.
 - Δt is the time increment during the time step.
 - $t + \Delta t$ corresponds to the time at the end of the step.

- ▶ Time is discretized in intervals named time steps.
 - t denotes the time at the beginning of the (considered) time step.
 - Δt 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 $\vec{Z}|_t$ and their increments during the time step, denoted $\Delta \vec{Z}$, are known.
 - Since only $\vec{Z}|_t$ and $\Delta \vec{Z}$ are known, the evolution of \vec{Z} during the time step can only be described using a linear interpolation :

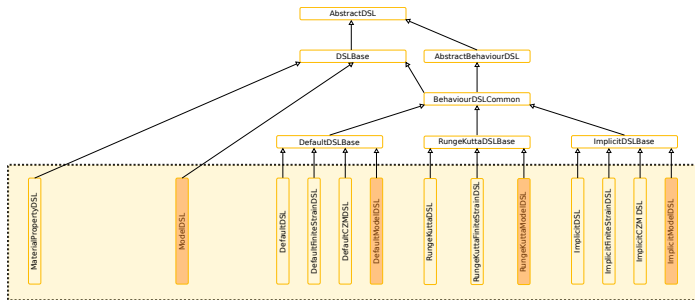
$$\vec{Z}|_{t+\theta \Delta t} = \vec{Z}|_t + \theta \Delta \vec{Z}$$

where θ is a number in the range $[0, 1]$

- This assumption generally implies that high order algorithms are generally useless.



- The Model DSL is historically the first DSL dedicated 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.

A simple 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 :



- 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 :



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

$$\begin{cases} \frac{d[A]}{d\tau} = k_2 [B](\tau) - k_1 [A](\tau) \\ \frac{d[B]}{d\tau} = k_1 [A](\tau) - k_2 [B](\tau) \end{cases}$$

- 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 :



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

$$\begin{cases} \frac{d[A]}{d\tau} = k_2 [B](\tau) - k_1 [A](\tau) \\ \frac{d[B]}{d\tau} = k_1 [A](\tau) - k_2 [B](\tau) \end{cases}$$

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

- 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 :



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

$$\begin{cases} \frac{d[A]}{d\tau} = k_2 [B](\tau) - k_1 [A](\tau) \\ \frac{d[B]}{d\tau} = k_1 [A](\tau) - k_2 [B](\tau) \end{cases}$$

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

$$\frac{d[A]}{d\tau} = k_2 ([A]|_t + [B]|_t) - (k_1 + k_2) [A](\tau)$$

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

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

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

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

with : $B = k_2 ([A]|_t + [B]|_t)$ and $K = k_1 + k_2$.

```
@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;
}
```

```
///! 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.

```
#!/ 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 $x|_{t+\Delta t}$, is available and associated with a variable named x .
 - 1 : $x|_t$ is associated with the variable x_1 and $x|_{t+\Delta t}$ is associated with the variable x .
 - higher values are hardly ever been used and highly discouraged.

```

//! 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 $x|_{t+\Delta t}$, is available and associated with a variable named x .
 - 1 : $x|_t$ is associated with the variable x_1 and $x|_{t+\Delta t}$ 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 :

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


```

//! 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 $x|_{t+\Delta t}$, is available and associated with a variable named x .
 - 1 : $x|_t$ is associated with the variable x_1 and $x|_{t+\Delta t}$ 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.

```
@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 practice. It is highly recommended to define only one function per model.
 - The '@Integrator' keyword has been introduced in Version 4.1 for consistency with behaviours and does not expect a function name;
- The implementation readily translates the analytical formula in C++ :

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

with : $B = k_2 ([A]_t + [B]_t)$ and $K = k_1 + k_2$.

```
// 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
```

```
// 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`

```
// 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.

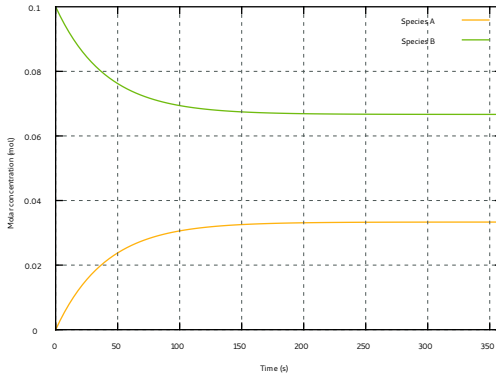


FIGURE – Evolution of the molar concentrations of species $[A]$ and $[B]$.

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

```
// 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.

```
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.


```
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

Reaction rate coefficients varying with temperature

Reaction rate coefficients as functions of the temperature

Material properties	Value
k_{10}	$0.018377505387559667 \text{ s}^{-1}$
k_{20}	$0.01013198112809354 \text{ s}^{-1}$
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 :

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

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

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

$$\Delta \vec{Y} = \int_t^{t+\theta \Delta t} \vec{G}(\vec{Y}(\tau), \vec{Z}(\tau)) d\tau$$

- ▶ 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

- 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.

- ▶ 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.

- ▶ 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.

- ▶ 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";
```



```
@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";
```

```
#!/ 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 = 0.01013198112809354;
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");
```

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

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

- The integral on the right hand side 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 :

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

- The integral on the right hand side 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 :

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

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

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

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

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

where θ is a numerical parameter between 0 and 1.

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

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

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


```

@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;
  }
}

```

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

```
@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;  
}
```

```
@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.

The DefaultModel DSL

```
@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;
}
```

- ▶ A priori time step scaling factor

- ▶ A priori time step scaling factor
- ▶ A posterior time step scaling factor

- Many models of the literature describing the oxidation of pipes made of Zircaloy alloys at constant temperature takes the following form :

$$l_0^2(\tau) = K(T) \tau$$

where l_0 is the oxidation length and τ the time from the beginning of the experiment.

- Many models of the literature describing the oxidation of pipes made of Zircaloy alloys at constant temperature takes the following form :

$$l_0^2(\tau) = K(T) \tau$$

where l_0 is the oxidation length and τ the time from the beginning of the experiment.

- The function $K(T)$ is generally chosen of the form :

$$K(T) = K_0 \exp\left(-\frac{E_a}{RT}\right)$$

where K_0 is characteristic of the pipe considered (material and size), E_a is an activation energy ($J.mol^{-1}$) and R is the perfect gas constant.

- Many models of the literature describing the oxidation of pipes made of Zircaloy alloys at constant temperature takes the following form :

$$l_0^2(\tau) = K(T) \tau$$

where l_0 is the oxidation length and τ the time from the beginning of the experiment.

- The function $K(T)$ is generally chosen of the form :

$$K(T) = K_0 \exp\left(-\frac{E_a}{RT}\right)$$

where K_0 is characteristic of the pipe considered (material and size), E_a is an activation energy ($J.mol^{-1}$) and R is the perfect gas constant.

- A model applicable in case of non constant temperature can be obtained by differentiating the previous equation with respect to time :

$$2 l_0 \frac{dl_0}{d\tau} = K(T)$$

- The previous Ordinary Differential Equation can be integrated over a time step $[t : t + \Delta t]$ as follows :

$$I_0|_{t+\Delta t}^2 - I_0|_t^2 = \int_t^{t+\Delta t} K(T(\tau)) \Delta \tau \approx K(T + \theta \Delta T) \Delta t,$$

- This leads to the following update scheme :

$$I_0|_{t+\Delta t} = \sqrt{I_0|_t^2 + K(T + \theta \Delta T) \Delta t}$$

The RungeKuttaModel DSL

- Runge-Kutta algorithms estimates the solution at the end of the time step by the weighted sum of approximations $\Delta \vec{Y}$ of the increment $\Delta \vec{Y}$

as follows :

$$\vec{Y}\Big|_{t+\Delta t} = \vec{Y}\Big|_t + \sum_{i=1}^n b_i \Delta \vec{Y}^{(i)}$$

- ▶ Runge-Kutta algorithms estimates the solution at the end of the time step by the weighted sum of approximations $\Delta \vec{Y}$ of the increment $\Delta \vec{Y}$

as follows :
$$\vec{Y}\Big|_{t+\Delta t} = \vec{Y}\Big|_t + \sum_{i=1}^n b_i \Delta \vec{Y}^{(i)}$$

- ▶ The approximations $\Delta \vec{Y}^{(i)}$ are given by successive evaluations of the rate function \vec{G} :

$$\begin{cases} \Delta \vec{Y}^{(1)} = \Delta t \vec{G} \left(\vec{Y}^{(1)}, \vec{Z}\Big|_t \right) & \text{with } \vec{Y}^{(1)} = \vec{Y}\Big|_t \\ \Delta \vec{Y}^{(2)} = \Delta t \vec{G} \left(\vec{Y}^{(2)}, \vec{Z}\Big|_{t+c_2 \Delta t} \right) & \text{with } \vec{Y}^{(2)} = \vec{Y}\Big|_t + a_{21} \Delta \vec{Y}^{(1)} \\ \vdots & \\ \Delta \vec{Y}^{(n)} = \Delta t \vec{G} \left(\vec{Y}^{(n)}, \vec{Z}\Big|_{t+c_n \Delta t} \right) & \text{with } \vec{Y}^{(n)} = \vec{Y}\Big|_t + a_{n1} \Delta \vec{Y}^{(1)} + \dots + a_{n,n-1} \Delta \vec{Y}_{n-1} \end{cases}$$

with $\vec{Z}\Big|_{t+c_j \Delta t} = \vec{Z}\Big|_t + c_j \Delta \vec{Z}$ being the estimates of \vec{Z} at time $t + c_j \Delta t$.

- ▶ Runge-Kutta algorithms estimates the solution at the end of the time step by the weighted sum of approximations $\Delta \vec{Y}$ of the increment $\Delta \vec{Y}$

as follows :
$$\vec{Y}\Big|_{t+\Delta t} = \vec{Y}\Big|_t + \sum_{i=1}^n b_i \Delta \vec{Y}^{(i)}$$

- ▶ The approximations $\Delta \vec{Y}^{(i)}$ are given by successive evaluations of the rate function \vec{G} :

$$\begin{cases} \Delta \vec{Y}^{(1)} = \Delta t \vec{G} \left(\vec{Y}^{(1)}, \vec{Z}\Big|_t \right) & \text{with } \vec{Y}^{(1)} = \vec{Y}\Big|_t \\ \Delta \vec{Y}^{(2)} = \Delta t \vec{G} \left(\vec{Y}^{(2)}, \vec{Z}\Big|_{t+c_2 \Delta t} \right) & \text{with } \vec{Y}^{(2)} = \vec{Y}\Big|_t + a_{21} \Delta \vec{Y}^{(1)} \\ \vdots & \\ \Delta \vec{Y}^{(n)} = \Delta t \vec{G} \left(\vec{Y}^{(n)}, \vec{Z}\Big|_{t+c_n \Delta t} \right) & \text{with } \vec{Y}^{(n)} = \vec{Y}\Big|_t + a_{n1} \Delta \vec{Y}^{(1)} + \dots + a_{n,n-1} \Delta \vec{Y}_{n-1} \end{cases}$$

with $\vec{Z}\Big|_{t+c_j \Delta t} = \vec{Z}\Big|_t + c_j \Delta \vec{Z}$ being the estimates of \vec{Z} at time $t + c_j \Delta t$.

- ▶ A particular algorithm is thus characterized by n , a_{ij} , b_i and c_i .

- The classical fourth order Runge-Kutta is based on the following scheme :

$$\vec{y}\big|_{t+\Delta t} = \vec{y}\big|_t + \frac{1}{6} \left(\Delta \vec{y}^{(1)} + 2 \Delta \vec{y}^{(2)} + 2 \Delta \vec{y}^{(3)} + \Delta \vec{y}^{(4)} \right)$$

with :

$$\left\{ \begin{array}{l} \Delta \vec{y}^{(1)} = \Delta t \vec{G} \left(\vec{y}^{(1)}, \vec{z}\big|_t \right) \\ \Delta \vec{y}^{(2)} = \Delta t \vec{G} \left(\vec{y}^{(2)}, \vec{z}\big|_t + \frac{\Delta \vec{z}}{2} \right) \\ \Delta \vec{y}^{(3)} = \Delta t \vec{G} \left(\vec{y}^{(3)}, \vec{z}\big|_t + \frac{\Delta \vec{z}}{2} \right) \\ \Delta \vec{y}_4 = \Delta t \vec{G} \left(\vec{y}^{(4)}, \vec{z}\big|_{t+\Delta t} \right) \end{array} \right. \quad \begin{array}{l} \text{with } \vec{y}^{(1)} = \vec{y}\big|_t \\ \text{with } \vec{y}^{(2)} = \vec{y}\big|_t + \frac{\Delta \vec{y}^{(1)}}{2} \\ \text{with } \vec{y}^{(3)} = \vec{y}\big|_t + \frac{\Delta \vec{y}^{(2)}}{2} \\ \text{with } \vec{y}^{(4)} = \vec{y}\big|_t + \Delta \vec{y}^{(3)} \end{array}$$

- ▶ Some algorithms allow to compute a lower order approximation for free.
- ▶ The difference between the higher order estimate and the lower order approximation is an estimate of the integration error :
 - automatic sub-stepping.

- ▶ Without substepping :
 - euler, rk2, rk4
- ▶ With automatic substepping :
 - rk42, rk54, rkCastem

- ▶ The `@Derivative` code block is used to compute the rate $\dot{\vec{Y}}$ of the state variables for a given estimate $\vec{Y}^{(i)}$:
- ▶ In the `@Derivative` code block, the following conventions holds :
 - `dy` denotes the value of the rate of the variable `y`.
 - `y` denotes the value of $y^{(i)}$.

```
@DSL RungeKuttaModel;
@Model ChemicalReaction6;
@UseQt true;
@UnitSystem SI;

@Algorithm rk54;
@Epsilon 1e-14;

@StateVariable quantity<real, 0, 0, 0, 0, 0, 0, 1> ca;
ca.setEntryName("MolarConcentrationOfSpeciesA");
@AuxiliaryStateVariable quantity<real, 0, 0, 0, 0, 0, 0, 1> cb;
cb.setEntryName("MolarConcentrationOfSpeciesB");

@Import "ChemicalReaction—parameters.mfront";

@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;
}

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

The ImplicitModel DSL

```
@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;
```

Applications

Conclusions

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



Thank you for your attention.

Time for discussion!

<https://tfel.sourceforge.net>

<https://www.researchgate.net/project/TFEL-MFront>

https://twitter.com/TFEL_MFront

<https://github.com/thelfer/>

tfel-contact@cea.fr

**The development of MFront is supported
financially by CEA, EDF and Framatome
in the framework of the PLEIADES project.**