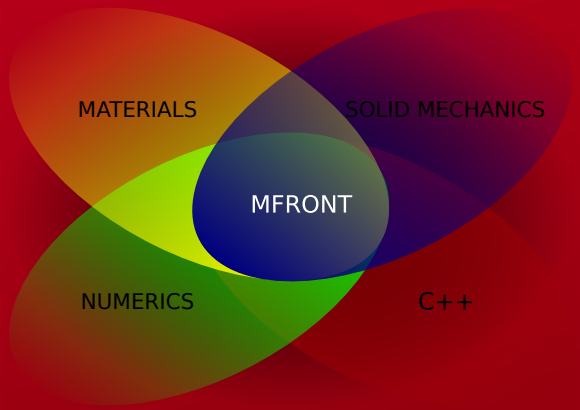
DE LA RECHERCHE À L'INDUSTRIE



www.cea.fr

TFEL 3.0

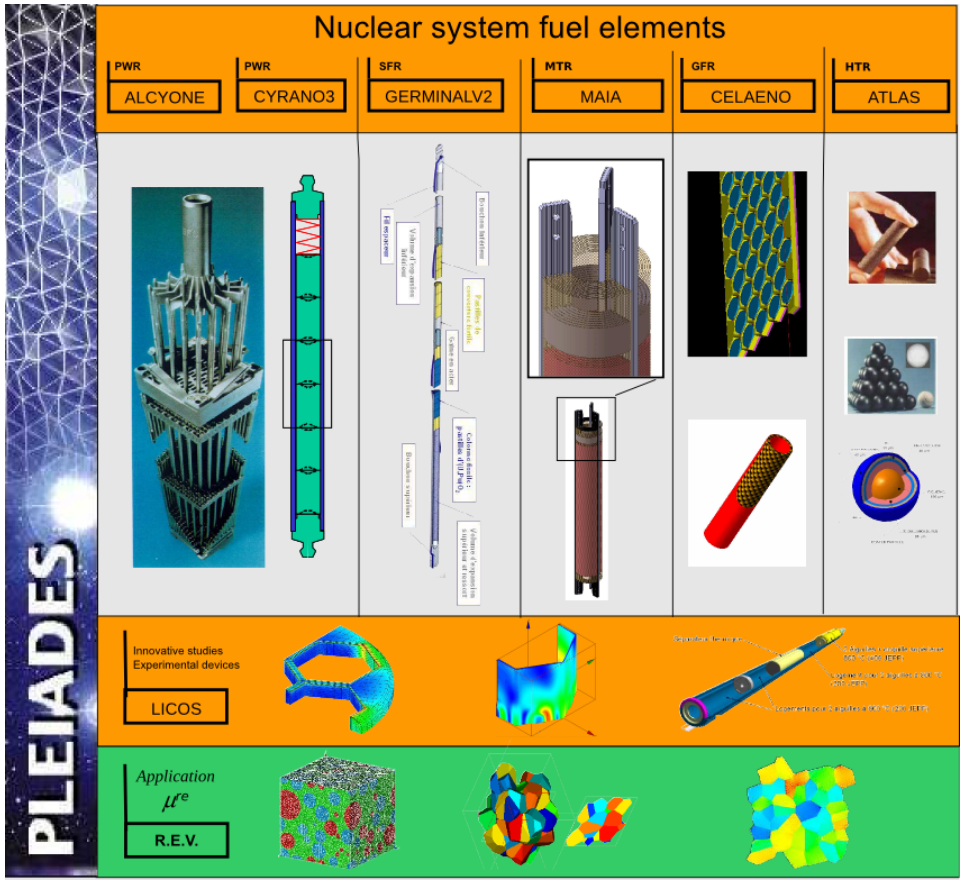
Second MFront User day | THOMAS HELFER, DAVID HABOUSSA, NICOLAS SELLENET, OLIVER JAMON, OLIVIER FANDEUR, THANG TRAN DANG, KARINE AUDIC

20 MAY 2016

Summary

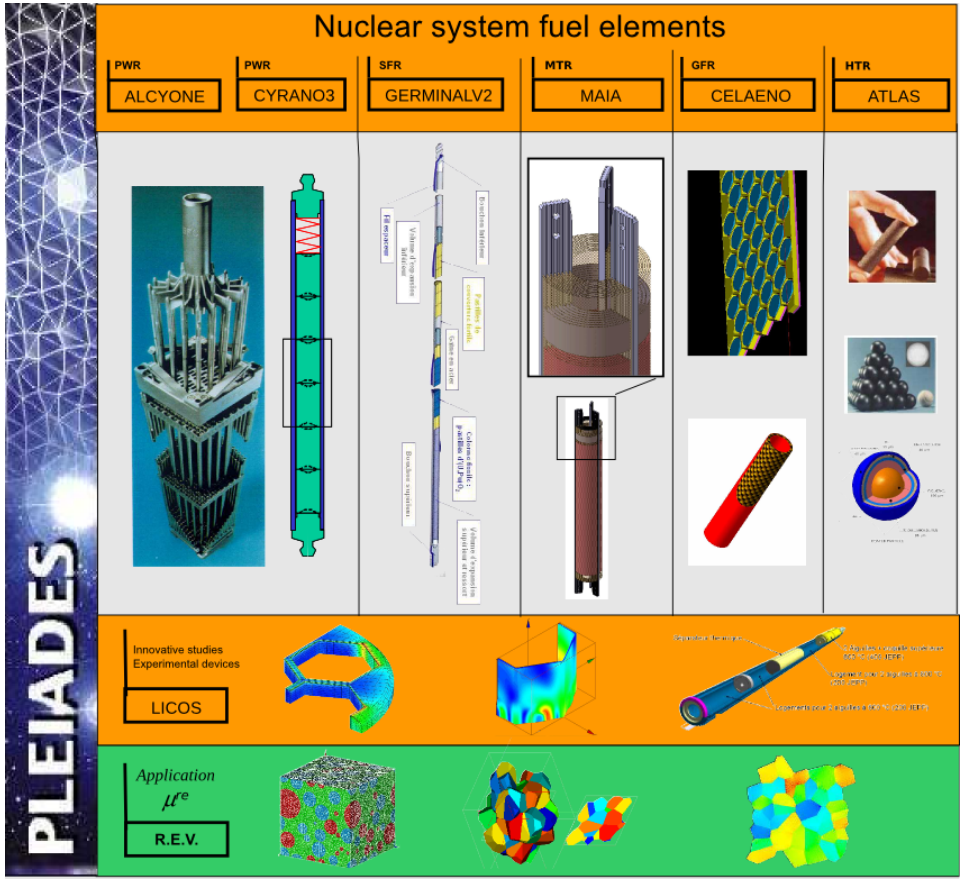
- Material knowledge management
- MFront improvements in TFEL 3.0.x
- Highlights
- Software developments
- Perspectives
- Improvements to the Cast3M interface
- Feed-back of MFront usage in `Code-Aster`
- The Europlexus Interface
- The Abaqus Interface
- Conclusion of the User day

The Pleiades platform



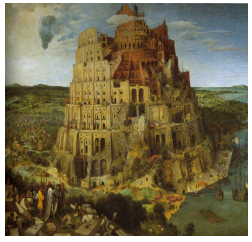
- A wide range of materials (ceramics, metals, composites)
- A wide range of mechanical phenomena and behaviours
  - creep, swelling, irradiation effects, phase transitions, etc..
- A wide range of mechanical loadings

The Pleiades platform



- A strong emphasis on :
  - Quality assurance (See the "Material knowledge management" Section)
  - Numerical performances
  - Portability between CEA and its partners (EDF and AREVA)

Interfaces



Plasticity.mfront

mfront --interface=umat → Fichiers C++ → (mfront --obuild, g++, clang, icpc) → Cast3M

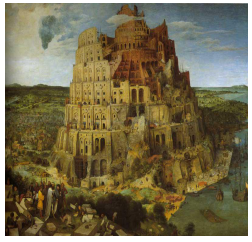
mfront --interface=aster → Fichiers C++ → (mfront --obuild, g++, clang, icpc) → Code_Aster

mfront --interface=zmat → Fichiers C++ → (mfront --obuild, g++, clang, icpc) → ZeBuLon

mfront --interface=.... → Fichiers C++ → (mfront --obuild, g++, clang, icpc) → ???

- A unique implementation, multiple targets :
  - Finite element solvers : `Cast3M`, `Code-Aster`, `ZeBuLoN`, **`Europlexus`**, **`Abaqus`** **(Standard and Explicit)**, `Calculix`
  - Fast Fourier Transform solvers : `TMFFTT`, `AMITEX_FFT`, `Craft` (?)

Interfaces



Plasticity.mfront

mfront --interface=umat → Fichiers C++ → (mfront --obuild, g++, clang, icpc) → Cast3M

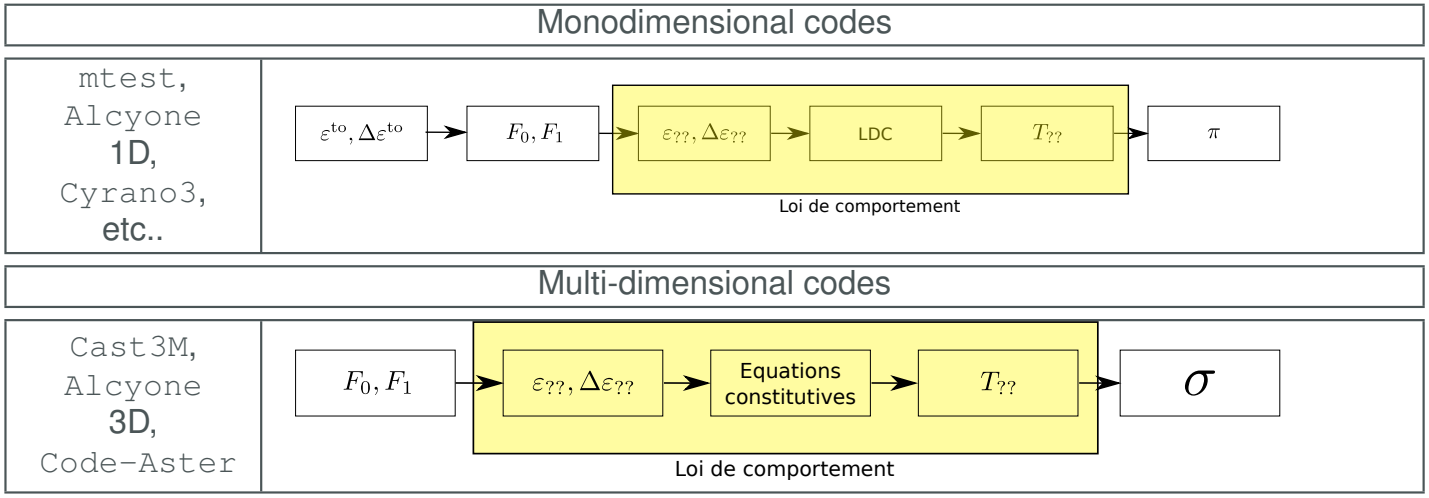
mfront --interface=aster → Fichiers C++ → (mfront --obuild, g++, clang, icpc) → Code_Aster

mfront --interface=zmat → Fichiers C++ → (mfront --obuild, g++, clang, icpc) → ZeBuLon

mfront --interface=.... → Fichiers C++ → (mfront --obuild, g++, clang, icpc) → ???

- A unique implementation, multiple targets :
  - Finite element solvers : `Cast3M`, `Code-Aster`, `ZeBuLoN`, **`Europlexus`**, **`Abaqus`** **(Standard and Explicit)**, `Calculix`
  - Fast Fourier Transform solvers : `TMFFTT`, `AMITEX_FFT`, `Craft` (?)

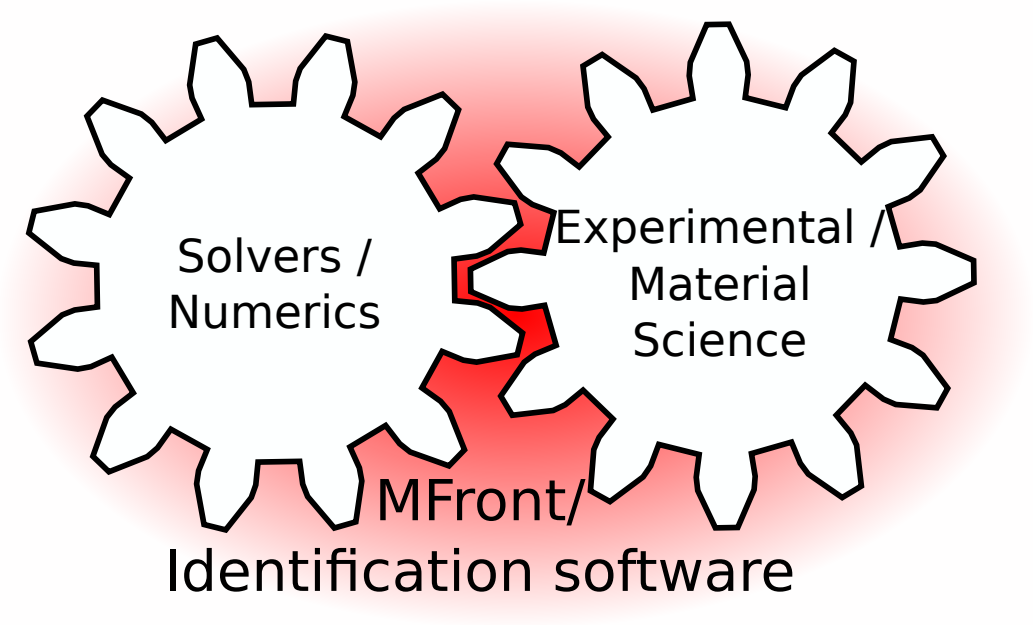
Interfaces



- A unique implementation, multiple targets :
  - Finite element solvers : `Cast3M`, `Code-Aster`, `ZeBuLoN`, **`Europlexus`**, **`Abaqus` (Standard and Explicit)**, `Calculix`
  - Fast Fourier Transform solvers : `TMFFTT`, `AMITEX_FFT`, `Craft` (?)
  - Interfaces take into account the specific aspects of solvers and/or the various strategies available

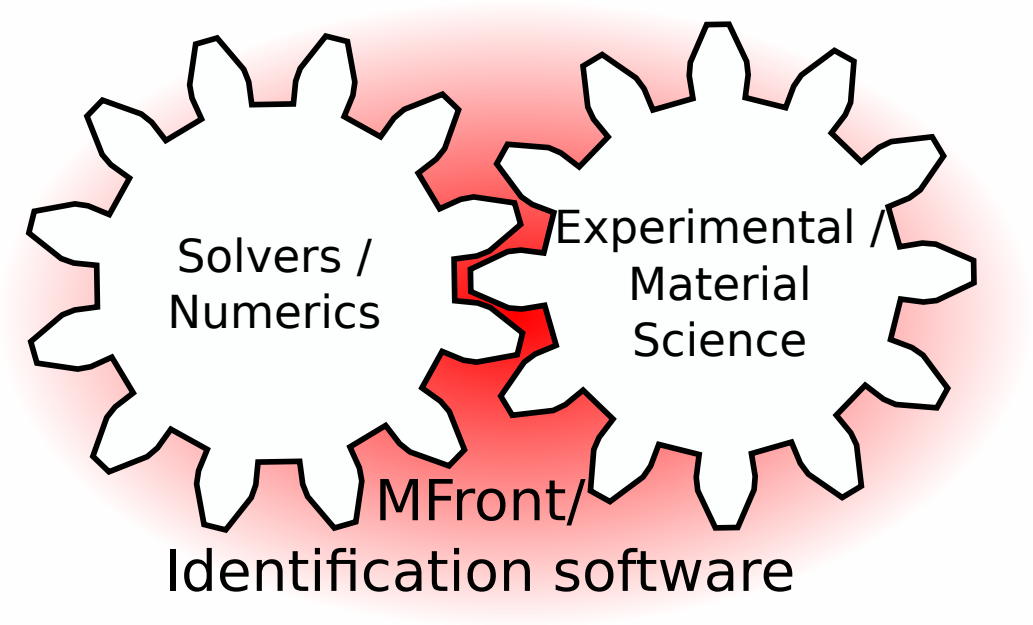
Material knowledge management : a consistent approach from experimentation to simulation

A much needed dialog



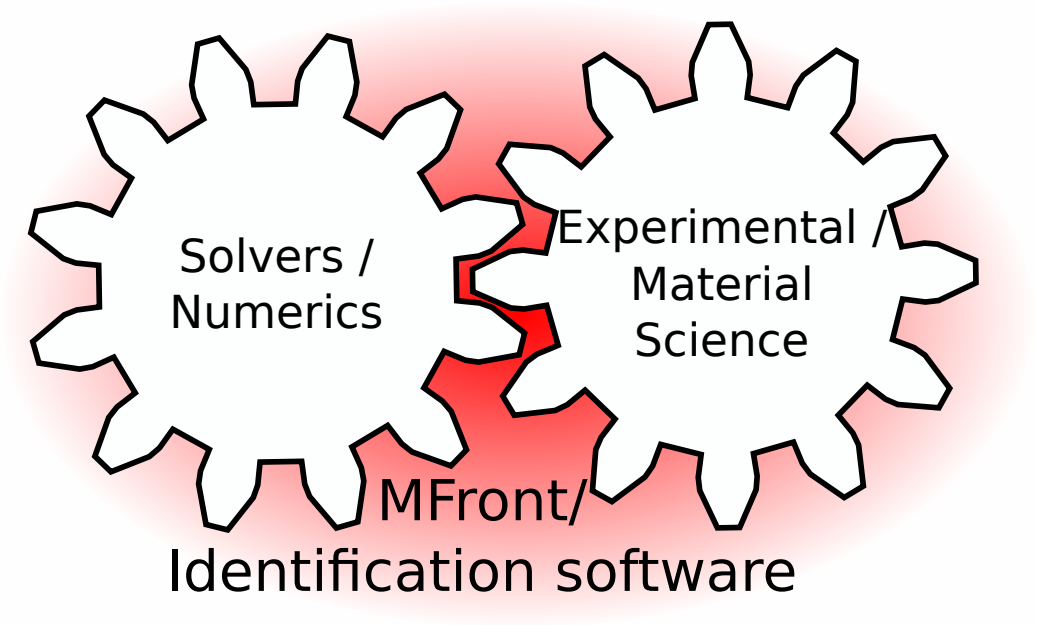
- Experimental people must be in charge of identifying mechanicals behaviours

A much needed dialog



- Experimental people must be in charge of identifying mechanicals behaviours
- Solver people must provide the tools (software and theoretical background) to make a proper identification
  - Especially when advanced topics are involved (finite strain, non local models, etc...)

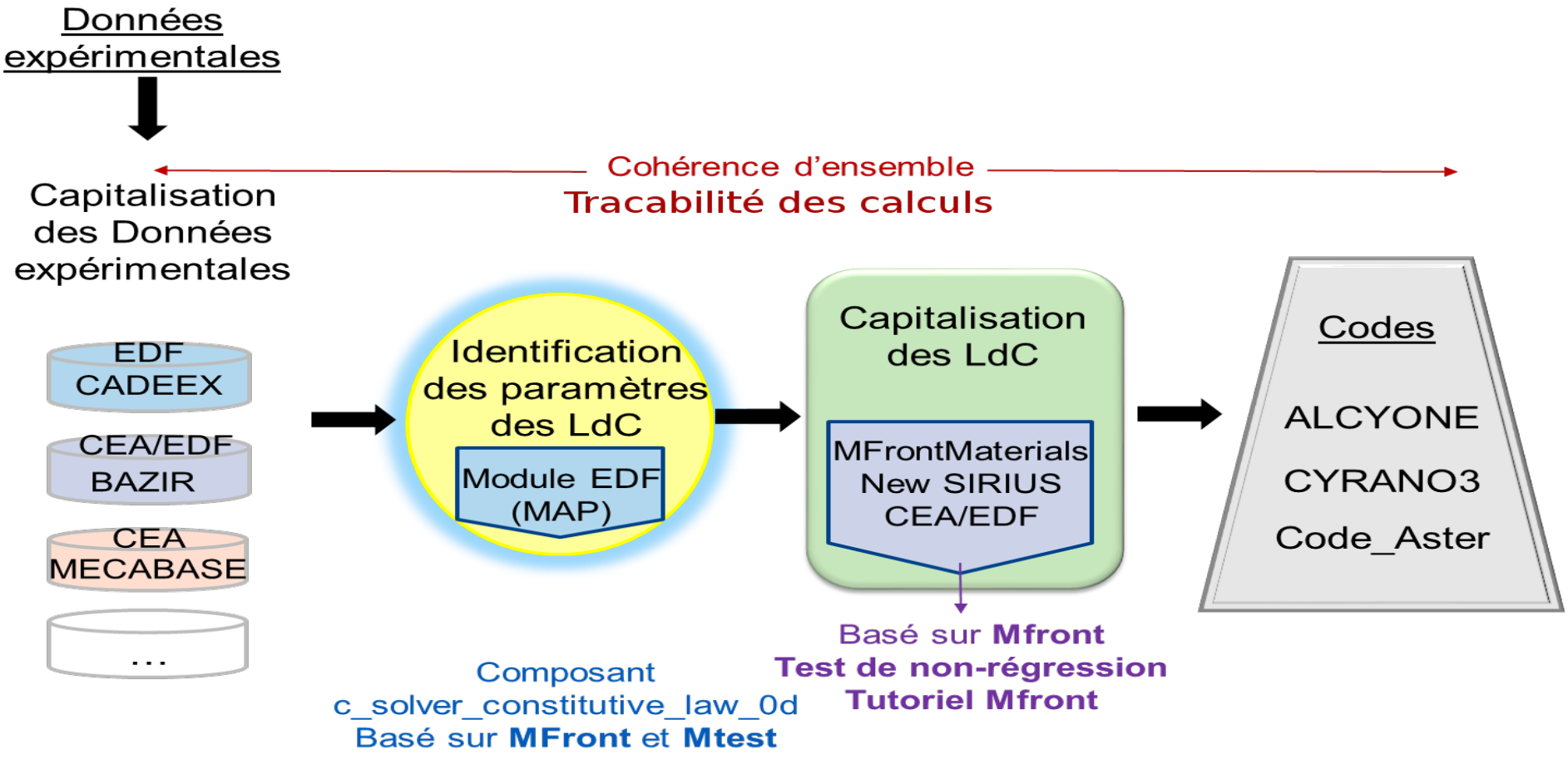
A much needed dialog



- Experimental people must be in charge of identifying mechanicals behaviours
- Solver people must provide the tools (software and theoretical background) to make a proper identification
  - Especially when advanced topics are involved (finite strain, non local models, etc...)

See also Rémi Munier's talk

Toward shared strategy and tools for material knowledge management ?



- Strong mutual technical interest !
- Still need to find a proper framework to work together...

MFront improvements in TFEL 3.0.x

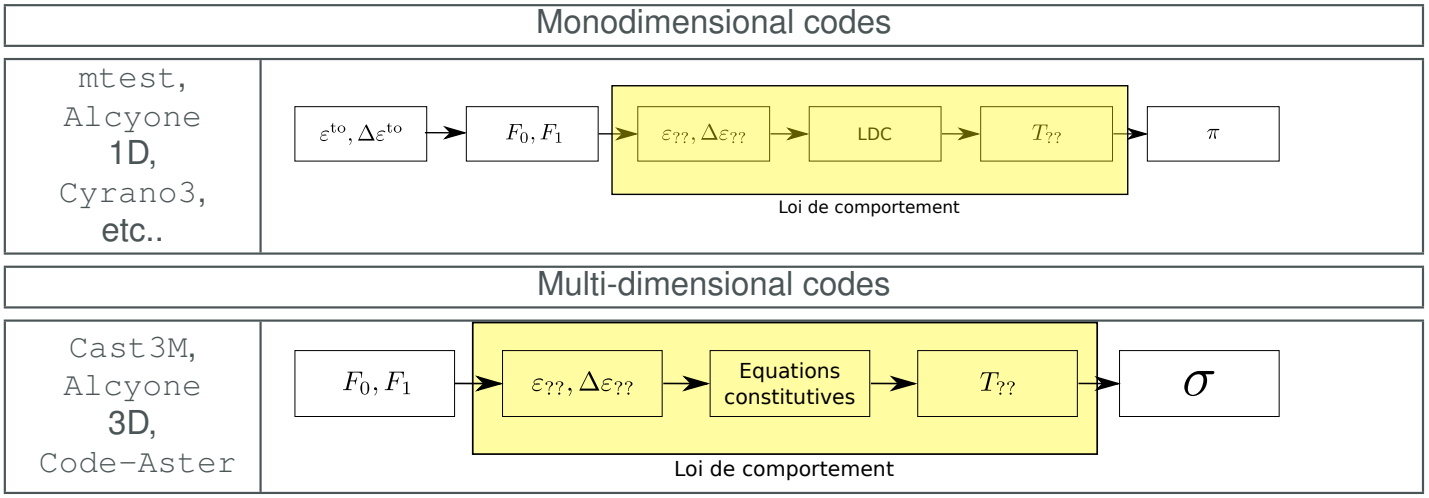
MFront goals : a reminder

- Writting mechanical behaviours shall be :
  - simple, numerically efficient, portable

MFront goals : a reminder

- Writting mechanical behaviours shall be :
  - simple, numerically efficient, portable
- From an AQ point of view, all physical information shall have an unique definition and a mechanical behaviour shall be self-consistent :
  - No `@MaterialProperty` !
    - Including elastic material properties

MFront goals : a reminder



- Writting mechanical behaviours shall be :
  - simple, numerically efficient, portable
- From an AQ point of view, all physical information shall have an unique definition and a mechanical behaviour shall be self-consistent :
  - No `@MaterialProperty` !
    - Including elastic material properties
  - No external handling of thermal expansion, swelling or axial growth !

Isotropic function of tensors and derivatives

$$\underline{\mathbf{s}} = \sum_{i=1}^{3} \lambda_i \, \underline{\mathbf{n}}_i \Rightarrow f\left(\underline{\mathbf{s}}\right) = \sum_{i=1}^{3} f\left(\lambda_i\right) \, \underline{\mathbf{n}}_i$$

- Three new functions are now available :
  - `computeIsotropicFunction`, `computeIsotropicFunctionDerivative` and `computeIsotropicFunctionAndDerivative`

Isotropic function of tensors and derivatives

$$\underline{\mathbf{s}} = \sum_{i=1}^{3} \lambda_i \, \underline{\mathbf{n}}_i \Rightarrow f\left(\underline{\mathbf{s}}\right) = \sum_{i=1}^{3} f\left(\lambda_i\right) \, \underline{\mathbf{n}}_i$$

- Three new functions are now available :
  - `computeIsotropicFunction`, `computeIsotropicFunctionDerivative` and `computeIsotropicFunctionAndDerivative`
- used to compute :
  - logarithmic strain and dual stresses

Isotropic function of tensors and derivatives

$$\underline{\mathbf{s}} = \sum_{i=1}^{3} \lambda_i \, \underline{\mathbf{n}}_i \Rightarrow f\left(\underline{\mathbf{s}}\right) = \sum_{i=1}^{3} f\left(\lambda_i\right) \, \underline{\mathbf{n}}_i$$

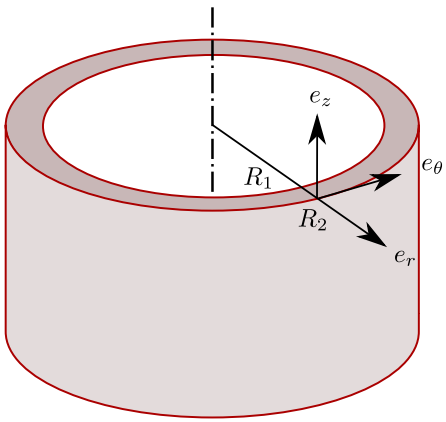
- Three new functions are now available :
  - `computeIsotropicFunction`, `computeIsotropicFunctionDerivative` and `computeIsotropicFunctionAndDerivative`
- used to compute :
  - logarithmic strain and dual stresses
  - energies in phase field approach of brittle fracture

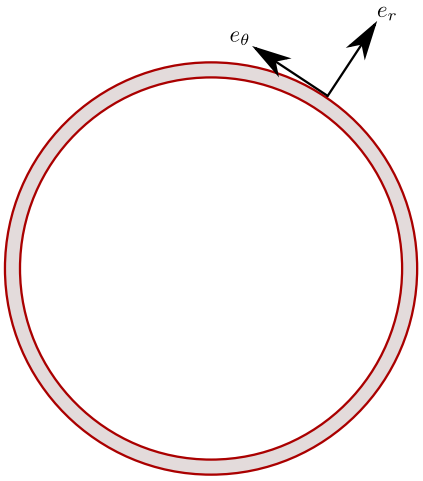
Isotropic function of tensors and derivatives

$$\underline{\mathbf{s}} = \sum_{i=1}^{3} \lambda_i \, \underline{\mathbf{n}}_i \Rightarrow f\left(\underline{\mathbf{s}}\right) = \sum_{i=1}^{3} f\left(\lambda_i\right) \, \underline{\mathbf{n}}_i$$

- Three new functions are now available :
  - `computeIsotropicFunction`, `computeIsotropicFunctionDerivative` and `computeIsotropicFunctionAndDerivative`
- used to compute :
  - logarithmic strain and dual stresses
  - energies in phase field approach of brittle fracture
  - coupling viscoelasticity and the Mazars damage behaviour, computing the consistent tangent operator

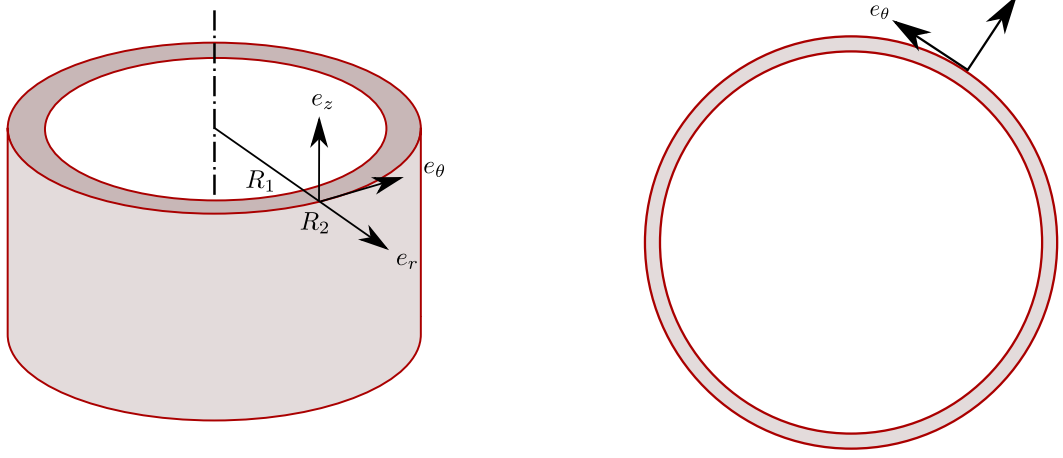
Orthotropic axes convention





- No solver allows a consistent definition of the the orthotropic axes for all modelling hypothesis

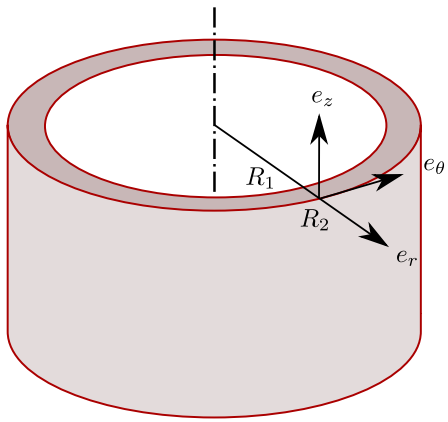
Orthotropic axes convention

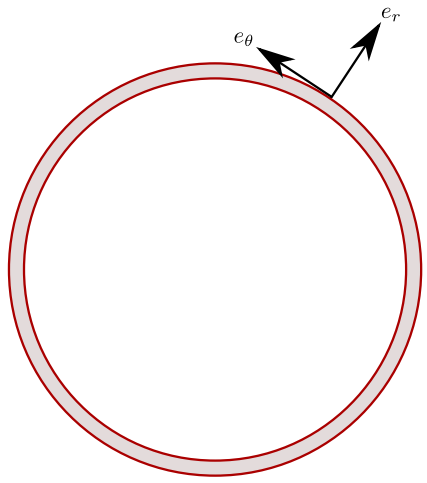


- No solver allows a consistent definition of the the orthotropic axes for all modelling hypothesis
- `MFront` introduces the notion of orthotropic axes convention which is chosen as an option of the `@OrthotropicBehaviour` keyword :

```
@OrthotropicBehaviour<Pipe>
```

Orthotropic axes convention



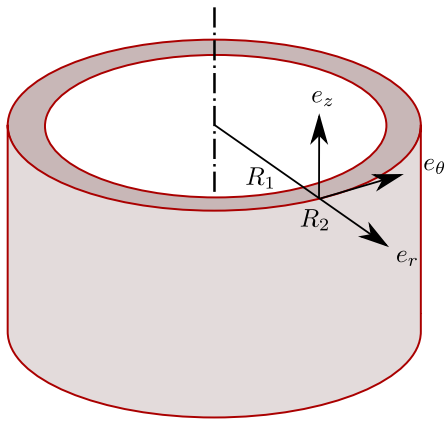


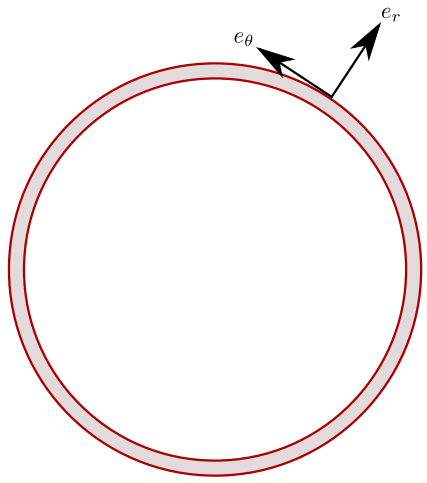
- No solver allows a consistent definition of the the orthotropic axes for all modelling hypothesis
- `MFront` introduces the notion of orthotropic axes convention which is chosen as an option of the `@OrthotropicBehaviour` keyword :

```
@OrthotropicBehaviour<Pipe>
```

- Two conventions are currently available : `Default` and `Pipe`

Orthotropic axes convention





- No solver allows a consistent definition of the the orthotropic axes for all modelling hypothesis
- `MFront` introduces the notion of orthotropic axes convention which is chosen as an option of the `@OrthotropicBehaviour` keyword :

```
@OrthotropicBehaviour<Pipe>
```

- Two conventions are currently available : `Default` and `Pipe`
- No solution for `ZeBuLoN` and `Abaqus` yet...

Orthotropic axes convention

```
@ComputeStiffnessTensor<UnAltered> {
  // YoungModulus1 YoungModulus2 YoungModulus3
  7.8e+10,2.64233e+11,3.32e+11,
  // PoissonRatio12 PoissonRatio23 PoissonRatio13
  0.13,0.24,0.18,
  // ShearModulus12 ShearModulus23 ShearModulus13
  4.8e+10,1.16418e+11,7.8e+10
};
```

- **The choice of the orthotropic axes convention affects the behaviour of many keywords** : `@ComputeThermalExpansion`, `@ComputeStiffnessTensor`, `@HillTensor`, `@Swelling`, `@AxialGrowth`, etc... See the associated documentation

Orthotropic axes convention

```
@ComputeStiffnessTensor<UnAltered> {
  // YoungModulus1 YoungModulus2 YoungModulus3
  7.8e+10,2.64233e+11,3.32e+11,
  // PoissonRatio12 PoissonRatio23 PoissonRatio13
  0.13,0.24,0.18,
  // ShearModulus12 ShearModulus23 ShearModulus13
  4.8e+10,1.16418e+11,7.8e+10
};
```

- **The choice of the orthotropic axes convention affects the behaviour of many keywords** : `@ComputeThermalExpansion`, `@ComputeStiffnessTensor`, `@HillTensor`, `@Swelling`, `@AxialGrowth`, etc... See the associated documentation
- For those keywords, the `Default` convention does nothing. **This is only valid if the material is symmetric with respect to the second and the third axes**.

Orthotropic axes convention

```
@ComputeStiffnessTensor<UnAltered> {
  // YoungModulus1 YoungModulus2 YoungModulus3
  7.8e+10,2.64233e+11,3.32e+11,
  // PoissonRatio12 PoissonRatio23 PoissonRatio13
  0.13,0.24,0.18,
  // ShearModulus12 ShearModulus23 ShearModulus13
  4.8e+10,1.16418e+11,7.8e+10
};
```

- **The choice of the orthotropic axes convention affects the behaviour of many keywords** : `@ComputeThermalExpansion`, `@ComputeStiffnessTensor`, `@HillTensor`, `@Swelling`, `@AxialGrowth`, etc... See the associated documentation
- For those keywords, the `Default` convention does nothing. **This is only valid if the material is symmetric with respect to the second and the third axes**.
- For those keywords, the `Pipe` convention exchanges the second and third axes in `PlaneStrain`, `PlaneStress`, `GeneralisedPlaneStress`.

New keywords

- `@ElasticMaterialProperties` : defines the elastic material properties which can be :
  - defined in an external `MFront` file
  - constant values (automatically turned into parameters)
  - For implicit schemes, `young`, `nu`, `lambda`, `mu`, `young_tdt`, `nu_tdt`, `lambda_tdt`, `mu_tdt` are automatically made available

New keywords

- `@ElasticMaterialProperties` : defines the elastic material properties which can be :
  - defined in an external `MFront` file
  - constant values (automatically turned into parameters)
  - For implicit schemes, `young`, `nu`, `lambda`, `mu`, `young_tdt`, `nu_tdt`, `lambda_tdt`, `mu_tdt` are automatically made available
- `@ComputeStiffnessTensor` : defines the elastic material properties and makes `MFront` compute the stiffness tensor :
  - for implicit schemes, `D` (at $t + \theta\ \mathrm{d}\,t$) and `D_tdt` (at $t + \mathrm{d}\,t$) are available
  - for Runge-Kutta schemes, `D` is available in `@Derivative` and its value depends on the current time

New keywords

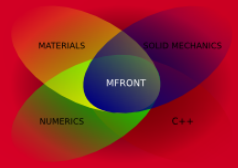
- `@ElasticMaterialProperties` : defines the elastic material properties which can be :
  - defined in an external `MFront` file
  - constant values (automatically turned into parameters)
  - For implicit schemes, `young`, `nu`, `lambda`, `mu`, `young_tdt`, `nu_tdt`, `lambda_tdt`, `mu_tdt` are automatically made available
- `@ComputeStiffnessTensor` : defines the elastic material properties and makes `MFront` compute the stiffness tensor :
  - for implicit schemes, `D` (at $t + \theta\, \mathrm{d}\, t$) and `D_tdt` (at $t + \mathrm{d}\, t$) are available
  - for Runge-Kutta schemes, `D` is available in `@Derivative` and its value depends on the current time
- `@ComputeThermalExpansion` defines the linear mean thermal expansion coefficien and makes `MFront` handle it properly (finite strain strategy is taken into account).

New keywords

- `@ElasticMaterialProperties` : defines the elastic material properties which can be :
  - defined in an external `MFront` file
  - constant values (automatically turned into parameters)
  - For implicit schemes, `young`, `nu`, `lambda`, `mu`, `young_tdt`, `nu_tdt`, `lambda_tdt`, `mu_tdt` are automatically made available
- `@ComputeStiffnessTensor` : defines the elastic material properties and makes `MFront` compute the stiffness tensor :
  - for implicit schemes, `D` (at $t + \theta\ \mathrm{d}\,t$) and `D_tdt` (at $t + \mathrm{d}\,t$) are available
  - for Runge-Kutta schemes, `D` is available in `@Derivative` and its value depends on the current time
- `@ComputeThermalExpansion` defines the linear mean thermal expansion coefficien and makes `MFront` handle it properly (finite strain strategy is taken into account).
- `@HillTensor` (to be implemented before 3.0.x release)

New keywords

- `@ElasticMaterialProperties` : defines the elastic material properties which can be :
  - defined in an external `MFront` file
  - constant values (automatically turned into parameters)
  - For implicit schemes, `young`, `nu`, `lambda`, `mu`, `young_tdt`, `nu_tdt`, `lambda_tdt`, `mu_tdt` are automatically made available
- `@ComputeStiffnessTensor` : defines the elastic material properties and makes `MFront` compute the stiffness tensor :
  - for implicit schemes, `D` (at $t + \theta\ \mathrm{d}\,t$) and `D_tdt` (at $t + \mathrm{d}\,t$) are available
  - for Runge-Kutta schemes, `D` is available in `@Derivative` and its value depends on the current time
- `@ComputeThermalExpansion` defines the linear mean thermal expansion coefficien and makes `MFront` handle it properly (finite strain strategy is taken into account).
- `@HillTensor` (to be implemented before 3.0.x release)
- `@Swelling`, `@AxialGrowth` (to be implemented before 3.0.x release)



Behaviours bricks : rationale

- Rationale : current implementation have to take care of **"details"** :

Behaviours bricks : rationale

```
@LocalVariable stress lambda;
@LocalVariable stress mu;

@InitLocalVariables{
  lambda = computeLambda(young,nu);
  mu = computeMu(young,nu);
} // end of @InitLocalVariables

@ComputeStress{
  sig = lambda*trace(eel)*Stensor::Id()+2*mu*eel;
} // end of @ComputeStresss
```

- Rationale : current implementation have to take care of **"details"** :
  - computation of the stress

Behaviours bricks : rationale

```
@TangentOperator{
  if ((smt==ELASTIC)||(smt==SECANTOPERATOR)||
      (smt==TANGENTOPERATOR)){
    computeAlteredElasticStiffness<hypothesis,Type>::exe(Dt,lambda,mu);
  } else if (smt==CONSISTENTTANGENTOPERATOR){
    StiffnessTensor Hooke;
    Stensor4 Je;
    computeElasticStiffness<N,Type>::exe(Hooke,lambda,mu);
    getPartialJacobianInvert(Je);
    Dt = Hooke*Je;
  } else {
    return false;
  }
}
```

- Rationale : current implementation have to take care of **"details"** :
  - computation of the stress
  - computation of the stiffness tangent operator

Behaviours bricks : rationale

```
@StateVariable<PlaneStress> real etozz;
PlaneStress::etozz.setGlossaryName("AxialStrain");

@Integrator<PlaneStress,Append,AtEnd>{
  // the plane stress equation is satisfied at the end of the time
  // step
  const stress szz = (lambda+2*mu)*(eel(2)+deel(2))+lambda*(eel(0)+deel(0)+eel(1)+deel(1));
  fetozz    = szz/young;
  // modification of the partition of strain
  feel(2) -= detozz;
  // jacobian
  dfeel_ddetozz(2)=-1;
  dfetozz_ddetozz  = real(0);
  dfetozz_ddeel(2) = (lambda+2*mu)/young;
  dfetozz_ddeel(0) = lambda/young;
  dfetozz_ddeel(1) = lambda/young;
}
```

- Rationale : current implementation have to take care of **"details"** :
  - computation of the stress
  - computation of the stiffness tangent operator
  - handling of the plane stress and axisymmetrical generalised plane stress modelling hypotheses

Behaviours bricks : rationale

- Rationale : current implementation have to take care of **"details"** :
  - computation of the stress
  - computation of the stiffness tangent operator
  - handling of the plane stress and axisymmetrical generalised plane stress modelling hypotheses
- Example of the **ImplicitNorton** behaviour :
  - A total of 130 lines
  - *100 lines of noise*
  - *Only 30 lines of "true" physical content*
  - It gets even worse with real implementations when elastic properties evolve over the time step !

Behaviours bricks : rationale

- Rationale : current implementation have to take care of **"details"** :
  - computation of the stress
  - computation of the stiffness tangent operator
  - handling of the plane stress and axisymmetrical generalised plane stress modelling hypotheses
- Example of the **ImplicitNorton** behaviour :
  - A total of 130 lines
  - *100 lines of noise*
  - *Only 30 lines of "true" physical content*
  - It gets even worse with real implementations when elastic properties evolve over the time step !
- **Behaviour bricks are a pratical way to allow users to focus of the true physical content**

Behaviour bricks : example

```
@DSL Implicit;
@Behaviour Norton;
@ModellingHypotheses {".+"};

@Brick "StandardElasticity";

@StateVariable strain p;
p.setGlossaryName("EquivalentViscoplasticStrain");

@Integrator{
  const auto mu = computeMu(young,nu);
  const real A = 8.e-67;
  const real E = 8.2;
  const stress seq = sigmaeq(sig);
  const real tmp = A*pow(seq,E-1.);
  const real df_dseq = E*tmp;
  const auto iseq = (seq > 1.e-8*young) ? 1/seq : 0;
  const auto n = 1.5*deviator(sig)*iseq;
  feel += dp*n;
  fp   -= tmp*seq*dt;
  dfeel_ddeel += 2.*mu*theta*dp*iseq*(Stensor4::M()-(n^n));
  dfeel_ddp    = n;
  dfp_ddeel    = -2*mu*theta*df_dseq*dt*n;
}
```

- all the "noise" is handled by the `StandardElasticity` brick ;
- Only **21 lines** for the whole implementation !

The `StandardElasticity` brick

- The `StandardElasticity` brick is usable in `Implicit` schemes :
  - appropriate jacobian blocks are defined if needed.
- The `StandardElasticity` brick describes :
  - the Hooke Law for isotropic or orthotropic behaviours ;
  - elastic material properties defined through :
    - parameters or material properties ;
    - `@RequireStiffnessTensor`
    - `@ComputeStiffnessTensor`
    - `@ElasticMaterialProperties`
  - For isotropic beahviour, it automatically declares the elastic material properties if mandatory.
  - For isotropic beahviour, it automatically requests the stiffness tensor if mandatory.
  - handles the computation of the consistent tangent operator.
  - handles plane stress and axisymmetrical generalised plane stress hypotheses.

StandardElasticity::computeElasticPrediction

```
// computes 2*mu*deviator(eel+theta*deto)
s_el = deviator(computeElasticPrediction());
for(ushort i=0;i!=3;++i){
  s_el -= 2*C[i]*a[i]/3;
}
const auto seq_el = sigmaeq(s_el);
const real Rpel  = R_inf + (R_0-R_inf)*exp(-b*p) ;
bo = seq_el>Rpel; // true on plastic loading
```

- Most plastic behaviours use an elastic prediction of the stresses to see if a plastic flow occurs during the time step
  - The modelling hypothesis must be taken into account to do this appropriately
  - `MFront` syntax to do this is awful
- The standard elasticity brick defines the `computeElasticPrediction` method to do this

A Chaboche-like Behaviour

```
@Predictor{
  mu    = computeMu(young,nu);
  // computes 2*mu*deviator(eel+theta*deto)
  s_el = deviator(computeElasticPrediction());
  for(ushort i=0;i!=3;++i){
    s_el -= 2*C[i]*a[i]/3;
  }
  const auto seq_el = sigmaeq(s_el);
  const real Rpel  = R_inf + (R_0-R_inf)*exp(-b*p) ;
  bo = seq_el>Rpel; // true on plastic loading
  deel=deto;
}
```

- Predictor phase

A Chaboche-like Behaviour

```
@Integrator{
  if(bo){
    // du to numerical jacobian, dp can be negative
    const auto dp_ = max(dp,strain(0));
    // compute the B tensor
    StressStensor B = s_el;
    stress tmp = 2*mu*theta;
    for(ushort i=0;i!=3;++i){
      const auto Ceff = 2*C[i]/(3*(1+theta*g[i]*dp_));
      tmp += Ceff;
      B   += theta*dp_*Ceff*g[i]*a[i];
    }
    const auto seq = 3*(sqrt(2*(B|B)/3)-tmp*dp_)/2;
    if(seq<0){return false;};
    // compute n
    n = B/(2*max(seq,1.e-10*young)/3+tmp*dp_);
    // isotropic part
    const auto p_  = p +theta*max(dp_,strain(0)) ;
    const auto Rp_ = R_inf + (R_0-R_inf)*exp(-b*p_) ;
    // implicit system
    fp    = (seq-Rp_)/young;
    feel += dp_*n;
  }
}
```

- Integration

A Chaboche-like Behaviour

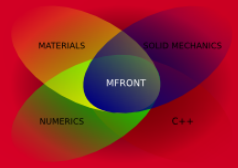
```
@UpdateAuxiliaryStateVariables {
  for(ushort i=0;i!=3;++i){
    a[i] += dp/(1+theta*dp*g[i])*(n-g[i]*a[i]);
  }
  ep+=deto-deel;
}
```

- Update auxiliary state variables

A Chaboche-like Behaviour

```
@UpdateAuxiliaryStateVariables {
  for(ushort i=0;i!=3;++i){
    a[i] += dp/(1+theta*dp*g[i])*(n-g[i]*a[i]);
  }
  ep+=deto-deel;
}
```

- Update auxiliary state variables
- **The whole implementation is** 78 **lines long**



Planed Behaviours bricks, perspectives

- **Bricks are easy to implement** :
  - About 600 lines of codes for the `StandardElasticityBrick` brick

Planed Behaviours bricks, perspectives

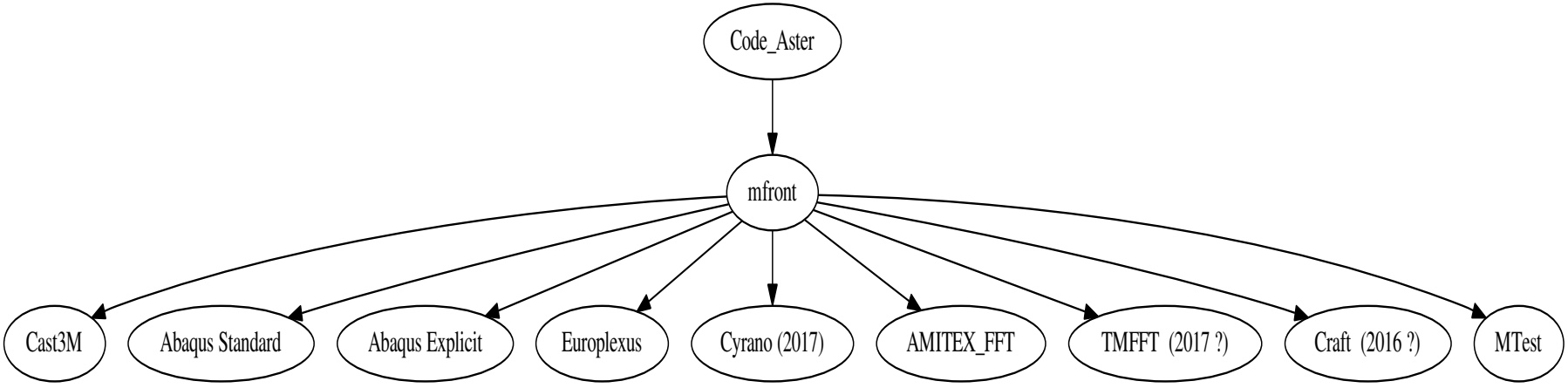
- **Bricks are easy to implement** :
  - About 600 lines of codes for the `StandardElasticityBrick` brick
- The following behaviours bricks are planed :
  - various damage models :
    - Marigo and Lorentz (Marigo+unilateral effects) damage models :
    - DDIF2 damage model
    - Mazars
  - Finite strain single crystal :
    - handles the $\tilde{\mathbf{F}}_e . \tilde{\mathbf{F}}_p$ decomposition
    - introduces a plastic strain for each gliding systems
    - computes of second PIOLA-KIRCHHOFF stress in the intermediate configuration
    - computes of the MANDEL stress in the reference configuration
    - computes of the consistent tangent operator
    - **the user "only" have to specify the plastic strain and state variables evolution !**

Planed Behaviours bricks, perspectives

- **Bricks are easy to implement** :
  - About 600 lines of codes for the `StandardElasticityBrick` brick
- The following behaviours bricks are planed :
  - various damage models :
    - Marigo and Lorentz (Marigo+unilateral effects) damage models :
    - DDIF2 damage model
    - Mazars
  - Finite strain single crystal :
    - handles the $\tilde{\mathbf{F}}_e . \tilde{\mathbf{F}}_p$ decomposition
    - introduces a plastic strain for each gliding systems
    - computes of second PIOLA-KIRCHHOFF stress in the intermediate configuration
    - computes of the MANDEL stress in the reference configuration
    - computes of the consistent tangent operator
    - **the user "only" have to specify the plastic strain and state variables evolution !**
- Future versions of `MFront` may include some standard models :
  - Chaboche (plasticity or viscoplasticity), GTN, etc..

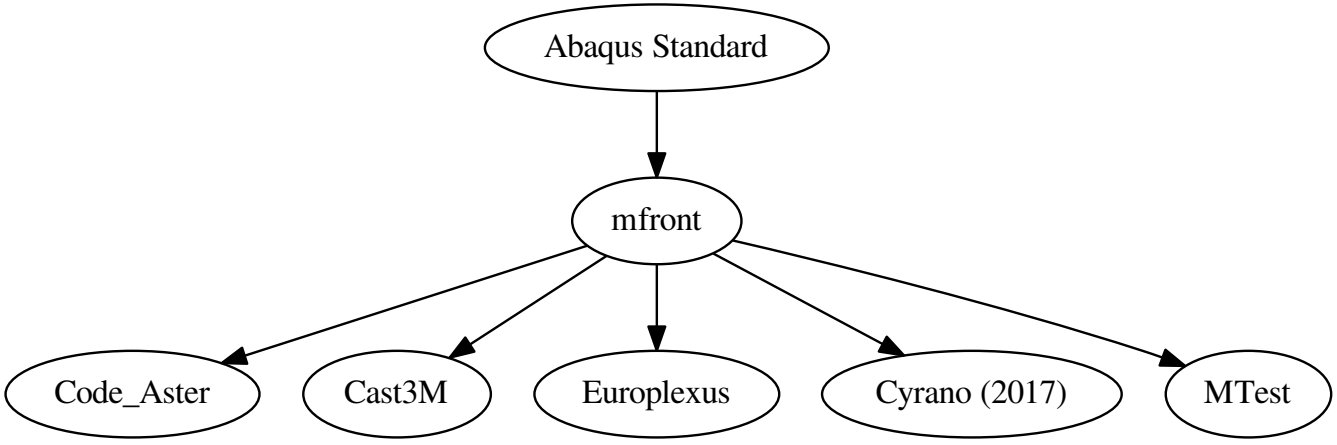
Highlights

Technology transfer



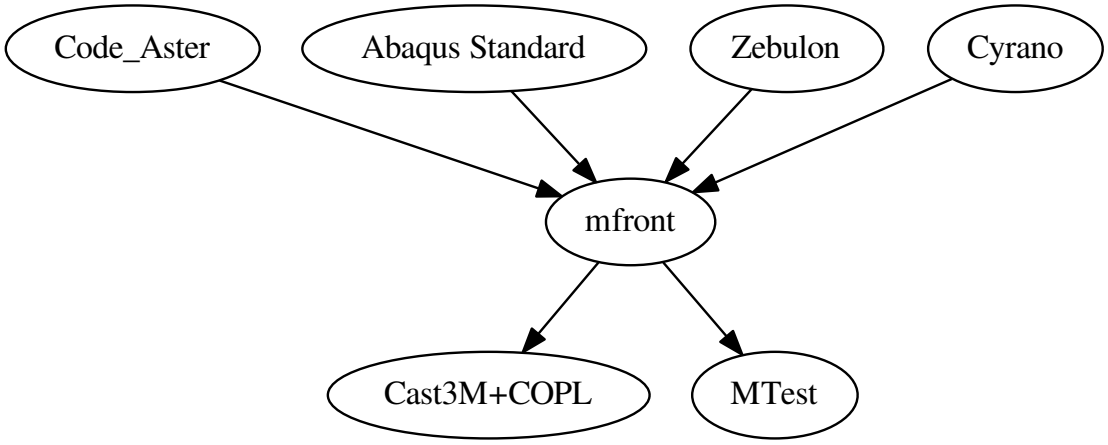
- Miehe, Apel, Lambrecht logarithmic strain framework

Technology transfer



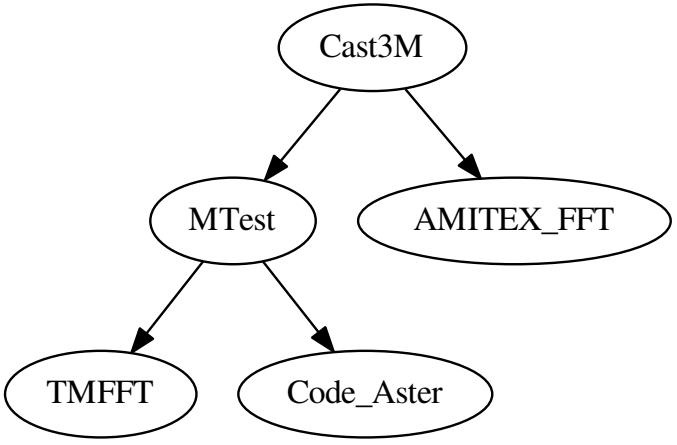
- Miehe, Apel, Lambrecht logarithmic strain framework
- Time step increase/decrease from the behaviour

Technology transfer



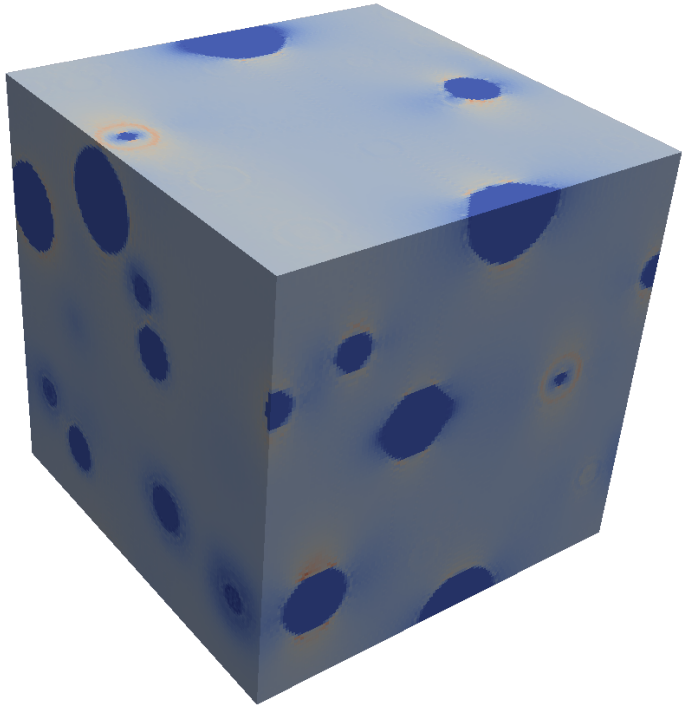
- Miehe, Apel, Lambrecht logarithmic strain framework
- Time step increase/decrease from the behaviour
- Consistent tangent operator

Technology transfer



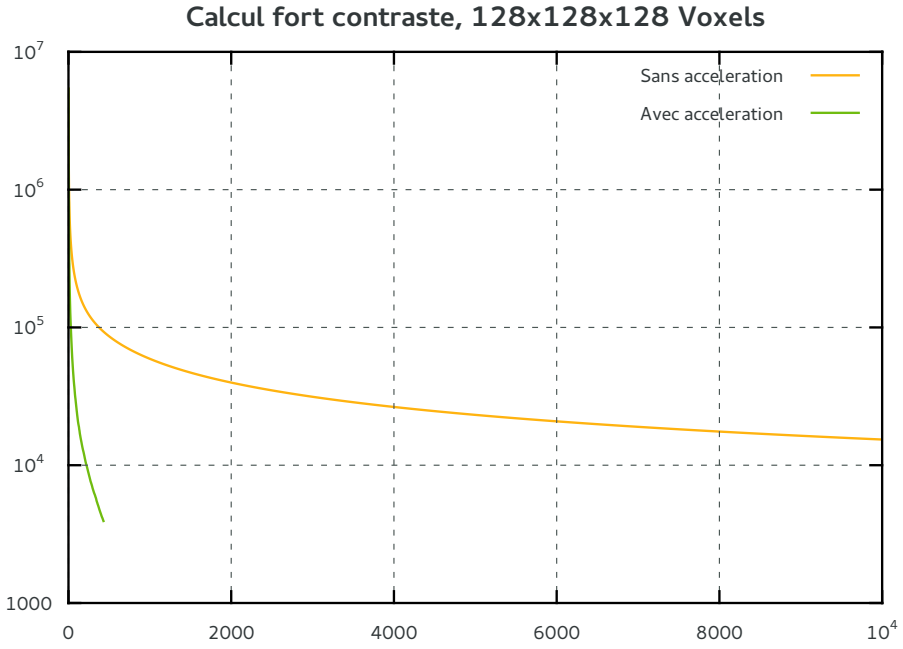
- Miehe, Apel, Lambrecht logarithmic strain framework
- Time step increase/decrease from the behaviour
- Consistent tangent operator
- Acceleration algorithms. See next slide.

Anderson based Acceleration algorithms



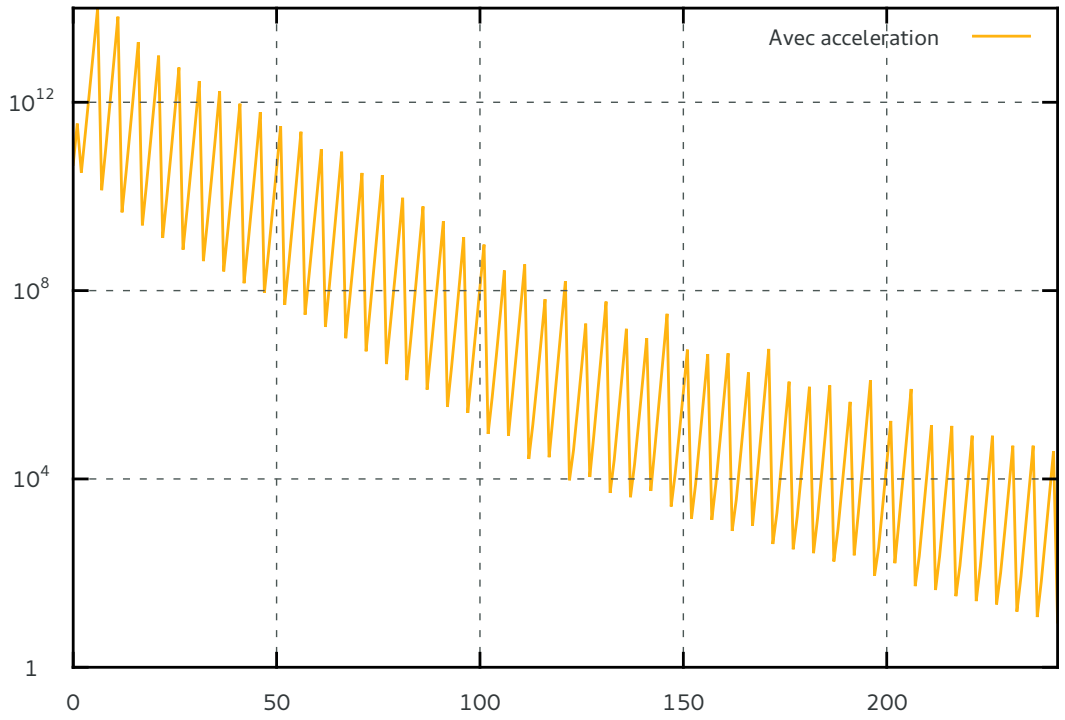
- Equilibrium is ensured through a fixed point algorithm
  - slow convergence similar to FE solver with elastic stiffness matrix
- Results on a case of study extracted from Coralie Esnoul PhD using TMFFT 6 10^6 *ddl*

Anderson based Acceleration algorithms



- Equilibrium is ensured through a fixed point algorithm
  - slow convergence similar to FE solver with elastic stiffness matrix
- Results on a case of study extracted from Coralie Esnoul PhD using TMFFT 6 10^6 *ddl*

Anderson based Acceleration algorithms



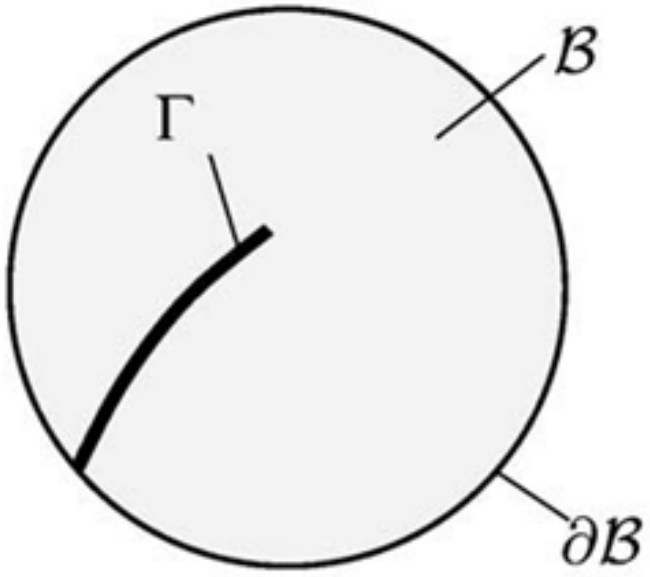
- Equilibrium is ensured through a fixed point algorithm
  - slow convergence similar to FE solver with elastic stiffness matrix
- Periodic application of the anderson algorithm on a diverging case

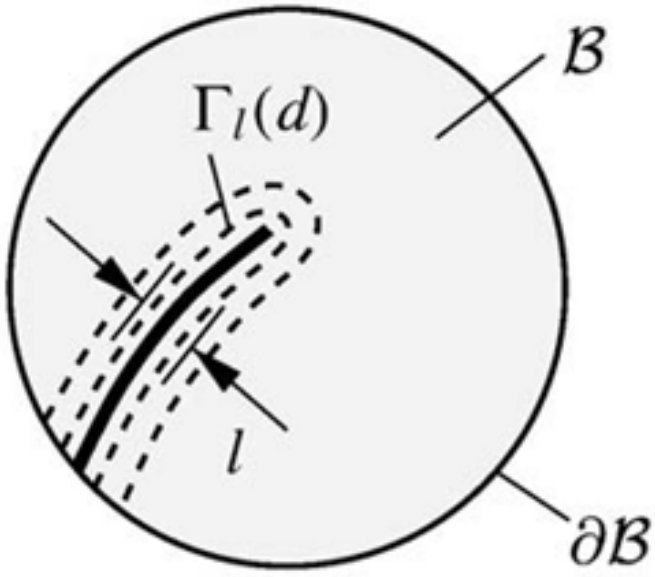
Pipe Modelling with MTest

```
@PredictionPolicy 'LinearPrediction';
@InnerRadius '8.346e-3/2';
@OuterRadius '9.504e-3/2';
@NumberOfElements 5;
@UseCastemAccelerationAlgorithm true;

@Times<data> @data@ using 1;
@InnerPressureEvolution<data> @data@ using 1:'$3*1.e5';

@Behaviour<castem> @library@ @behaviour@;
@MaterialProperty<constant> 'YoungModulus' 70e9;
@MaterialProperty<constant> 'PoissonRatio' 0.372;
@MaterialProperty<constant> 'ThermalExpansion' 4.e-6;
@ExternalStateVariable<data> 'Temperature' @data@ using 1:'$2+273.15';

@Profile @output@ {'SRR','STT','SZZ','EquivalentViscoplasticStrain',
    'BetaPhaseFraction'};
```

- Extension of MTest to support pipes modelling :
  - Monodimensional modelling
  - Proper finite strain modelling in the reference configuration
  - Various radial loading : imposed internal pressure, imposed external pressure, imposed outer radius evolution
  - Various axial loading : end cap effect, imposed axial strain, imposed axial force

Phase field approach of cement brittle failure





- Post-doc of Tran Thang Dang under the supervision of B. Bary (CEA/DEN/DPC/SECR/LECBA)
- Initial formulation by C. Miehe, based on Marigo/Bourdin approach

Phase field approach of cement brittle failure

Déformations et projecteurs

$\boldsymbol{\varepsilon} = \boldsymbol{\varepsilon}^+ + \boldsymbol{\varepsilon}^- = \boldsymbol{P}^+ : \boldsymbol{\varepsilon} + \boldsymbol{P}^- : \boldsymbol{\varepsilon}$

$\boldsymbol{P}^+ = \dfrac{\partial \boldsymbol{\varepsilon}^+}{\partial \boldsymbol{\varepsilon}}, \ \boldsymbol{P}^- = \boldsymbol{I} - \boldsymbol{P}^+$

Energie **SANS** endommagement

$W_u = \psi_0^+ + \psi_0^-$

$\psi_0^+ = \lambda \langle tr[\boldsymbol{\varepsilon}] \rangle_+^2 + \mu tr[\boldsymbol{\varepsilon}_+^2]$

$\psi_0^- = \lambda \langle tr[\boldsymbol{\varepsilon}] \rangle_-^2 + \mu tr[\boldsymbol{\varepsilon}_-^2]$

$m(d) = (1-d)^2$

$k \ll 1$

g_c : critère de Griffith

$\gamma(d)$: densité surfacique associée à la fissure

Energie **AVEC** endommagement

$W = \underbrace{\psi_0^+ [m(d) + k] + \psi_0^-}_{\text{volumique}} + \underbrace{g_c \, \gamma(d)}_{\text{surfacique}}$

$\dot{\Gamma}_l \geq 0 \ \Rightarrow \ \Phi = \boldsymbol{\sigma} : \dot{\boldsymbol{\varepsilon}} - \dot{W} \geq 0 \ \Rightarrow \ \dfrac{\partial W}{\partial d} = 2(1-d)\psi_0^+ + g_c \dfrac{\partial \gamma}{\partial d} = 0$

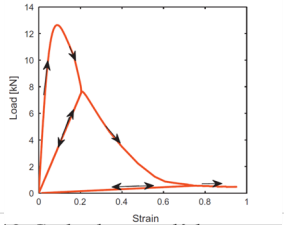


Fig 10. Cycle charge – décharge.

Fonction d'histoire

$\mathcal{H}(\boldsymbol{x}, t) = max_{\tau \in [0,t]} \{\psi_0^+(\boldsymbol{x}, \tau)\}$

$2(1-d)\mathcal{H} - \dfrac{g_c}{l}(d - l^2 \Delta d) = 0$ dans $\mathcal{B}$

$\nabla d . \boldsymbol{n} = 0$ sur $\partial \mathcal{B}$

$d = 1$ sur Γ_l

$\boldsymbol{\varepsilon}_i(\boldsymbol{u}), \ d_i$ connus

⇓ **MFRONT**

$\mathcal{H}_n = max\{\psi_{n-1}^+, \psi_n^+\}$

⇓ **CAST3M**

Résolution d'un ***problème thermique*** *pour* d_{n+1}

⇓ **MFRONT**

Construction des matrices $\boldsymbol{P}^+{}_n, \boldsymbol{P}^-{}_n, \mathbf{K}_n$

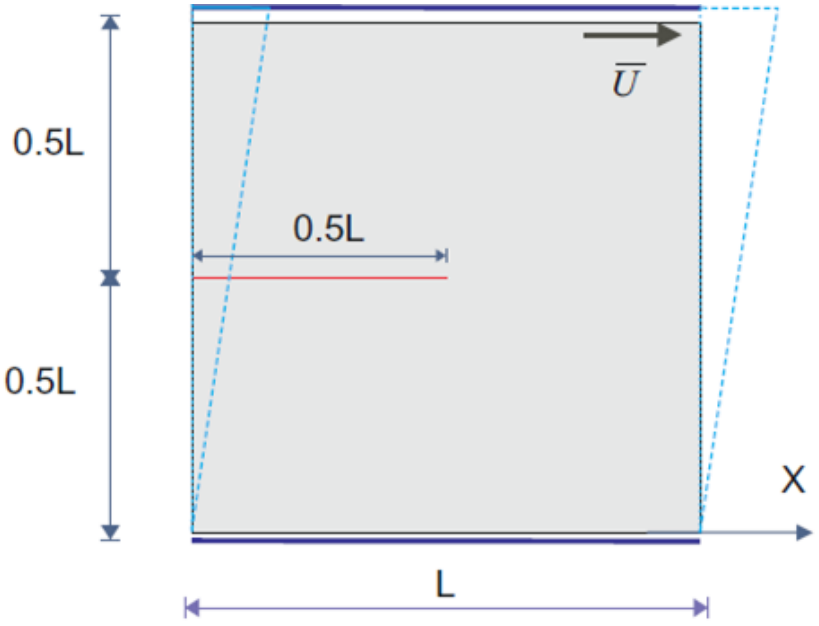
⇓ **CAST3M/COPL**

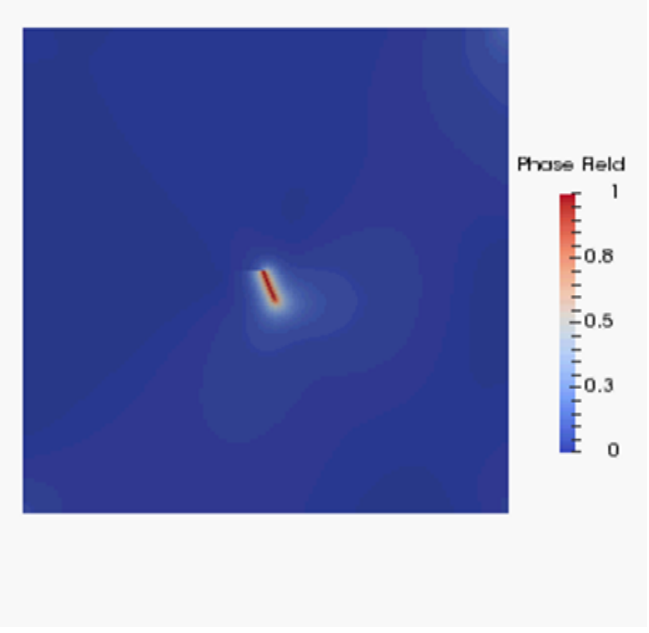
Résolution d'un ***problème mécanique*** *pour* $\boldsymbol{u}_{n+1}$

LINEAIRE

- Post-doc of Tran Thang Dang under the supervision of B. Bary (CEA/DEN/DPC/SECR/LECBA)
- Initial formulation by C. Miehe, based on Marigo/Bourdin approach
- Implementation mostly based on `MFront` and `COPL`

Phase field approach of cement brittle failure

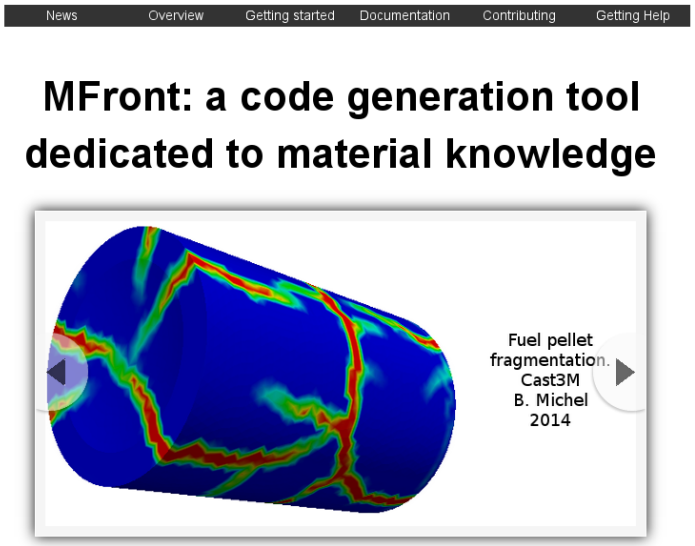


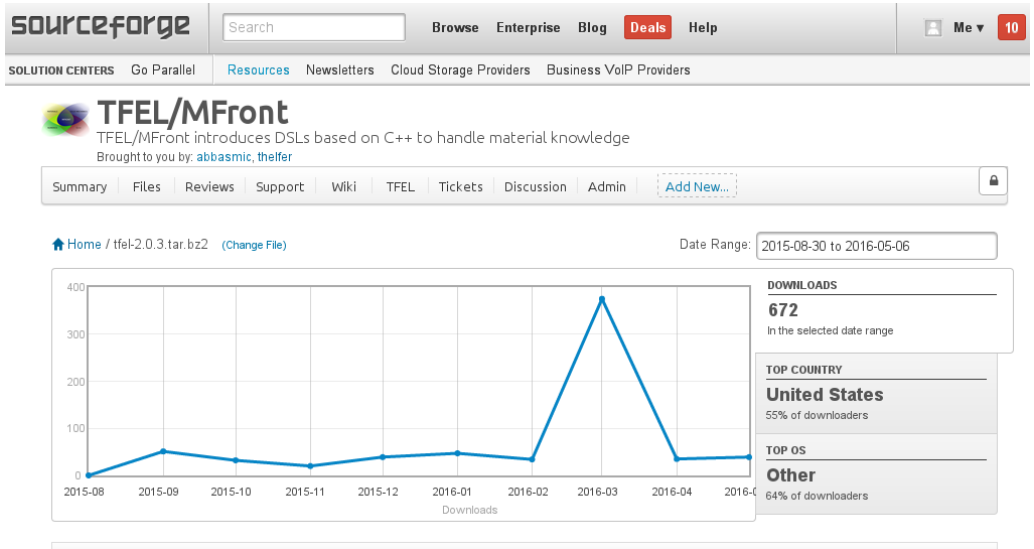


- Post-doc of Tran Thang Dang under the supervision of B. Bary (CEA/DEN/DPC/SECR/LECBA)
- Initial formulation by C. Miehe, based on Marigo/Bourdin approach
- Implementation mostly based on `MFront` and `COPL`
- Very promising results

Software developments

The TFEL website on sourceforge



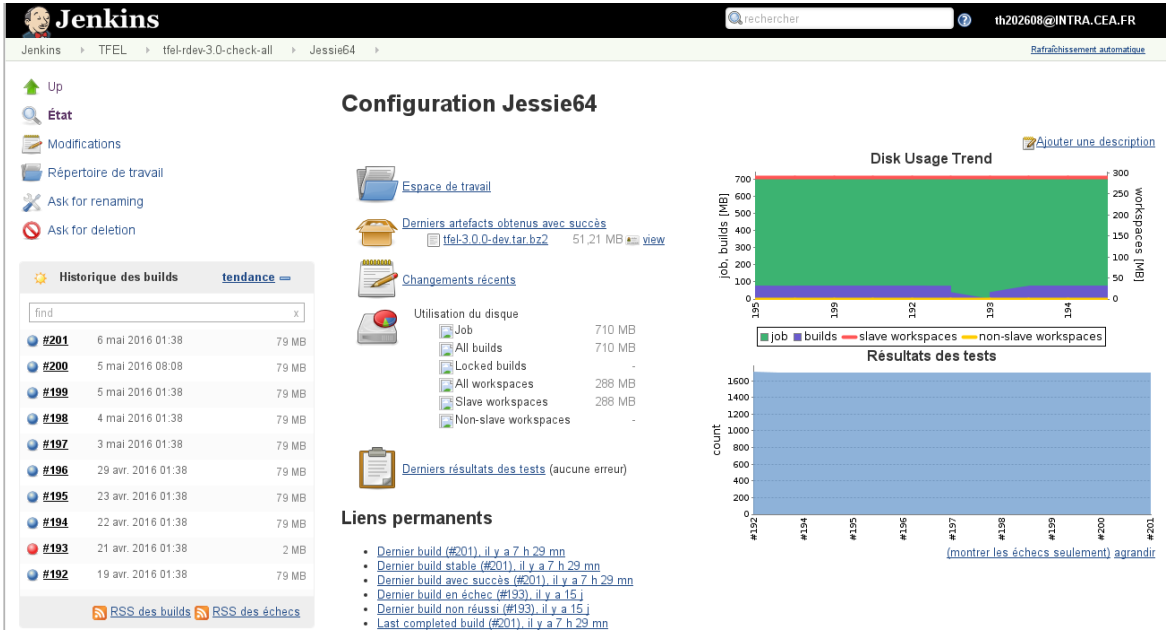


`http://tfel.sourceforge.net`

≈ 700 downloads for `tfel-2.3.0`

- "News" page regularly updated
- A forum for posting questions, requesting help
- Report bugs/improvement requests through tickets
- Download TFEL releases and development versions
- Mailing-lists (`tfel-announce` and `tfel-discuss`)
- Contact the developers : tfel-contact@cea.fr

PLEIADES software engineering workbench



- Continuous Integration and Continuous Delivery using the jenkins automation server :
  - `https://jenkins.io/`
  - `TFEL` is build at each committed change on various flavour of `LiNuX` :
    - **no automatic testing on `Windows` nor** `Mac Os`

TFEL-3.0 for the software developers

`tfel-2.0.3` :

```
void
MFrontCppLawInterface::writeOutputFiles(const std::string& file,
                                        const std::string&,
                                        const std::string& material,
                                        const std::string& className,
                                        ....
                                        const StaticVariableDescriptionContainer& staticVars,
                                        const std::vector<std::string>& params,
                                        const std::map<std::string,double>& paramValues,
                                        const LawFunction& function,
                                        const std::vector<VariableBoundsDescription>& bounds,
                                        const std::vector<VariableBoundsDescription>& physicalBounds,
                                        const bool useTemplate,
                                        const std::vector<std::string>& namespaces){
```

`tfel-3.0.x` :

```
void
CppMaterialPropertyInterface::writeOutputFiles(const MaterialPropertyDescription& mpd,
                                               const FileDescription& fd)
```

- port to C++-11 :
  - major code re-factoring :
    - suppress parts superseded by equivalents in the standard library
    - more compact code due to new language features (`auto`, `lambda`)
    - correct life time management of sub-expressions in the expression template engine
    - new **highly-documented classes** `BehaviourDescription`, `BehaviourData`, `MaterialDescription`, `FileDescription`, `TargetsDescription` to ease **interface implementations** and provide **a stable and simple API**
  - Support of the Visual Studio compiler on Windows

TFEL-3.0 for the software developers

```python
import mfront
# create dsl and analyse file
dsl = mfront.getDSL("Chaboche.mfront")
dsl.setInterfaces(["aster"])
dsl.analyseFile("Chaboche.mfront",[])
# file description
fd = dsl.getFileDescription()
print("file author:       ", fd.authorName)
print("file date:         ", fd.date)
print("file descrption:\n", fd.description)
# targets information
tgt = dsl.getTargetsDescription()
# loop over (to be) generated libraries
for l in tgt:
  print(l)
```

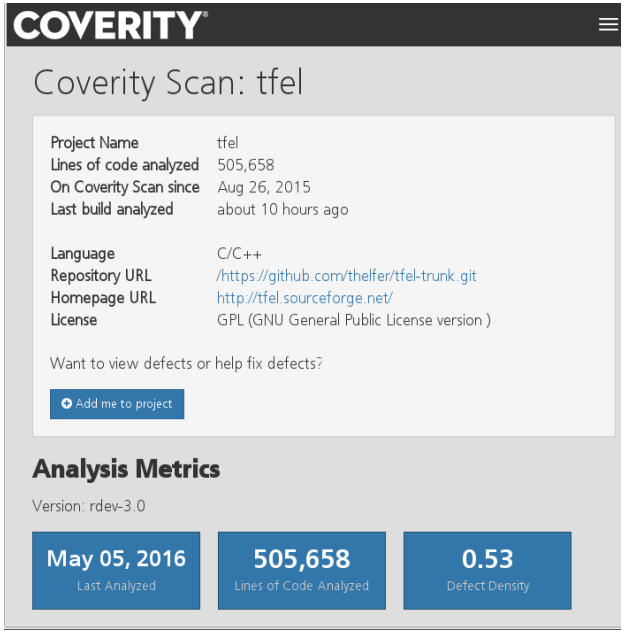
- port to C++-11 :
  - major code re-factoring :
  - Support of the Visual Studio compiler on Windows
- Improvements of the `python` bindings :
  - a new `mtest` module (called `mfront.mtest` previously)
  - a new `mfront` module to analyse MFront file

TFEL-3.0 for the software developers

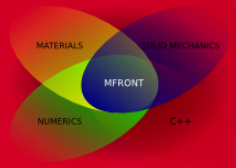
```
       Start 2197: t2tot2
2197/2202 Test #2197: t2tot2 ..................................................
.....................................   Passed    0.00 sec
       Start 2198: TensorProductDerivative
2198/2202 Test #2198: TensorProductDerivative .................................
.....................................   Passed    0.01 sec
       Start 2199: TransposeDerivative
2199/2202 Test #2199: TransposeDerivative .....................................
.....................................   Passed    0.00 sec
       Start 2200: VelocityGradientDerivative
2200/2202 Test #2200: VelocityGradientDerivative ..............................
.....................................   Passed    0.01 sec
       Start 2201: SpinRateDerivative
2201/2202 Test #2201: SpinRateDerivative ......................................
.....................................   Passed    0.01 sec
       Start 2202: cadna_cadna
2202/2202 Test #2202: cadna_cadna .............................................
.....................................   Passed    0.01 sec

100% tests passed, 0 tests failed out of 2202

Total Test time (real) = 258.39 sec
[1]+  Fini                    emacs Makefile
th202608@pleiades077:~/codes/tfel/tfel-3.0.x/src/build$
```

- port to C++-11 :
  - major code re-factoring :
  - Support of the Visual Studio compiler on Windows
- Improvements of the `python` bindings :
- Improved software quality :
  - More than 2 300 unit tests (about 800 in TFEL 2.0.3) :
    - Many test cases in client projects (`MFrontMaterials`, `Code-Aster MAP`...)

TFEL-3.0 for the software developers



- port to C++-11 :
  - major code re-factoring :
  - Support of the Visual Studio compiler on Windows
- Improvements of the `python` bindings :
- Improved software quality :
  - More than 2 300 unit tests (about 800 in TFEL 2.0.3) :
  - Static analysis (`coverity`, `cppcheck`, `clang-tidy`), stringent debug mode, sanitizers, etc...



New tools

- `tfel-check` : a tool to automate unit-tests based on external processes execution. Build-in support for results comparison to reference data.

New tools

ImplicitNorton behaviour description

- file : ImplicitNorton.mfront
- author : (unspecified)
- date : (unspecified)

This file implements the Norton law , described as :

$$\begin{cases} \underline{\epsilon}^{\mathrm{to}} = \underline{\epsilon}^{\mathrm{el}} + \underline{\epsilon}^{\mathrm{vis}} \\ \underline{\sigma} = \underline{\mathbf{D}} : \underline{\epsilon}^{\mathrm{el}} \\ \underline{\dot{\epsilon}}^{\mathrm{vis}} = \dot{p}\,\underline{n} \\ \dot{p} = A\,{\sigma_{\mathrm{eq}}}^{m} \end{cases}$$

List of supported Hypotheses

- AxisymmetricalGeneralisedPlaneStrain
- AxisymmetricalGeneralisedPlaneStress, specialised
- Axisymmetrical
- PlaneStress, specialised
- PlaneStrain
- GeneralisedPlaneStrain
- Tridimensional

- `tfel-check` : a tool to automate unit-tests based on external processes execution. Build-in support for results comparison to reference data.
- `mfront-doc` : a tool to generate a documentation out of a `MFront` file. The output is a `markdown` file suitable for `pandoc` : `http://pandoc.org`

New tools

```
$ mfront--query --parameters Chaboche.mfront
- theta (theta) : theta value used by the implicit scheme
- epsilon (epsilon) : value used to stop the iteration of the implicit algorithm
- minimal_time_step_scaling_factor (minimal_time_step_scaling_factor) : minimal value for the
    time step scaling factor
- maximal_time_step_scaling_factor (maximal_time_step_scaling_factor) : maximal value for the
    time step scaling factor
- numerical_jacobian_epsilon (numerical_jacobian_epsilon) : perturbation value used to compute
    a numerical approximation of the jacobian
- iterMax (iterMax) : maximum number of iterations allowed
```

- `tfel-check` : a tool to automate unit-tests based on external processes execution. Build-in support for results comparison to reference data.
- `mfront-doc` : a tool to generate a documentation out of a `MFront` file. The output is a `markdown` file suitable for `pandoc` : `http://pandoc.org`
- `mfront-query` : retrieve information about a `MFront` file.

Perspectives

Planed work

- Composable behaviour bricks (2016) :
- Support for non local behaviours and generalised behaviours (2016) :
  - beam, micromorphic behaviours, etc...
- Support for plane stress and generalised plane stress for specialised dsl's (2017)
- Multiphysics (2017) :
  - Coupling diffusivity and mechanics
- Smart prediction for implicit schemes (2017) :
- Smart criteria for time-step increase/reduction (2017) :
- Multi-surface plasticity (2017) :
  - much needed for fuel performance codes (DDIF2 behaviour)

Improvements to the Cast3M interface

Standard version of Cast3M

```
mod1 = 'MODELISER' s1 'MECANIQUE' 'ELASTIQUE' 'ISOTROPE'
    'NON_LINEAIRE' 'UTILISATEUR'
    'LIB_LOI' 'src/libUmatBehaviour.so'
    'FCT_LOI' 'umatnorton'
    'C_MATERIAU' coel2D
    'C_VARINTER' stav2D
    'PARA_LOI'   para2D
    'CONS' M;
```

- Call to external behaviour is supported since Cast3M 2015.
- Call to external material properties is supported since Cast3M 2016.
- Works on Linux, Windows
  - Mac Os untested

Standard version of Cast3M

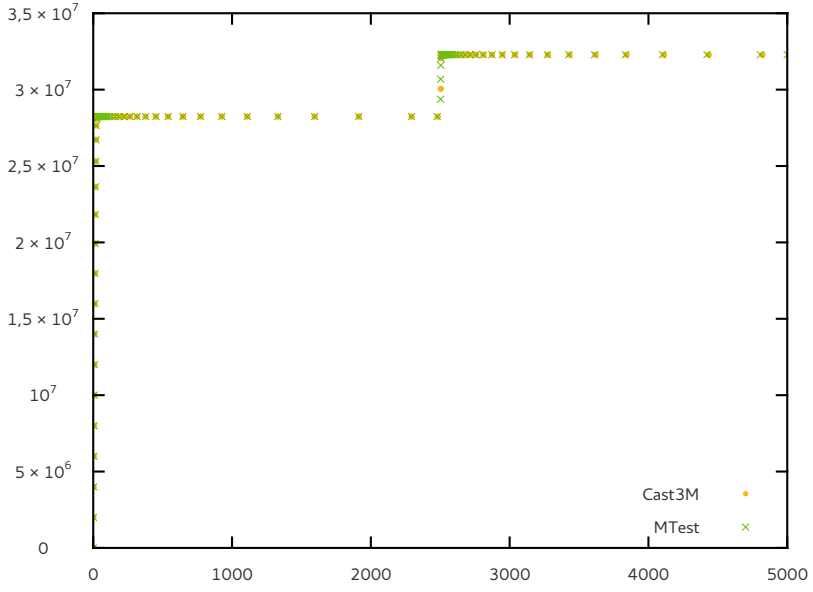
```
Ty = 'TABLE' ;
Ty.'LIB_LOI'    = 'libCastemM5.so' ;
Ty.'FCT_LOI'    = 'M5_YoungModulus' ;
Ty.'VARIABLES' = 'MOTS' 'T';

mo = 'MODELISER' m 'MECANIQUE' 'ELASTIQUE';
ma = 'MATERIAU' mo 'YOUN' Ty 'NU' 0.3;
```

- Call to external behaviour is supported since Cast3M 2015.
- Call to external material properties is supported since Cast3M 2016.
- Works on Linux, Windows
  - Mac Os untested

Version of Cast3M specific to PLEIADES

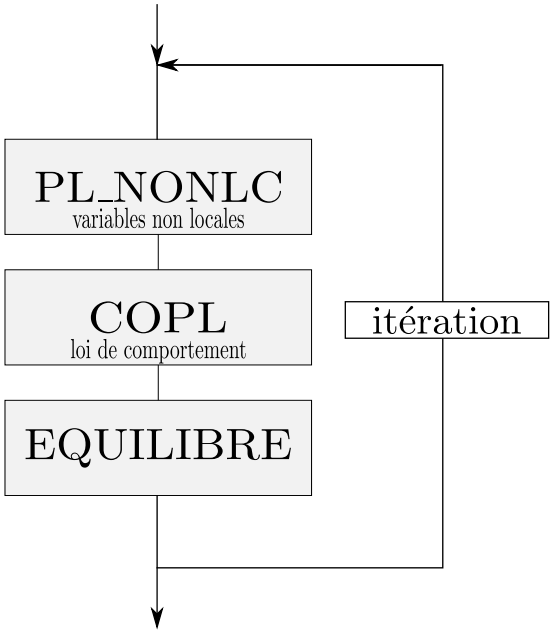
- Specialised operators have been developed for PLEIADES :
  - `COPL` for mechanical behaviour integration
  - `INCREPL/MEPL` for solving the mechanical equilibrium on one step

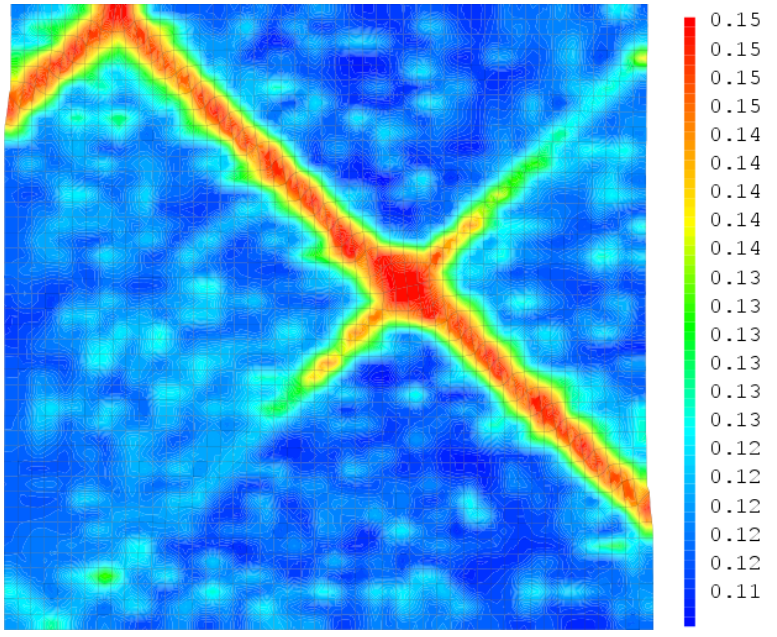
Version of Cast3M specific to PLEIADES



- Specialised operators have been developed for PLEIADES :
  - `COPL` for mechanical behaviour integration
  - `INCREPL/MEPL` for solving the mechanical equilibrium on one step
- Support for time step reduction/increase

Version of Cast3M specific to PLEIADES





- Specialised operators have been developed for PLEIADES :
  - `COPL` for mechanical behaviour integration
  - `INCREPL/MEPL` for solving the mechanical equilibrium on one step
- Support for time step reduction/increase
- Support for non local behaviours

Version of Cast3M specific to PLEIADES

Operator	Algorithm	CPU time
MEPL	Consistent tangent operator	0.328s
INCREPL	Consistent tangent operator	1.161s
MEPL	Cast3M acceleration	0.599s (divergence)
INCREPL	Cast3M acceleration	0.570s (divergence)
PASAPAS	Cast3M acceleration	3.932s (divergence)
PASAPAS (2015/COMP)	Cast3M acceleration	5.805s (divergence)

- Specialised operators have been developed for PLEIADES :
  - `COPL` for mechanical behaviour integration
  - `INCREPL/MEPL` for solving the mechanical equilibrium on one step
- Support for time step reduction/increase
- Support for non local behaviours
- Support for consistent tangent operator

Feed-back of MFront usage in Code_Aster

Main feed-back

Je viens d'en discuter avec Mathieu et on n'a pas vraiment souvenir de problèmes. Normalement, c'est l'inverse, les gens se souviennent plus facilement de ce qui n'a pas marché. On doit être d'un naturel optimiste. :-)

Plus sérieusement, je pense que ce qu'il faut retenir, c'est que l'intégration de MFront dans Aster a été et est toujours simple.

On n'a encore jamais été dans la situation de dire qu'une chose ne sera pas possible dans Aster avec MFront à cause d'un problème d'interfaçage ou d'une limitation informatique.

Improvements to the Code-Aster interface in 2015

```
MATF=DEFI_MATERIAU(UMAT=_F( C1  = young,
                            C2  = 0.3 ,
                            NB_VALE=12,
                            C3  = 151.,
                            C4  = 87.,
                            C5  = 2.3,
                            C6  = k,
                            C7  = w,
                            C8  = C1_I,
                            C9  = C2_I,
                            C10 = g1,
                            C11 = g2,
                            C12 = 1.,),)
```

```
MATF=DEFI_MATERIAU(Chaboche=_F(YoungModulus=young,
                               PoissonRatio=0.3 ,
                               R_inf  = 151.,
                               R_0  = 87.,
                               b  = 2.3,
                               k  = k,
                               w  = w,
                               C_inf_0 = C1_I,
                               C_inf_1 = C2_I,
                               g_0_0 = g1,
                               g_0_1 = g2,
                               a_inf = 1.,),)
```

- The names of the material properties in DEFI_MATERIAU are the ones declared in the MFront file.

Improvements to the Code_Aster interface in 2015

AnisoLemaitre, Burger, Chaboche, DruckerPrager, DruckPragEcroLin, GdefMonoCrystal, GdefMono_Jacnum, GTN, Hayhurst, ImplicitHayhurst, ImplicitNorton, MetaAcierEPIL_PT, MonoCrystal_CFC, MonoCrystal_DD_CC_InteractionMatrix, MonoCrystalDDCC, MonoCrystal_DD_CC_SlidingSystems, MonoCrystal_DD_CFC_InteractionMatrix, MonoCrystalDDCFC, MonoDDCC_Irra, MonoDDCFC_Irra, Norton_Jacnum, Norton, Norton_RK54, OrthotropicElast, Plasticity, Plasticity_Sy, PlasticityTH, PolyCrystalDDCC, PolyCrystal_DD_CC_SlidingSystems, PolyCrystalDDCFC, PolyCrystal_MC, PolyCrystal_Orientation_100grains, PolyCrystal_Orientation, SimoMieheVmis, SVenantKirchhoff, Tvergaard, ViscoChaboche, ViscoMemoNrad

- The names of the material properties in DEFI_MATERIAU are the ones declared in the MFront file.
- 38 MFront behaviours have superseded the native implementations
  - transparent for the end-user

Current work on the Code_Aster interface in 2016

- Port to MFront 3.0
- Better error messages in Code_Aster
  - Better handling of convergence difficulties
  - Handling of out of bounds policies
- Allow transparent use of Hillberborg-like models
  - access to characteristic element size(s) using a convention on entry names

The Europlexus Interface

The Abaqus Interface

Conclusion of the User day

Merci et à l'année prochaine !

Commissariat à l'énergie atomique et aux énergies alternatives
Centre de Cadarache | DEN/DEC/SESC b151 - 13108 Saint-Paul-Lez-Durance
T. +33 (0)4.42.25.23.66 | F. +33 (0)4.42.25.47.47
Établissement public à caractère industriel et commercial | RCS Paris B 775 685 019

Direction de l'Énergie Nucléaire
Département d'Études des Combustibles
Service d'Études et de Simulation du comportement des Combustibles