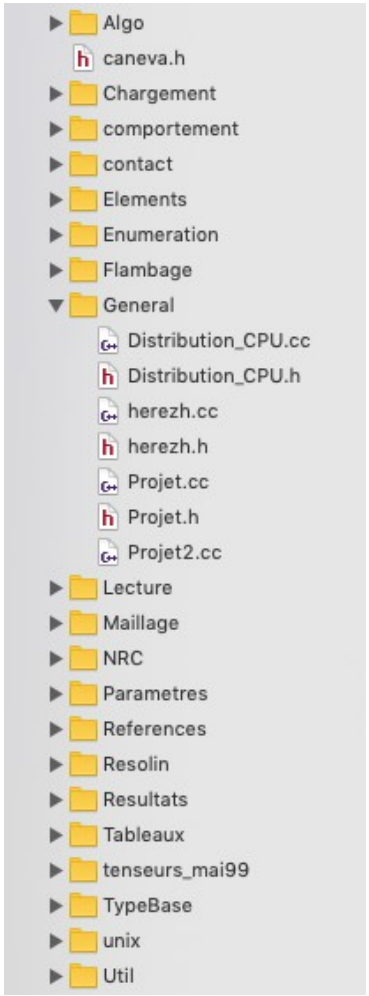
Formation pour développeurs : Logiciel Herezh++

Première partie

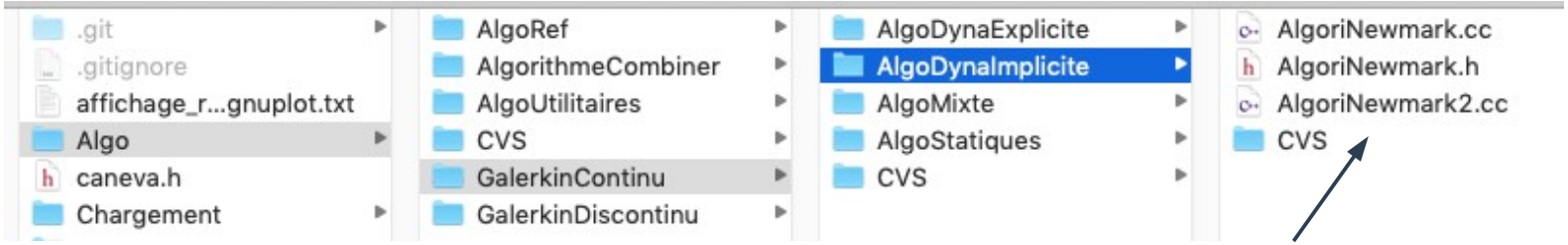
Gérard Rio
Professeur émérite Université Bretagne sud
Juin 2023

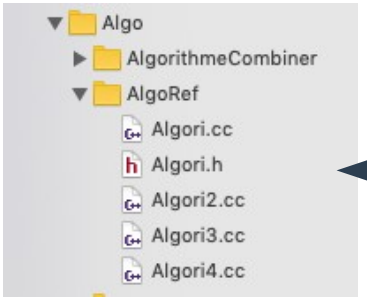
Les sources : (normalement en UTF8)



- Un répertoire par thème : contient les .h et les .cc

- puis une hiérarchie → et les classes finales



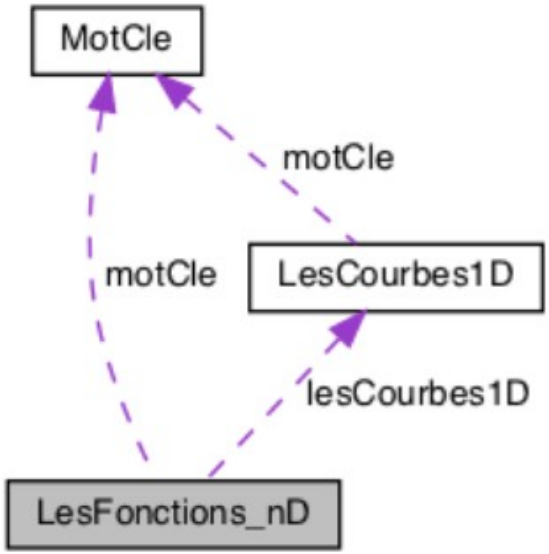


AlgoRef :
la classe générique
de tous les algos

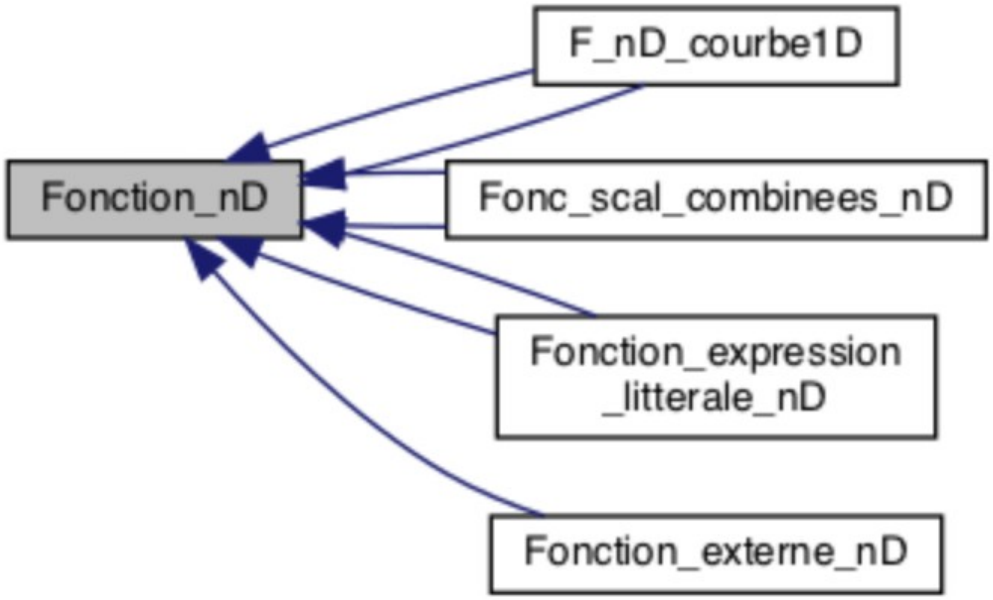
AlgoriNewmark :
algorithme de Newmark

La classe et le nom du fichier sont identiques (ou très proches) :
AlgoriNewmark.h → AlgoriNewmark()

Fonctions utilisateurs : nD



. Fonction_nD : Une classe virtuelle partielle d'interface :
Contiens la gestion générique des données
. Des fonctions spécifiques
. LesFonctions_nD : classe de gestion de l'ensemble des fct nD



Principe d'utilisation

1) demande à la fct pour obtenir les infos et le stockage nécessaires à son calcul
2) récupération des infos et stockage dans les conteneurs fournis par la fct
3) appel de la fonction pour son calcul et récupération du résultat

Exemple : dans le calcul de la pression : cf. ElemMeca::SMR_charge_pres_I(...

```
if (pt_fonct != NULL)
 { // on commence par récupérer les conteneurs des grandeurs à fournir
   List_io <Ddl_enum_etendu>& li_enu_scal = pt_fonct->Li_enu_etendu_scalaire();
   List_io <TypeQuelconque >& li_quelc = pt_fonct->Li_equi_Quel_evolue();
   bool absolue = true; // on se place systématiquement en absolu
   // on va utiliser la méhode Valeur_multi_interpoler_ou_calculer pour les grandeurs strictement scalaire
    Tableau <double> val_ddl_enum(Valeur_multi_interpoler_ou_calculer
              (absolue,TEMPS_tdt,li_enu_scal,defS,ex_impli,ex_expli_tdt,ex_expli) );
   // on utilise la méthode Valeurs_Tensorielles_interpoler_ou_calculer pour les Coordonnees et Tenseur
   Valeurs_Tensorielles_interpoler_ou_calculer
              (absolue,TEMPS_tdt,li_quelc,defS,ex_impli,ex_expli_tdt,ex_expli);
   // calcul de la valeur et retour dans tab_ret
   Tableau <double> & tab_val = pt_fonct->Valeur_FnD_Evoluee(&val_ddl_enum,&li_enu_scal,&li_quelc,NULL,NULL);
    // seule la première valeur est ok
    pression_applique *= tab_val(1);
    };
```

Types principaux d'appel a utiliser pour calculer la fonction

```
// calcul  des valeurs de la fonction, à l'aide de paramètres =  grandeurs évoluées
// retour d'un tableau de scalaires
// les différents tableaux doivent contenir toutes les informations nécessaires pour l'appel
// l'ordre des paramètres doit respecter celui donné par les fonctions Index_dans_tableau()
// il peut y avoir éventuellement des grandeurs quelconques et éventuellement des types non scalaire
// dans ce dernier cas il faut renseigner le tableau de numéro d'ordre t_num_ordre
Tableau <double> & Val_FnD_Evoluee(Tableau <double >* val_ddl_enum
                    ,Tableau <Coordonnee> * coor_ddl_enum
                    ,Tableau <TenseurBB* >* tens_ddl_enum
                    ,Tableau <const TypeQuelconque *  >* tqi = NULL
                    ,Tableau <int> * t_num_ordre = NULL
                   )
// calcul équivalent : pour que l'appel à cette méthode soit correcte, il faut que les listes correspondent
// à celles de Li_equi_Quel_evolue(), et Li_enu_etendu_scalaire(),
// =>=>=> il faut donc utiliser ces listes en les récupérants auparavant !!
// Si t_num_ordre est Null cela signifie que tous les tqi sont de grandeurs scalaire
// dans le cas contraire t_num_ordre(i) donne le numéro d'ordre du scalaire dans tqi qui correspond à Nom_variable(i)
// =>=>=> important: il est permis "que" de changer les valeurs dans les conteneurs
Tableau <double> & Valeur_FnD_Evoluee(Tableau <double >* val_ddl_enum
                     ,List_io <Ddl_enum_etendu>* li_evolue_scalaire
                     ,List_io <TypeQuelconque >* li_evoluee_non_scalaire
                     ,Tableau <const TypeQuelconque *  >* tqi
                     ,Tableau <int> * t_num_ordre)
```

Les conteneurs ? Tout d'abord : Ddl_enum_etendu

```
*     BUT:  une classe qui permet de définir des identificateurs de
*           grandeurs secondaires associées à des Enum_ddl ddl de base.
*           Ces grandeurs ne sont pas vraiment des ddl, mais ils s'en
*           déduisent. Ainsi pour ne pas alourdir le type Enum ddl
class Ddl_enum_etendu
{ ….les différentes méthodes qui permettent de manipuler les variables protégées
  protected :
    // VARIABLES PROTEGEES :
    string nom;
    Enum_ddl enu;
    int posi_nom;  // position du nom  : équivalent à un type énuméré
---------------------------------------  Enum_ddl quesaco ? ----------------------------
//  l'énumération de degré de liberté de base, celle qui sert pour les calculs   (nombre de ddl limité)
enum Enum_ddl { X1 = 1, X2 , X3, EPAIS , TEMP , UX, UY, UZ , V1 , V2 , V3,
                PR, GAMMA1, GAMMA2, GAMMA3,
                SIG11,SIG22,SIG33,SIG12,SIG23,SIG13,ERREUR,
                EPS11,EPS22,EPS33,EPS12,EPS23,EPS13,
                DEPS11,DEPS22,DEPS33,DEPS12,DEPS23,DEPS13,
                PROP_CRISTA,DELTA_TEMP,FLUXD1,FLUXD2,FLUXD3,R_TEMP,
                GRADT1,GRADT2,GRADT3,DGRADT1,DGRADT2,DGRADT3,
                R_X1,R_X2,R_X3,R_EPAIS,R_V1,R_V2,R_V3,R_GAMMA1,R_GAMMA2,R_GAMMA3,
                NU_DDL };
```

Nombreux types énumérés dans Herezh :
- amélioration de la lisibilité
- permet de typer des fonctions spécifiques

Enum_boolddl.cc
Enum_boolddl.h
Enum_calcul_masse.cc
Enum_calcul_masse.cpp
Enum_calcul_masse.h
Enum_categorie_loi_comp.cc
Enum_categorie_loi_comp.h
Enum_chargement.cc
Enum_chargement.h
Enum_comp.cc
Enum_comp.h
Enum_contrainte_mathematique.cc
Enum_contrainte_mathematique.h
Enum_crista.cc
Enum_crista.h
Enum_Critere_loi.cc
Enum_Critere_loi.h
Enum_ddl (copy/ conflict on 2017-11-21).cc
Enum_ddl (copy/ conflict on 2017-11-21).h
Enum_ddl_var_static.cc
Enum_ddl.cc
Enum_ddl.h
Enum_dure.cc
Enum_dure.h
Enum_geom.cc
Enum_geom.h
Enum_GrandeurGlobale.cc
Enum_GrandeurGlobale.h
Enum_interpol.cc
Enum_interpol.h
Enum_IO_XML.cc
Enum_IO_XML.h
Enum_liaison_noeud.cc
Enum_liaison_noeud.h

Enum_mat.cc
Enum_mat.h
Enum_matrice.cc
Enum_matrice.cpp
Enum_matrice.h
Enum_PiPoCo.cc
Enum_PiPoCo.h
Enum_proj_aniso.cc
Enum_proj_aniso.h
Enum_StabHourglass.cc
Enum_StabHourglass.h
Enum_StabMembrane.cc
Enum_StabMembrane.h
Enum_Suite.cc
Enum_suite.h
Enum_type_deformation.cc
Enum_type_deformation.h
Enum_type_geom.cc
Enum_type_geom.h
Enum_type_pt_integ.cc
Enum_type_pt_integ.h
Enum_type_resolution_matri.cc
Enum_type_resolution_matri.cpp
Enum_type_resolution_matri.h
Enum_type_stocke_deformation.cc
Enum_type_stocke_deformation.h
Enum_TypeQuelconque.cc
Enum_TypeQuelconque.h
Enum_variable_metrique.cc
Enum_variable_metrique.h
EnumCourbe1D.cc
EnumCourbe1D.h
EnumElemTypeProblem.cc
EnumElemTypeProblem.h

EnumFonction_nD.cc
EnumFonction_nD.h
EnumLangue.cc
EnumLangue.h
EnumTypeCalcul.cc
EnumTypeCalcul.h
EnumTypeGradient.cc
EnumTypeGradient.h
EnumTypeGrandeur.cc
EnumTypeGrandeur.h
EnumTypePilotage.cc
EnumTypePilotage.h
EnumTypeViteRotat.cc
EnumTypeViteRotat.h
EnumTypeVitesseDefor.cc
EnumTypeVitesseDefor.h
EnuTypeCL.cc
EnuTypeCL.h
EnuTypeQuelParticulier.cc
EnuTypeQuelParticulier.h
MotCle.cc
MotCle.h

Exemple d'organisation typique d'un type énuméré

```
/// énuméré permettant de savoir si une loi est : 1D, 2D en deformations planes ou contraintes planes, 3D générale
enum Enum_comp_3D_CP_DP_1D { COMP_1D =1, COMP_CONTRAINTES_PLANES
                    , COMP_DEFORMATIONS_PLANES, COMP_3D , RIEN_COMP_3D_CP_DP_1D};
// Retourne le nom d'une loi de comportement a partir de son identificateur
string Nom_comp(const Enum_comp id_comport);
// Retourne l'identificateur de type enumere associe au nom de la loi de comportement nom_comport
Enum_comp Id_nom_comp(const string& nom_comport);
// indique si la loi est inactive mécaniquement typiquement de type : LOI_RIEN... ou RIEN_COMP
bool Loi_rien(const Enum_comp id_comport);
// surcharge de l'operator de lecture
istream & operator >> (istream & entree, Enum_comp& a);
// surcharge de l'operator d'ecriture
ostream & operator << (ostream & sort, const Enum_comp& a);
// ----- 2) concernant Enum_comp_3D_CP_DP_1D -----
// Retourne le nom  a partir de son identificateur du type enumere id_Enum_comp_3D_CP_DP_1D correspondant
string Nom_comp_3D_CP_DP_1D(const Enum_comp_3D_CP_DP_1D id_Enum_comp_3D_CP_DP_1D);
// Retourne l'identificateur de type enumere associe à un nom nom_comp_3D_CP_DP_1D
Enum_comp_3D_CP_DP_1D Id_nom_comp_3D_CP_DP_1D(const string nom_comp_3D_CP_DP_1D);
// indique le type Enum_comp_3D_CP_DP_1D correspondant à une  loi de comportement
Enum_comp_3D_CP_DP_1D Comp_3D_CP_DP_1D(const Enum_comp id_comport);
// surcharge de l'operator de lecture
istream & operator >> (istream & entree, Enum_comp_3D_CP_DP_1D& a);
// surcharge de l'operator d'ecriture
ostream & operator << (ostream & sort, const Enum_comp_3D_CP_DP_1D& a);
```

Suite des Ddl_enum_etendu

```
protected :
  // VARIABLES PROTEGEES :
  string nom;
  Enum_ddl enu;
  int posi_nom;  // position du nom  : équivalent à un type énuméré

  // on définie le  tableau des Ddl_enum_etendu qui sont valides en variables globales
  static Tableau < Ddl_enum_etendu > tab_Dee;
```

---------- et au niveau du constructeur on définies les Ddl enum etendu ----------

```
  tab_Dee(1).nom = "Green-Lagrange11" ;tab_Dee(1).enu = EPS11;tab_Dee(1).posi_nom = nbenumddl + 1;
  tab_Dee(2).nom = "Green-Lagrange22" ;tab_Dee(2).enu = EPS22;tab_Dee(2).posi_nom = nbenumddl + 2;
  tab_Dee(3).nom = "Green-Lagrange33" ;tab_Dee(3).enu = EPS33;tab_Dee(3).posi_nom = nbenumddl + 3;
  tab_Dee(4).nom = "Green-Lagrange12" ;tab_Dee(4).enu = EPS12;tab_Dee(4).posi_nom = nbenumddl + 4;
  tab_Dee(5).nom = "Green-Lagrange13" ;tab_Dee(5).enu = EPS13;tab_Dee(5).posi_nom = nbenumddl + 5;
.....
  tab_Dee(131).nom = "I_B" ;    tab_Dee(131).enu = EPS11; tab_Dee(131).posi_nom = nbenumddl + 131;
  tab_Dee(132).nom = "II_B"  ;  tab_Dee(132).enu = EPS11; tab_Dee(132).posi_nom = nbenumddl + 132;
  tab_Dee(133).nom = "III_B"  ; tab_Dee(133).enu = EPS11; tab_Dee(133).posi_nom = nbenumddl + 133;
  tab_Dee(134).nom = "J1" ;     tab_Dee(134).enu = EPS11; tab_Dee(134).posi_nom = nbenumddl + 134;
  tab_Dee(135).nom = "J2"  ;   tab_Dee(135).enu = EPS11; tab_Dee(135).posi_nom = nbenumddl + 135;
  tab_Dee(136).nom = "J3"  ;  tab_Dee(136).enu = EPS11; tab_Dee(136).posi_nom = nbenumddl + 136;
```

Donc au final : Ddl_enum_etendu

Ce sont des noms qui sont utilisés via des équivalents d'entiers :

- permet une adresse indirecte rapide
- vérification du typage dans les passages de paramètres
- lisibilité du code
- méthodes spécifiques, ex :

```
// retour le type de grandeur auquel apartient le ddl étendue
// par exemple : UY : appartiens à un vecteur
// SIG12 : à un tenseur, TEMP : à un scalaire
EnumTypeGrandeur TypeDeGrandeur() const;
```

- manipulation uniquement via des méthodes

→ encapsulage total, ex :

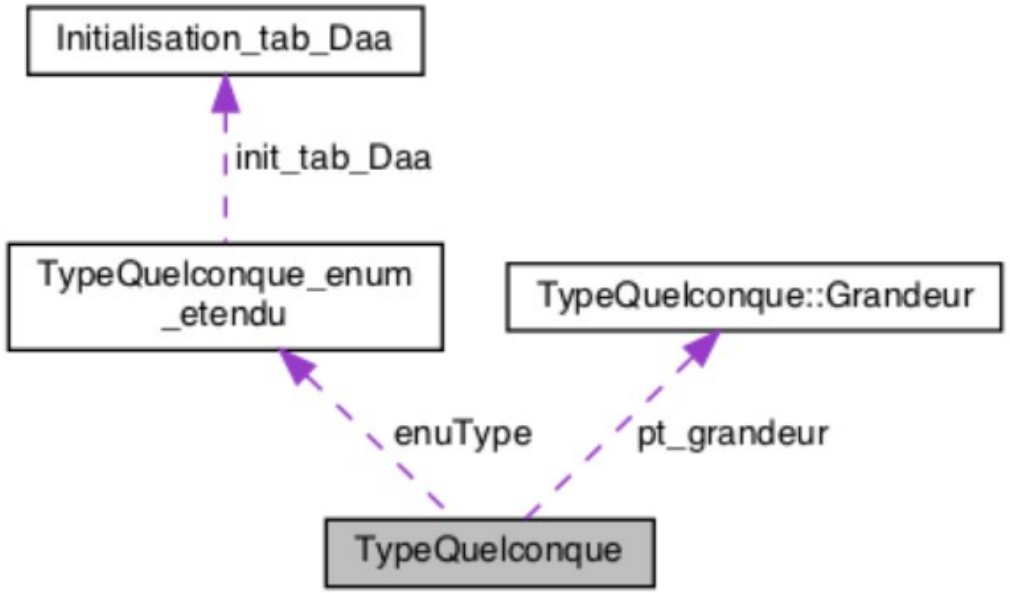
```
Enum_ddl Enum() const {return enu;}; // récup de l'énumération
 string Nom() const {return nom;};// récup du nom
 string NomGenerique() const
  {if (nom=="-"){return string(NomGeneric(enu));}
   else {return string(nom.substr(1,nom.length()-2));};
   }; // récup du nom générique (sans les indices de composantes)
```

→ traçabilité dans le cas d'erreur via un breakpoint sur la méthode qui induit l'erreur

Pratiquement toutes les entitées manipulées dans Herezh sont référencées dans les `Ddl_enum_etendu` et les `TypeQuelconque_enum_etendu`

La classe d'interface : TypeQuelconque

Uilisée
- en particulier par les fct nD,
- initialement pour les I/O d'Herezh
- mais finalement pratiquement partout dans Herezh !



Cousin germain de
Ddl_enum_etendu : → TypeQuelconque_enum_etendu

Ddl_enum_etendu Est le prolongement du type énuméré Enum_ddl

TypeQuelconque_enum_etendu Est le prolongement du type énuméré EnumTypeQuelconque

---------- le type énuméré ----------

```
/// Définition de l'enuméré pour repérer un type quelconque
enum EnumTypeQuelconque { RIEN_TYPEQUELCONQUE = 1, SIGMA_BARRE_BH_T
                ,CONTRAINTE_INDIVIDUELLE_A_CHAQUE_LOI_A_T
                ,CONTRAINTE_INDIVIDUELLE_A_CHAQUE_LOI_A_T_SANS_PROPORTION
                ,CONTRAINTE_COURANTE,DEFORMATION_COURANTE,VITESSE_DEFORMATION_COURANTE
.....
                 ,COMP_TORSEUR_REACTION
                ,NUM_NOEUD,NUM_MAIL_NOEUD,NUM_ELEMENT,NUM_MAIL_ELEM,NUM_PTI
                ,NUM_FACE,NUM_ARETE
               };
```

Actuellement (juin 2023) 224 éléments !

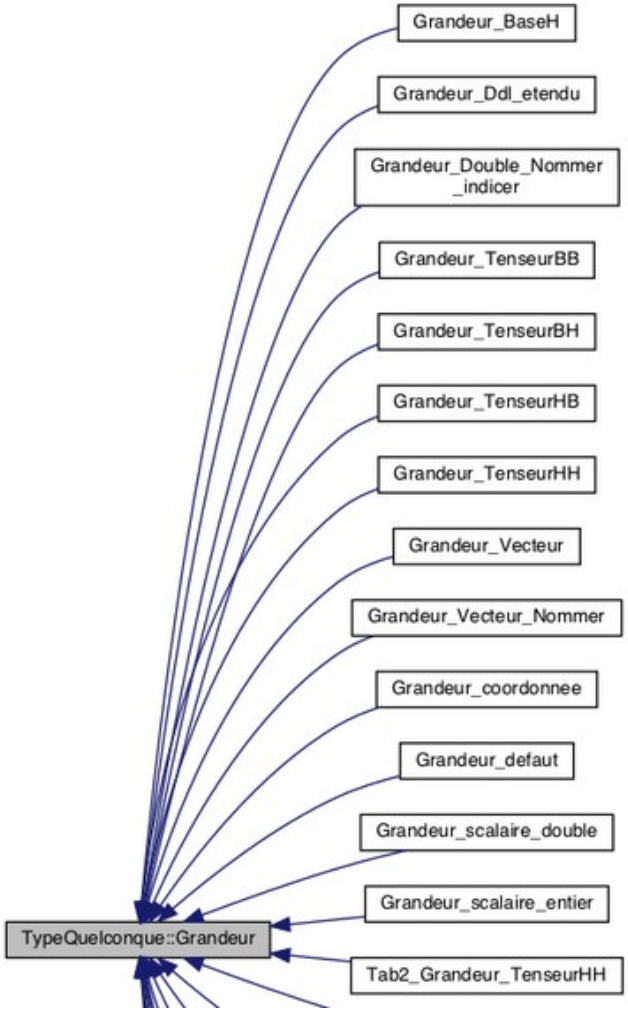
→ Utilisation d'une map pour les opérations de trie et de recherche rapide lors d'utilisation des chaines de caractère

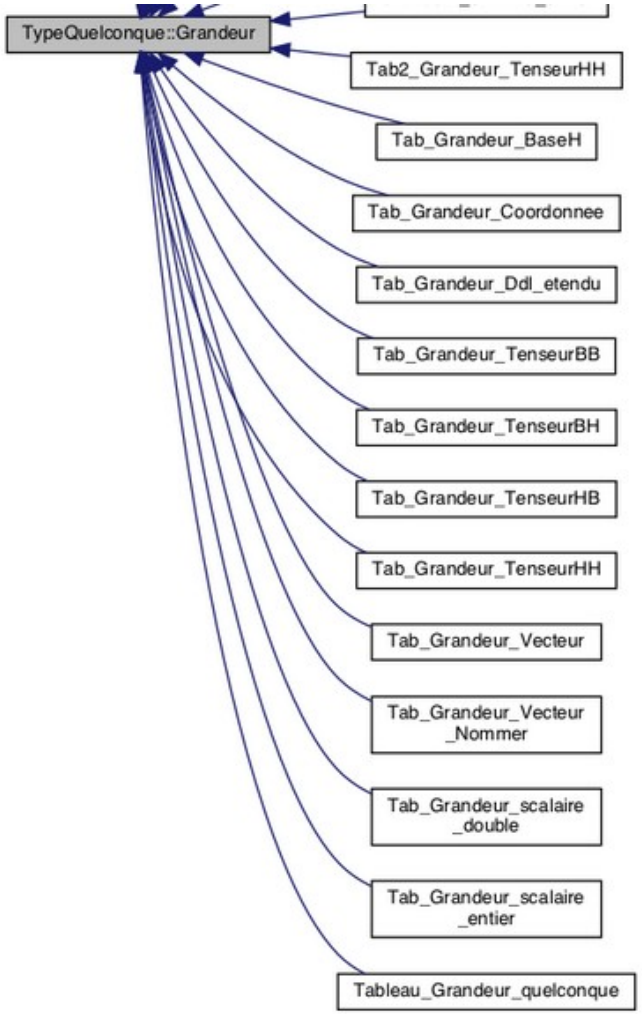
```
// def de la map qui fait la liaison entre les string et les énumérés
static map < string, EnumTypeQuelconque , std::less < string> >
map_EnumTypeQuelconque;
```

→ Sinon : utilisation de switch + case pour l'adressage direct d'un enuméré

Classe virtuelle : TypeQuelconque::Grandeur

→ interface des grandeurs utiles





Utilisation typique
Les méthodes principales

Exemple d'utilisation :

On est dans un élément de contact et on veut récupérer la normale à la frontière via le nœud qui la stocke :

1) on demande au nœud de fournir la grandeur quelconque associée

```
const TypeQuelconque& tiq = tabNoeud(2)→Grandeur_quelconque(N_FRONT_t);
```

2) on extrait la grandeur que l'on sait être de type coordonnées donc on la cast en un type `Grandeur_coordonnee`

```
const Grandeur_coordonnee& gr= *((Grandeur_coordonnee*) (tiq.Const_Grandeur_pointee()));
```

3) on récupère la normale (en lecture seule ici)

```
N = gr.ConteneurCoordonnee_const();
```

Méthodes principales :

. récupération du nom → permet de faire un cast

. opérations de base sur grandeur non typée : +,-,*, +=, -=, *=, =, <<, ”

. init

. récup de la grandeur typée:

→ en const ou non (permet de gérer les méthodes const ou non)

→ accès aux spécificités de la grandeur typée

. sur grandeur non typée :

→ si numérique, le nombre de valeurs numériques

→ accès à chaque valeurs numériques

→ I/O sur `istream et ostream utilisée par ex. pour le .BI`

Application : introduction de l'utilisation de fonction nD pour les paramètres des lois viscoélastiques de type Maxwell : Loi_maxwell3D, Loi_maxwell2D_D, Loi_maxwell2D_C,Loi_maxwell1D

Les lois intègrent déjà l'option : paramètres dépendants de courbe 1D

→ on peut se servir de l'exemple d'implantation courbe 1D pour introduire les fct nD

→ on peut se servir de l'exemple de la loi Loi_iso_elas3D qui utilise conjointement des courbes 1D et des fonction nD

Indications de méthodologie : (cf. ex Loi_iso_elas3D) **1) gestion des données**

- déclaration dans le .h des fct nD
- intégration des fct nD dans : les contructeurs, le destructeur,
- lecture des données Loi_iso_elas3D::LectureDonneesParticulieres

À associer avec les informations à transmettre à l'utilisateur

Loi_iso_elas3D::Info_commande_LoisDeComp

- lecture et écriture Base_info : sert pour .BI et le restart
- affichage et test pour savoir si tout est complet :

Loi_iso_elas3D::Affiche() et Loi_iso_elas3D::TestComplet()

- gestion I/O des grandeurs particulières : (sera peut-être modifié ou complété lors de l'utilisation)

Loi_iso_elas3D::Grandeur_particuliere et Loi_iso_elas3D::ListeGrandeurs_particulieres

se font conjointement avec le stockage particulier au pti :

Loi_comp_abstraite::SaveResul qui donc doit-être modifié

Et également mettre à jour si besoin : Insertion_conteneur_dans_save_result et Activation_stockage_grandeurs_quelconques

QQ Régles d'écriture, développement, suggestion

- utiliser la hierarchie des sources pour inclure de nouveaux fichiers : .h pour header et .cc pour le source d'exécution
- **en édition** :
. pas d'utilisation de tabulation, utilisation exclusive d'espace (ex : pour l'indentation)
. utilisation systématique de l'indentation (corps de méthode, if, switch etc.)
. utilisation du codage UTF8 (unix, pas d'iso, pas de macos)
. nom de méthode, de classe : première lettre en majuscule
. nom de variable, d'objet (instance de classe) : première lettre en minuscule
. nom explicite (éviter les chaines incompréhensibles)
. utilisation de préférence de méthodes dédiées plutôt qu'implantée le développement de bas niveau : si pb de vitesse → utilisation d'inline
- **suggestion de couleurs** dans un environnement de développement (CodeBlocks, codelite, Xcode etc. : utilisation fortement recommandée)

Suite de l'application sur les fct nD

2) Utilisation des fct nD

- dans les méthodes spécifiques :

  Ex : Module_young_equivalent , Module_compressibilite_equivalent

- dans les 3 méthodes génériques :

  Calcul_SigmaHH Calcul_DsigmaHH_tdt Calcul_dsigma_deps

- dans les méthodes associés (utilisées) par les méthodes génériques (dépendent du type de loi)

Ex (`Loi_iso_elas3D`)

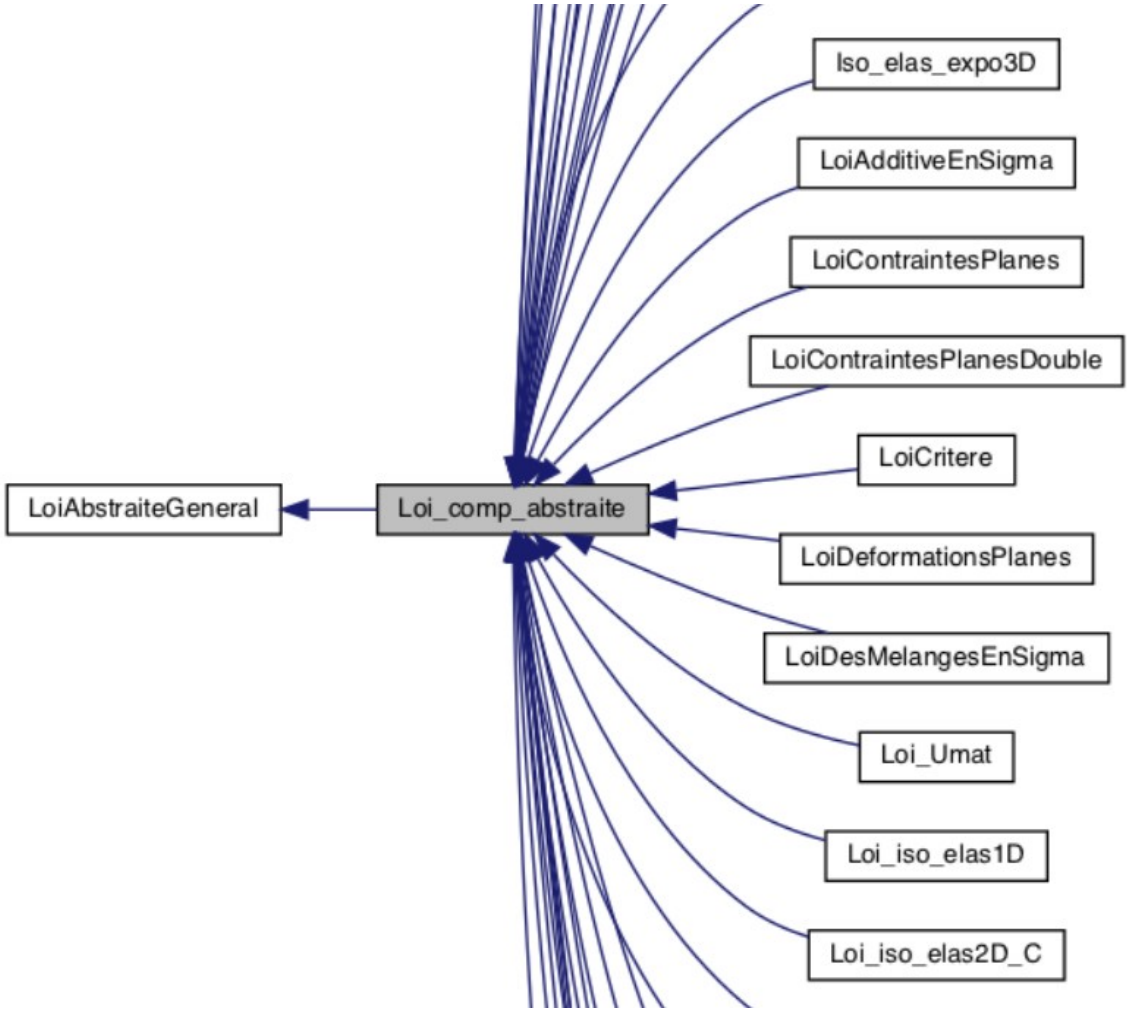
CalculGrandeurTravail

3) écriture d'un test basique d'utilisation :

- rédaction du test via l'utilisation d'Herezh en interactif → vérif de Loi_iso_elas3D::Info_commande_LoisDeComp
- utilisation du test → vérif que cela tourne : Calcul_SigmaHH Calcul_DsigmaHH_tdt Calcul_dsigma_deps
- sortie de grandeurs au format maple et au format gmsh :

→ vérif de Loi_iso_elas3D::Grandeur_particuliere et Loi_iso_elas3D::ListeGrandeurs_particulieres

Les lois de comportement : Héritage



LoiAbstraiteGeneral :
La classe générique de toutes les lois pas forcément mécanique

Loi_comp_abstraite :
Classe générique des lois spécifiquement mécanique

Loi_iso_elas1D :
Une loi particulière

Utilisation typique

```
// algorithme global
AlgoriNonDyna::CalEquilibre
   .....
   // calcul de la matrice de raideur et du second membre global
   Algori::RaidSmEner
    ....
    // calcul de la matrice de raideur et du second membre local (pour un élément)
    Element::ResRaid resu = el.Calcul_implicit(pa);
    ....
     // calcul spécifique au cas implicit en mécanique
     ElemMeca::Cal_implicit
        ....
        // calcul des contraintes et gradient : appel général
        loiComp→Cal_implicit
           ....
           // cas d'une loi particulière
           Loi_iso_elas1D::Calcul_DsigmaHH_tdt
```

La classe mère :

LoiAbstraiteGeneral

...regroupe les données générales permettant un choix typé
l'accès s'effectue via uniquement des méthodes dédiées → encapsulage total

```
protected :
// ---- donnees protegees
Enum_comp id_comp;   // identificateur de la loi de comportement
int dim ; // dimension de la loi
Enum_categorie_loi_comp categorie_loi_comp;
// identificateur donnant le type de degré de liberté
// de base, ou est calculé la loi, par exemple pour une loi mécanique pt d'integ SIG11
Enum_ddl enu_ddl_type_pt;
```

Et les méthodes communes

```
virtual void Affiche()  const = 0; // affichage de la loi (virtuelle pure )
// I/O base info ( .BI)
virtual void Lecture_base_info_loi
virtual void Ecriture_base_info_loi(ofstream& sort,const int cas) =0;
// affichage et definition interactive des commandes particulières à chaques lois
virtual void Info_commande_LoisDeComp(UtilLecture& lec) = 0;
virtual int TestComplet() ;  // test si la loi est complete (virtuelle, avec une def par défaut)
```

La classe générique pour les lois en mécanique :
Loi_comp_abstraite

Les données principales:

protected : // accessible par les classes dérivées

→ pour la gestion de la thermodépendance :

```
bool thermo_dependant; //  indique si oui ou non la loi dépend de la température
double temperature_0,temperature_t,temperature_tdt; // variables valides que si l'on est thermo_dependant utilisée par les classes dérivées
double* temperature; // pointeur sur la température de travail (à 0, à t ou tdt)
bool dilatation; // variable interne, qui est mise en route par l'appel de certaine fonction, comme celles
```

→ on utilise en générale des infos issues de la déformation (Classe Deformation)

```
Enum_type_deformation type_de_deformation;// indication du type de déformation utilisée
Deformation * def_en_cours; // si différent de NULL, indique une déformation qui peut être utilisée par les classes dérivées
```

→ associé à la loi de comportement, en général (pas obligatoire) on a des données/résultats spécifiques stockées typiquement au pti ces valeurs sont gérées par une classe spécifique à la loi, issue de la classe générique SaveResul

```
SaveResul * saveResul; // pointeur de travail utilise par les classes derivantes
```

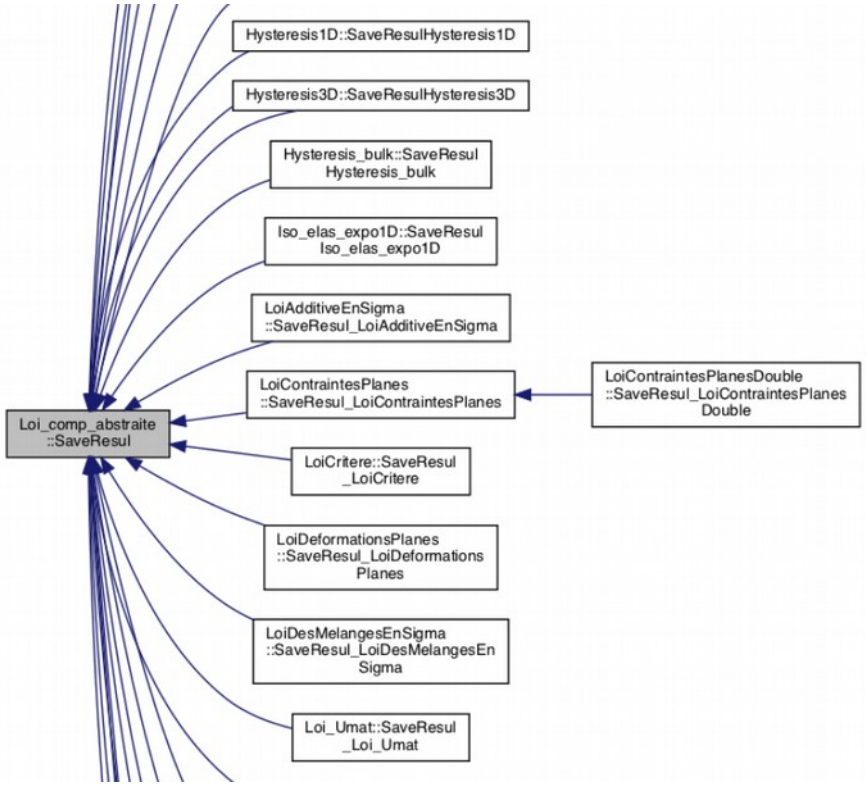
→ gestion de cas particuliers

```
bool comp_tangent_simplifie;// indic pour définir si oui ou non on utilise un comportement tangent simplifié
bool utilise_vitesse_deformation; // indication si l'on utilise ou pas la vitesse de déformation
```

→ pour la gestion des temps de calcul

```
Temps_CPU_HZpp  temps_loi; // spécifique à ce type de loi: cumule tous les appels
```

Conteneur et gestion des données spécifiques au pti

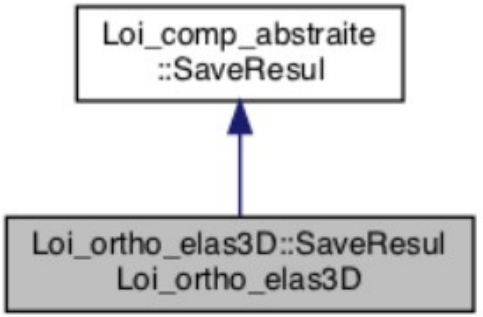


SaveResul :

D'une manière générale, est définie dès que l'on veut sauvegarder au pti, une grandeur spécifique à la loi

→ existe pour pratiquement toutes les lois

Exemple :



SaveResul :

- **Pas de données propres**
- **Des méthodes génériques** (virtuelles)
- **s'utilise avec la loi via des méthodes** (c'est la loi qui sait comment s'en servir)

```
virtual SaveResul * Nevez_SaveResul() const =0;  // définition d'une nouvelle instance identique
virtual SaveResul & operator = ( const SaveResul &) = 0; // affectation
  //============= lecture écriture dans base info ==========
virtual void Lecture_base_info (ifstream& ent,const int cas)  = 0;
virtual void Ecriture_base_info(ofstream& sort,const int cas) = 0;
// mise à jour des informations transitoires en définitif s'il y a convergence
virtual void TdtversT() {};
virtual void TversTdt() {};
// affichage à l'écran des infos
virtual void Affiche() const = 0;
//changement de base de toutes les grandeurs internes tensorielles stockées
virtual void ChBase_des_grandeurs(const Mat_pleine& beta,const Mat_pleine& gamma) = 0;
// procedure permettant de completer éventuellement
virtual SaveResul* Complete_SaveResul(const BlocGen & bloc, const Tableau <Coordonnee>& tab_coor
                        ,const Loi_comp_abstraite* loi) = 0;
virtual int TestComplet()const {return 1;};  // test si le conteneur est complet
// --- gestion d'une map de grandeurs quelconques éventuelles ---
virtual const map <  EnumTypeQuelconque , TypeQuelconque, std::less < EnumTypeQuelconque> >*
Map_type_quelconque() const  {return NULL;};
```

Mise en œuvre et utilisation de SaveResul

Stockage et gestion globale dans ElemMeca :

1) **données**

```
Tableau <Loi_comp_abstraite::SaveResul *> tabSaveDon; // donnée particulière à la loi mécanique
```

2) **utilisation**

Passage de l'info à la loi de comp au moment de son utilisation par ElemMeca: ex :

loiComp→Cal_implicit(tabSaveDon(ni), etc.

3) **I/O** ex : sur Base Info (.BI) : Utilisation des méthodes génériques Lecture_base_info et Ecriture_base_info de SaveResul

4) **mise à jour** de t→ tdt ou l'inverse : Utilisation des méthodes génériques TdtversT et TversTdt

Initialisation spécifique via la définition de la loi → s'effectue dans un élément particulier (Hérite d'ElemeMeca)

Ex : HexaMemb::DefLoi (LoiAbstraiteGeneral * NouvelleLoi)

```
// cas d'une loi mécanique
if (GroupeMecanique(NouvelleLoi->Id_categorie()))
 {loiComp = (Loi_comp_abstraite *) NouvelleLoi;
  // initialisation du stockage particulier, ici en fonction du nb de pt d'integ
  int imax = tabSaveDon.Taille();
  for (int i=1;i<= imax;i++) tabSaveDon(i) = loiComp->New_et_Initialise();
  // idem pour le type de déformation mécanique associé
  int iDefmax = tabSaveDefDon.Taille();
  for (int i=1;i<= iDefmax;i++) tabSaveDefDon(i) = def->New_et_Initialise();
  // définition du type de déformation associé à la loi
  loiComp->Def_type_deformation(*def);
  // on active les données particulières nécessaires au fonctionnement de la loi de comp
  loiComp->Activation_donnees(tab_noeud,dilatation,lesPtMecaInt);
 };
```

Suite Loi_comp_abstraite

```
// les méthodes centrales publiques
   virtual const Met_abstraite::Expli& Cal_explicit_t  // schema de calcul explicite à t
   virtual const Met_abstraite::Expli_t_tdt& Cal_explicit_tdt  // schema de calcul explicite à tdt
   virtual const Met_abstraite::Impli& Cal_implicit   // schema implicit
   virtual void Cal_flamb_lin// schema pour le flambage linéaire
   virtual void ComportementUmat     // schema pour le calcul de la loi de comportement dans le cas de  l'umat
// associées aux méthodes spécifiques protégées :
  virtual void Calcul_SigmaHH // calcul des contraintes
  virtual void Calcul_DsigmaHH_tdt // calcul des contraintes et ses variations  par rapport aux ddl a t+dt
  virtual void Calcul_dsigma_deps // calcul des contraintes et ses variations  par rapport aux déformations a t+dt
```

Ex plus détaillé, cas d'un schéma implicite utilisé par ElemMeca :

1) passage de paramètres :

```
  // schema implicit
  virtual const Met_abstraite::Impli& Cal_implicit
            ( Loi_comp_abstraite::SaveResul * saveDon ,Deformation & def,DdlElement & tab_ddl
              ,PtIntegMecaInterne& ptintmeca, Tableau <TenseurBB *>& d_epsBB_tdt,double& jacobien
              ,Vecteur& d_jacobien_tdt,Tableau <TenseurHH *>& d_sigHH,const ParaAlgoControle & pa
              ,CompThermoPhysiqueAbstraite::SaveResul * saveTP,CompThermoPhysiqueAbstraite* loiTP
              ,bool dilatation,EnergieMeca & energ,const EnergieMeca & energ_t,bool premier_calcul
             );
```

Exe : Loi_comp_abstraite::Cal_implicit

Les principales étapes :

- gestion des temps cpu : mise en route des compteurs ex :
  temps_cpu_loi.Mise_en_route_du_comptage(); // spécifique pti
- // passage des infos specifiques aux classes derivantes : on alimente des pointeurs
  saveResul = saveDon; def_en_cours=&def;
- récup des infos au pti via le conteneur ptintmeca : ex :
  TenseurHH & sigHH = *(ptintmeca.SigHH()); // ici la grandeur finale
- calcul des déformations et métriques au pti via le pointeur de déformation
  const Met_abstraite::Impli& ex = def.Cal_implicit(…
- si thermodépendance : récupération des infos de température
- si prise en compte de la dilatation : gestion et calcul des def associées def.DeformationThermoMecanique(…
- CalculInvariants_cinematique(...// calcul éventuel des invariants liés à la cinématique
- calcul de grandeurs préalable nécessaires ex : interpolation de valeurs aux nœuds : CalculGrandeurTravail(…
- **appel de la méthode spécifique à la loi → calcul des contraintes et de l'opérateur tangent**
  Calcul_DsigmaHH_tdt (…
-// calcul éventuel des invariants de contraintes
  CalculInvariants_contraintes(
- fin du comptage cpu temps_loi.Arret_du_comptage();

Exemple d'une loi de Hooke 3D : Loi_iso_elas3D::Calcul_DsigmaHH_tdt

```
// calcul des contraintes a t+dt et de ses variations
void Loi_iso_elas3D::Calcul_DsigmaHH_tdt (...
 - passage des tenseurs génériques en tenseurs spécifiques → permet d'utiliser les méthodes spécifiques, ex :
Tenseur3HH & sigHH = *((Tenseur3HH*) &sigHH_tdt);  //   on fait un cast
- calcul des paramètres de la loi si dépendance à une courbe ou une fct nD : ex pour E :
 if (E_temperature != NULL) {E = E_temperature->Valeur(*temperature);}
 else if (E_nD != NULL) // on utilise la méthode générique de loi abstraite
    { ...Tableau <double> & tab_val = Loi_comp_abstraite::Loi_comp_Valeur_FnD_Evoluee(...
     E = tab_val(1); // on récupère le premier élément du tableau uniquement  };
 - // sauvegarde des paramètres matériaux
   SaveResulLoi_iso_elas3D & save_resul = *((SaveResulLoi_iso_elas3D*) saveResul);
   save_resul.E = E;  save_resul.nu = nu;
- calcul des contraintes :
   Tenseur3BH  epsBH = epsBB * gijHH;  // deformation en mixte
   // calcul des coefficients
   double coef1 = (E*nu)/((1.-2.*nu)*(1+nu));
   double coef2 = E/(1.+ nu);
   // calcul de la trace des deformations
   double Ieps = epsBH.Trace();
.... sigBH = (Ieps * coef1) * IdBH3 + coef2 * epsBH ; // contrainte en mixte
   sigHH = gijHH * sigBH;  // en deux fois contravariant
```

$$\boldsymbol{\sigma} = \left(\frac{(E\ \nu)}{((1. - 2.\nu)\ (1+\nu))} \boldsymbol{I}_{\varepsilon} \right) \boldsymbol{I} + \frac{E}{(1+\nu)}\ \varepsilon$$

Suite : Loi_iso_elas3D::Calcul_DsigmaHH_tdt

```
energ.ChangeEnergieElastique(0.5 * (sigHH && epsBB)); // -- traitement des énergies
// -- calcul des modules
module_compressibilite = E/(3.*(1.-2.*nu));
module_cisaillement = 0.5 * coef2;
```

$$K = \frac{E}{3(1-2\,\nu)} \; et \; G = \frac{E}{2(1+\nu)}$$

- calcul de l'opérateur tangent

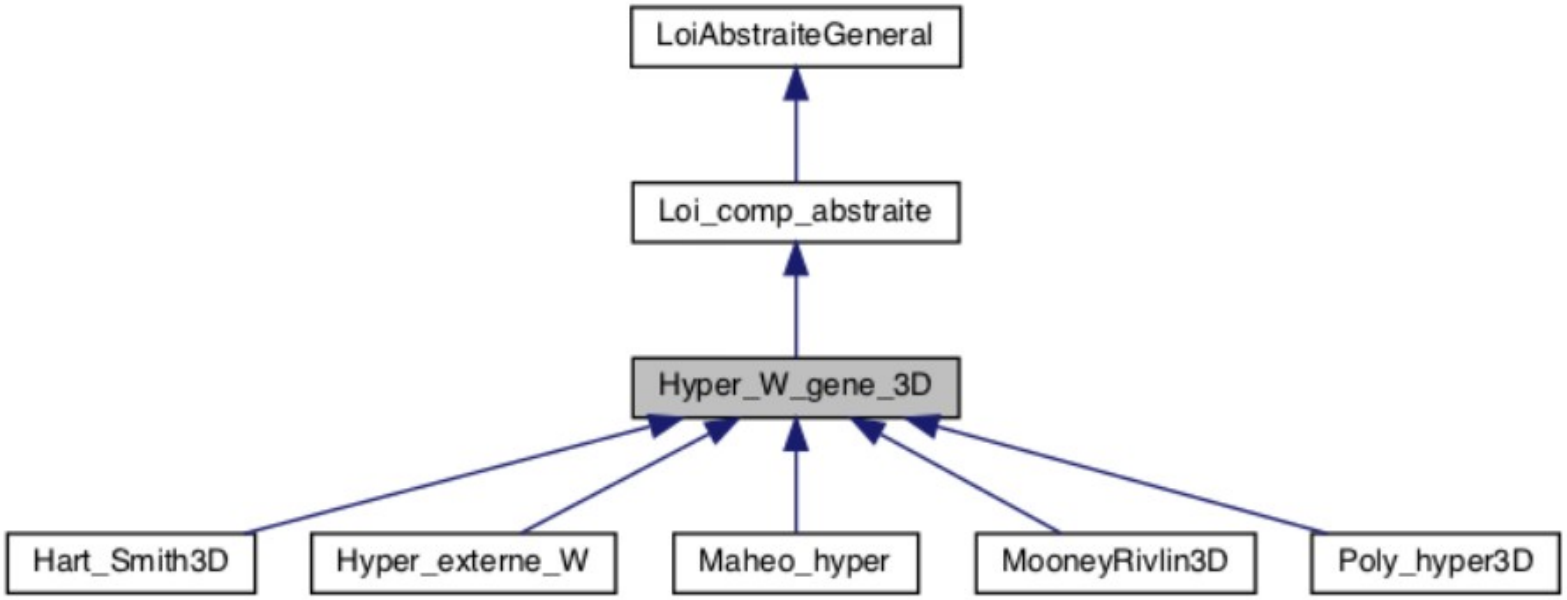
```
// cas le la variation du tenseur des contraintes
    for (int i = 1; i<= nbddl; i++)
    { // on fait de faire uniquement une égalité d'adresse
      Tenseur3HH & dsigHH = *((Tenseur3HH*) (d_sigHH(i)));  // passage en dim 3
      const Tenseur3HH & dgijHH = *((Tenseur3HH*)(d_gijHH_tdt(i))) ; // pour simplifier l'ecriture
      const Tenseur3BB & depsBB = *((Tenseur3BB *) (d_epsBB(i))); //    "
      // pour chacun des ddl on calcul les tenseurs derivees
      Tenseur3BH depsBH =  epsBB * dgijHH  + depsBB * gijHH ;
      double dleps = depsBH.Trace();
      ... dsigHH = dgijHH * sigBH + gijHH *
          ((dleps * coef1) * IdBH3 + coef2 * depsBH);
       };
```

Ajout d'une nouvelle loi : exemple d'implantation

Préalable : on dispose des éléments théoriques associés à la nouvelle loi Loi_toto_elas
→ fonctionnement de la loi : calcul des contraintes et des opérateurs tangents
→ type de réponse qui est attendue : permettra de valider le fonctionnement de la loi

1) on définie et ajoute un énuméré spécifique à la loi dans enum Enum_comp
   Exe : ISOELAS2D_C, TOTO_ELAS... on met à jour les méthodes spécifiques cf. Enum_comp.h
2) On duplique les fichiers d'une loi déjà existante (si possible d'un fonctionnement analogue)
   → **Loi_toto_elas.h** et **Loi_toto_elas.cc**
   Dans ces fichiers on change systématiquement la chaine : iso_elas2D_C en toto_elas
   On ajoute les deux fichiers à la liste des fichiers gérés par le projet Herezh
3) Dans le fichier LesLoisDeComp.cc on ajoute :
   `#include "Loi_toto_elas.h"`
…et on complète les différentes occurrences des lois : ex :
   `pt = new Loi_toto_elas ();list_de_loi.push_back(pt); // pour la loi toto elas`
4) On passe en revue toutes les méthodes virtuelles qui sont définies dans Loi_comp_abstraite
Et on les adapte en fonction des éléments théoriques associés à la nouvelle loi
5) compilation : suppression des bugs d'écriture, édition de lien
5) def d'un cas test basique via Herezh en mode interactif
6) vérification que le calcul tourne
7) création d'un .Cvisu
8) vérification que les résultats sont conformes aux attendus théorique

Précisions sur les lois hyperélastiques utilisant les invariants de B



`Hyper_W_gene_3D :`

Utilisation d'une classe intermédiaire qui globalise des méthodes communes aux différents potentiels

`Hyper_W_gene_3D::SaveResulHyper_W_gene_3D`

```
Infos spécifiques au pti pour toutes les classes dérivées
→ permet d'éviter une def spécifique d'un SaveResul dans chaque classe dérivées
```

Méthodes spécifiques d' Hyper_W_gene_3D :

- calculs à partir des éléments de métriques (gij)

```
void Invariants_et_var1 // calcul des invariants et de leurs variations premières
void Invariants_et_var2 // calcul des invariants et de leurs variations premières et secondes
// --- acces en lectures  aux différentes grandeurs calculées  par Invariants_et_var1 et Invariants_et_var2, ex :
const double& I__B() const {return I_B;};
const Tableau <Tenseur3HH>& D_J_r_epsBB_HH() const {return d_J_r_epsBB_HH;};
// --- acces en lectures  aux différentes grandeurs calculées  uniquemenent par  Invariants_et_var2, ex :
const Tenseur3HHHH&  D_J_1_eps2BB_HHHH () const {return d_J_1_eps2BB_HHHH;};
```

- implantation des méthodes virtuelles de `Loi_comp_abstraite`

```
// gestion des I/O grandeurs quelconques :
virtual void Grandeur_particuliere // récupération des grandeurs particulière (hors ddl )
virtual void ListeGrandeurs_particulieres  // récupération de la liste de tous les grandeurs particulières

virtual void Activation_stockage_grandeurs_quelconques(list <EnumTypeQuelconque >& listEnuQuelc);
virtual void Insertion_conteneur_dans_save_result(SaveResul * saveResul);
```

- stockage spécifique :

```
class SaveResulHyper_W_gene_3D: public Loi_comp_abstraite::SaveResul
  // données optionnelles : les paramètres matériaux réellement utilisés
  Vecteur *para_loi, *para_loi_t;
  // également des grandeurs qui sont éventuellement créées si on le demande
  Hyper_W_gene_3D::Invariantpost3D * invP, * invP_t;
  //  map de grandeurs quelconques particulière qui peut servir aux classes appelantes
  map <  EnumTypeQuelconque , TypeQuelconque, std::less < EnumTypeQuelconque> > map_type_quelconque;
  //r
```

Exemple d'utilisation : MooneyRivlin3D::Calcul_DsigmaHH_tdt

- Mise à jour éventuelle des paramètres de la loi, si dépendant d'une courbe ou une fct nD
Et sauvegarde dans le SaveResul
→ K_use , C01_use , C10_use

```
Hyper_W_gene_3D::Invariants_et_var2 // calcul des invariants et de leurs variations premières et secondes
// calcul du potentiel et de ses dérivées premières et secondes / aux invariants J_r
// méthode spécifique à Mooney Rivlin
this->Potentiel_et_var2(module_compressibilite);
// calcul du tenseur des contraintes
double unSurV=1./V;
sigHH = ((W_d_J1+W_c_J1)/V) * d_J_r_epsBB_HH(1) + (W_d_J2/V) * c
        + ((W_v_J3+W_c_J3)/V) * d_J_r_epsBB_HH(3);
```

$$\boldsymbol{\sigma} = \frac{\sqrt{g_0}}{\sqrt{g}} < W_{,J_i} > (\boldsymbol{J}_{,\boldsymbol{\epsilon}})_{..} = \frac{1}{V} < W_{,J_i} > (\boldsymbol{J}_{,\boldsymbol{\epsilon}})_{..}$$

---------- méthode spécifique à Mooney Rivlin ----------

```
// calcul du potentiel et de ses dérivées premières et secondes / aux invariants J_r
void MooneyRivlin3D::Potentiel_et_var2(double & module_compressibilite)
 { // calcul de grandeurs intermédiaires
   double unSurV=1./V;  double logV = log(V);
   W_d = C10_use * (J_r(1)-3.) + C01_use * (J_r(2)-3.); // partie déviatorique
   // calcul des variations premières non nulles du potentiel
   // les variations secondes sont nulles
   W_d_J1 = C10_use;  // variation / J1
   W_d_J2 = C01_use;  // variation / J2
```

Application : Introduction d'une loi hyperélastique utilisant le potentiel de Gent

Source :
https://fr.wikipedia.org/wiki/Hyper%C3%A9lasticit%C3%A9

Modèle de Gent [modifier | modifier le code]

Contrairement au modèle Néo-hookéen et de Mooney-Rivlin qui montre une certaine linéarité en grande déformation, le modèle de Gent (en) reproduit correctement le durcissement important des élastomères avant rupture dû à la limite d'extensibilité des chaînes macromoléculaires[4]. Gent a imaginé une densité d'énergie de déformation présentant une singularité quand le premier invariant atteint une valeur limite notée I_m

$$W(I_1) = -\mu(I_m - 3).ln\left(1 - \frac{I_1 - 3}{I_m - 3}\right)$$

Les paramètres de ce modèle sont donc μ, module d'élasticité, et $J_m = I_m - 3$.

En 2003, Cornelius O. Horgan, chercheur au département de Génie Civil de l'Université de Virginie (USA), et Giuseppe Saccomandi, chercheur au département d'Ingénierie de l'Innovation à l'Université de Lecce en Italie, ont prouvé[8] que le modèle de Gent, originellement phénoménologique, est analogue au modèle d'Arruda-Boyce à huit chaînes. Ainsi, le modèle de Gent est relié à la microstructure du matériau selon les relations suivantes :

$$\mu = nkT$$

$$J_m = 3(N - 1).$$

avec μ le module d'élasticité du modèle de Gent, J_m un scalaire ayant le rôle de limite d'extensibilité, n la densité de chaînes et N le nombre moyen de monomères par chaîne.

→ On partira d'une des lois existantes utilisant les invariants de Mooney Rivlin c-a-d ceux de **B**

Sortie des résultats - visualisation

Gérée globalement par un classe principale associée à chaque sortie particulière

- Visualisation_maple → sortie de type tableau (utilisable par gnuplot pas ex)
- Visualisation_Gmsh → sortie de fichiers utilisables par gmsh
- Visualisation_Gid → sortie de fichiers utilisables par gid
- Visualisation → sortie de fichiers utilisables par un lecteur vrml

Etc.

Classes de structure assez similaire

→ la sortie maple va servir d'illustration dans une première étape

Utilisations :

- déclaration dans Algori (= la classe générique de tous les algo globaux)

  Visualisation_maple visualise_maple; // instance sur les utilitaires de visualisation en maple

- sortie spécifique de type maple : Algori::Visu_maple (utilise les méthodes de Visualisation_maple)
- sortie au fil du calcul : Algori::VisuAuFilDuCalcul (" ")

NB : Algori::Visu_maple est utilisé après le calcul, d'où avec la sauvegarde .BI si on veut des incréments différents de l'incrément final

Algori::VisuAuFilDuCalcul est utilisé pendant le calcul donc ne se sert pas des valeurs sauvegardées mais uniquement des valeurs en cours : dans certain cas on a accès alors à un choix plus large de grandeurs

Algori::VisuAuFilDuCalcul

```
tempsSortieFilCalcul.Mise_en_route_du_comptage(); // temps cpu
- si c'est le premier incrément : opération d'initialisation
  . entreePrinc->Ouverture_CommandeVisu("lecture"); ouverture fichier .Cvisu
  . visualise_maple.List_maillage_disponible(lesMaillages->NbMaillage());
- if  (!(pa.SauvegardeFilCalculAutorisee … → arrêt sortie
- visualise_maple.List_balaie_init(list_incre_init);
- visualise_maple.Initialisation(... //  initialisation
- if (type_incre == OrdreVisu::PREMIER_INCRE) si  premier  increment :
  . visualise_maple.Lecture_parametres_OrdreVisu(*entreePrinc);
   . // initialisation de la sortie maple (fichier) si la sortie maple est valide
      if (visualise_maple.Visu_maple_valide())
       { entreePrinc->Ouverture_fichier_principal_maple();// initialisation de la maple (fichier)
        visualise_maple.Contexte_debut_visualisation(); } ;// == définition des paramètres d'entête de visualisation
- if (visualise_maple.Visu_maple_valide())
         {visualise_maple.Visu(… } ;
 - if (type_incre == OrdreVisu::DERNIER_INCRE)
      {visualise_maple.Contexte_fin_visualisation(); // == définition des paramètres de fin de visualisation
 - tempsSortieFilCalcul.Arret_du_comptage(); // temps cpu
```

→ **qq éclaississements sur** entreePrinc

Classe UtilLecture
gestion de base de toutes les I/O

Sert pour les I/O : principalement sur fichier, mais également qq méthodes pour I/O interactives

- lecture/écriture du .info, .Cvisu, .BI, .PI
- écriture : .maple, .pos, etc. : tous les fichiers de sortie de post-traitement

Les opérations typiques :

- ouverture et/ou création de fichiers et retour d'un pointeur sur le fichier : si fichier déjà ouvert, uniquement retour de pointeur
- fermeture fichier
- création du répertoire de résultats
- lecture d'une ligne sur fichier avec filtrage des commentaires, réorientation vers un sous-fichier lorsque l'on a une indirection : < toto
- lecture d'une ligne sur clavier
- gestion des erreurs d'I/O → génération d'exception : ex : class ErrNouvelleDonneeCVisu
- méthodes utilitaires :
  . lecture d'une constantes utilisateurs
  . écriture entête interactive début Herezh
  . lecture d'un mot clé suivi éventuellement d'une valeur ou d'une chaine de caractère

Utilisation typique : exemple 1

```
Hypo_ortho3D_entrainee::LectureDonneesParticulieres (UtilLecture * entreePrinc, ….

…….
if(strstr(entreePrinc->tablcar,"avec_parametres_de_reglage_")!=0) // --- lecture éventuelle des paramètres de réglage ----
   {string nom;  entreePrinc->NouvelleDonnee(); // on se positionne sur un nouvel enreg
    // on lit tant que l'on ne rencontre pas la ligne contenant "fin_parametres_reglage_" ou un nouveau mot clé global auquel cas il y a pb !!
    MotCle motCle; // ref aux mots cle
    while (strstr(entreePrinc->tablcar,"fin_parametres_reglage_")==0)
    { // si on a  un mot clé global dans la ligne courante c-a-d dans tablcar --> erreur
     if ( motCle.SimotCle(entreePrinc->tablcar))
      { cout << "\n erreur de lecture ….
        entreePrinc->MessageBuffer("** erreur5 des parametres de reglage de la loi de comportement de Hypo_ortho3D_entrainee **");
        throw (UtilLecture::ErrNouvelleDonnee(-1)); Sortie(1);} ;
     *(entreePrinc->entree) >> nom; // lecture d'un mot clé
      if ((entreePrinc->entree)->rdstate() == 0) {} // lecture normale
     else  if ((entreePrinc->entree)->fail())
       // on a atteind la fin de la ligne et on appelle un nouvel enregistrement
      {   entreePrinc->NouvelleDonnee();  // lecture d'un nouvelle enregistrement
          *(entreePrinc->entree) >>nom; } …

     if (nom == "type_transport_")  // cas de la lecture du type de transport
     { *(entreePrinc->entree) >> type_transport;
       if ((type_transport!=0)&&(type_transport!=1))
        { cout << "\n le type de transport lue   "<< type_transport << " n'est pas acceptable …...
          type_transport = 0;
        };} ;
```

Exemple 2

```
// Lecture des donnees de la classe sur fichier
void Projection_anisotrope_3D::LectureDonneesParticulieres (UtilLecture * entreePrinc, ...
 {string nom_class_methode("Projection_anisotrope_3D::LectureDonneesParticulieres");
  double val_defaut=0.; double min = 0.; double max = -1; // max < min => la condition n'est pas prise en compte
  string nom_pour_erreur("**erreur en lecture** Projection_anisotrope_3D:");

  // -- lecture du type de projection on lit le nom du type de projection
  {string nom_proj; string mot_cle("TYPE_PROJ_ANISO_");
   bool  lec = entreePrinc->Lecture_mot_cle_et_string(nom_class_methode,mot_cle,nom_proj);
   if (!lec )
     { entreePrinc->MessageBuffer(nom_pour_erreur+" mot cle "+mot_cle);
       throw (UtilLecture::ErrNouvelleDonnee(-1));
       Sortie(1);
     };
   if ( Type_Enum_proj_aniso_existe(nom_proj))
   { type_projection = Id_Nom_proj_aniso(nom_proj);
   }
   else
     { entreePrinc->MessageBuffer(nom_pour_erreur+" type de projection inexistant: "+mot_cle);
       throw (UtilLecture::ErrNouvelleDonnee(-1));
       Sortie(1);
     };
  };
```

Retour sur Visualisation_maple

- méthodes principales d'activation de la visualisation

```
void Initialisation   // initialisation des ordres disponibles
// par exemple pour les isovaleurs on définit la liste des isovaleurs disponibles et les extrémas
void Visu  // méthode principale pour activer la visualisation
```

- Préparation de la sortie de visualisation

```
void List_increment_disponible // information de l'instance de la liste d'incrément disponible pour la visualisation
const list<int> &  List_balaie_init  // indique le choix de la liste d'incrément à utiliser pour l'initialisation
void List_maillage_disponible  // information de l'instance du nombre de maillages disponibles pour la visualisation
```

- écriture sur le . maple

```
void Contexte_debut_visualisation();  // == définition des paramètres de visualisation  titre, navigation, éclairage
void Contexte_fin_visualisation();  // (points de vue) et enchainement si nécessaire
```

- gestion des I/O sur le .CVisu

```
void Lecture_parametres_OrdreVisu // lecture des paramètres de l'ordre dans un flux
void Ecriture_parametres_OrdreVisu // écriture des paramètres de l'ordre dans un flux
void EcritDebut_fichier_OrdreVisu  // écriture de l'entête du fichier de commande de visualisation
void EcritFin_fichier_OrdreVisu // écriture de la fin du fichier de commande de visualisation
```

- pour la gestion interactive de la visualisation : construction du .Cvisu

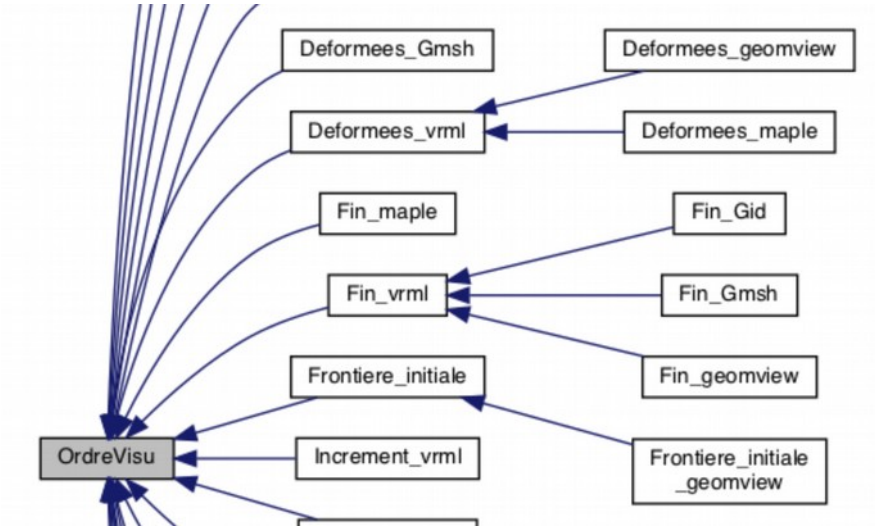
```
int OrdresPossible(); // affichage des différentes possibilités (ordres possibles)
```

Suite Visualisation_maple

```
private :  // VARIABLES PROTEGEES :
  UtilLecture* entreePrinc; // gestion des entrees/sorties
  list <OrdreVisu*> ordre_possible; // l'ensemble des ordres possibles
  OrdreVisu* fin_o; // ordre de fin de la visualisation
  OrdreVisu* visuali; // ordre de visualiser
  OrdreVisu* choix_inc; // ordre de choix des incréments
  OrdreVisu* anim; // ordre d'animation
  list <int>  list_incre; // liste des incréments possibles
  const list <int>*  list_balaie; // liste des incréments à visualiser
  int nb_maillage_dispo; // le nombre de maillage disponible
  OrdreVisu* choix_mail; // ordre de choix des maillages
  OrdreVisu* choix_grandeurs; // choix des grandeurs à visualiser
  bool animation; // indication si l'on est en animation ou pas
  bool activ_sort_maple;// indication si la visualisation de type
      // maple est active ou pas, par défaut = faux
```

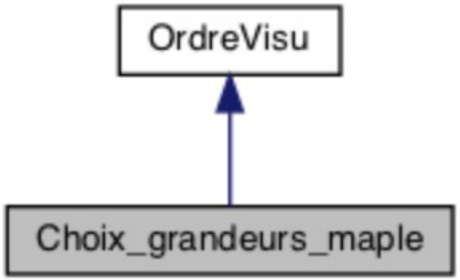
→ utilisation de la classe `OrdreVisu` pour la majorité des ordres de visualisation

→ classe virtuelle



Exemple : Choix_grandeurs_maple

- les principales méthodes
  void Initialisation // initialisation de l'ordre
  void Init_liste_grandeurs // initialisation de la liste des différentes grandeurs possibles à visualiser
  void ChoixOrdre(); // choix de l'ordre,
  void ExeOrdre // execution de l'ordre
  void Entete_fichier_maple // écriture des informations d'entête,
  void Lecture_parametres_OrdreVisu // lecture des paramètres de l'ordre dans un flux
  void Ecriture_parametres_OrdreVisu // écriture des paramètres de l'ordre dans un flux



Utilisation par Visualisation_maple::Visu

```
{  // récup de la liste de maillage (même méthode que pour vrml)
   const list <int>& li_mail =  ((ChoixDesMaillages_vrml*)choix_mail)->List_choisit();
   ...
   // ici on passe en revue l'ensemble des ordres possibles
   list <OrdreVisu*>::iterator s_or,s_or_fin; s_or_fin = ordre_possible.end();
  ....
   for (s_or = ordre_possible.begin();s_or != s_or_fin; s_or++)
      { if ((*s_or) != anim) // l'animation n'est appelé qu'à la fin
          (*s_or)→ExeOrdre(...
  …
```

Globalement : Choix_grandeurs_maple::ExeOrdre

```
// on balaie la liste des grandeurs a visualiser, toutes les données sont sur une même ligne
// --- tout d'abord les grandeurs globales
   ...
// --- puis les torseurs de reaction
   ....
Sortie_somme_moy_N // --- puis les grandeurs sommes, moyennes, maxi etc.
Sortie_somme_moy_E  // sortie pour les  ref E
Sortie_somme_moy_face_E   // sortie pour les  ref de face E
Sortie_somme_moy_arete_E  // sortie pour les  ref d'arete E
 // --- puis les grandeurs aux noeuds
...
// --- puis les grandeurs aux éléments
…
// --- puis les grandeurs aux faces d'éléments
...
// --- puis les grandeurs aux arêtes d'élément
...
```

Exemple des grandeurs aux noeuds : Choix_grandeurs_maple::ExeOrdre

```
// ---  les grandeurs aux noeuds
...
for (int n_mail = 1;n_mail<=nb_maillage;n_mail++)
 {....
  // écriture du temps que si il y a des noeuds et que l'on n'est pas avec un seul incrément
   ... sort << setprecision(nbdigit) << charge->Temps_courant() << " ";} // le temps
  // maintenant on affiche les infos pour la liste totale de nœuds
  for (inoeud = tab_total_noeud.begin();inoeud!=inoeudfin;inoeud++)
  { Noeud& noe=lesMail->Noeud_LesMaille(n_mail,*inoeud);
    … Coordonnee ncor =  noe.Coord2() ; // cas normal:   coordonnées du noeud à tdt
     for (int ic=1;ic<=ncor.Dimension();ic++) sort << setprecision(nbdigit) << ncor(ic) << " ";
    // .... sortie des ddl principaux .....
    {List_io<Ddl_enum_etendu>::iterator ienu,ienufin=tabnoeud_type_ddl_retenu(n_mail).end();
     switch(type_sortie_ddl_retenue)
     { case 0: // cas où on ne sort que la grandeur
        … val_a_sortir = noe.Valeur_tdt((*ienu).Enum()); ...  sort << setprecision(nbdigit) << val_a_sortir << " ";
    // .... sortie des ddl étendus .....
    ... if (noe.Existe_ici_ddlEtendu((*ienu))) {val_a_sortir = noe.DdlEtendue(*ienu).ConstValeur();}
         sort << setprecision(nbdigit) << val_a_sortir << " ";
    // .... sortie des grandeurs quelconque aux noeud .....
    {List_io<TypeQuelconque>::iterator ienu,ienufin=tabnoeud_TypeQuelconque_retenu(n_mail).end();
     for (ienu = tabnoeud_TypeQuelconque_retenu(n_mail).begin();ienu!=ienufin;ienu++)
      { TypeQuelconque_enum_etendu enuq = (*ienu).EnuTypeQuelconque();// récup de l'énuméré
        ... const TypeQuelconque& grand_quelconque_noe = noe.Grandeur_quelconque(enuq);
        ... grand_quelconque_noe.Grandeur_brut(sort,ParaGlob::NbdigdoGR());
```

Résumé des sorties

La compréhension du mécanisme de sortie est nécessaire si on veut corriger une erreur. **Mais….**
l'implantation est faite de manière à ne pas faire de référence explicite à des grandeurs spécifiques
Seule le type de grandeur est utilisé et les méthodes associées génériques
- si on ajoute un nouveau ddl, ou un Ddl_enum_etendu ou un nouveau TypeQuelconque il sera automatiquement correct pour une sortie, quelque soit la sortie : maple, gmsh, gid …
- de même les mécanismes de sortie aux pti, d'interpolation pour passage aux nœuds … utilisent que des méthodes génériques.

Principe de fonctionnement : la classe de visualisation récolte avant utilisation les grandeurs possibles ex :

```
void Choix_grandeurs_maple::Init_liste_grandeurs(LesMaillages * lesMail,LesCondLim*,LesContacts* lesContacts,bool fil_calcul)
   {  List_io<TypeQuelconque> liste_inter = lesContacts→ListeGrandeurs_particulieres(absolue);
     // récupération des ddl présents dans les maillages aux nœuds, les éléments
     tabnoeud_type_ddl = lesMail->Les_type_de_ddl_par_noeud(absolue);
     tabnoeud_type_ddlEtendu = lesMail→Les_type_de_ddl_etendu_par_noeud(absolue);
     tabnoeud_TypeQuelconque = lesMail->Les_type_de_TypeQuelconque_par_noeud(absolue);
     tabelement_type_ddl = lesMail→Les_type_de_ddl_par_element(absolue);
     tabelement_typeParti = lesMail->Les_type_de_donnees_particulieres_par_element(absolue);
     // idem mais sous forme de grandeurs évoluées
     tabelement_evoluee = lesMail->Les_type_de_donnees_evolues_internes_par_element(absolue);
      // pour les faces et arêtes
     tab_F_element_TypeQuelconque = lesMail->Les_type_de_donnees_evolues_internes_par_face_element(absolue);
     tab_A_element_TypeQuelconque = lesMail→Les_type_de_donnees_evolues_internes_par_arete_element(absolue);
    …..
```

Exemple :
lesMail->Les_type_de_donnees_particulieres_par_element(absolue);

- au niveau maillage

```
Tableau <List_io <TypeQuelconque> > LesMaillages::Les_type_de_donnees_particulieres_par_element(bool absolue)
 { … // on boucle sur les maillages
   for (int i1 = 1; i1<= nbMaillageTotal; i1++)
   { ... // on boucle sur les elements
     for (int i2 = 1; i2 <= nbelementmax; i2++)
       { Element * nee = & Element_LesMaille(i1,i2); // recup du element
         ... List_io <TypeQuelconque> tab_enum_parti(nee->Les_types_particuliers_internes(absolue));
          ... tab_enu_final.merge(tab_enum_parti); // ajout de la liste
   ...return tab_ret_enum; // retour du tableau
```

- **au niveau d'un élément** ex : un quadrangle (QuadraMemb → classe générique des quadrangles

```
List_io <TypeQuelconque> QuadraMemb::Les_types_particuliers_internes(bool absolue) const
 { // on commence par récupérer la liste général provenant d'ElemMeca
   List_io <TypeQuelconque> ret = ElemMeca::Les_types_particuliers_internes(absolue);
   // ensuite on va ajouter les données particulières
   Grandeur_scalaire_double grand_courant; // def d'une grandeur courante
   //  $$$ cas de l'épaisseur moyenne initiale
   TypeQuelconque typQ1(EPAISSEUR_MOY_INITIALE,SIG11,grand_courant); ret.push_back(typQ1); ….
```

-**au niveau d'ElemMeca**

```
List_io <TypeQuelconque> ElemMeca::Les_types_particuliers_internes(bool absolue) const
 { …. def→ListeGrandeurs_particulieres(absolue,ret); // cas des grandeurs liées à la déformation
   //    b) cas des grandeurs liées à la loi de comportement mécanique
   loiComp→ListeGrandeurs_particulieres(absolue,ret); //    b)  grandeurs liées à la loi de comportement mécanique
   … //    d) cas des infos stockés aux points d'intégration, donc par l'élément : ex : module de compressibilité total
    TypeQuelconque typQ1(MODULE_COMPRESSIBILITE_TOTAL,SIG11,grand_courant);ret.push_back(typQ1); etc.
```

Application : sortie post-traitement du potentiel hyperélastique relativement au volume final

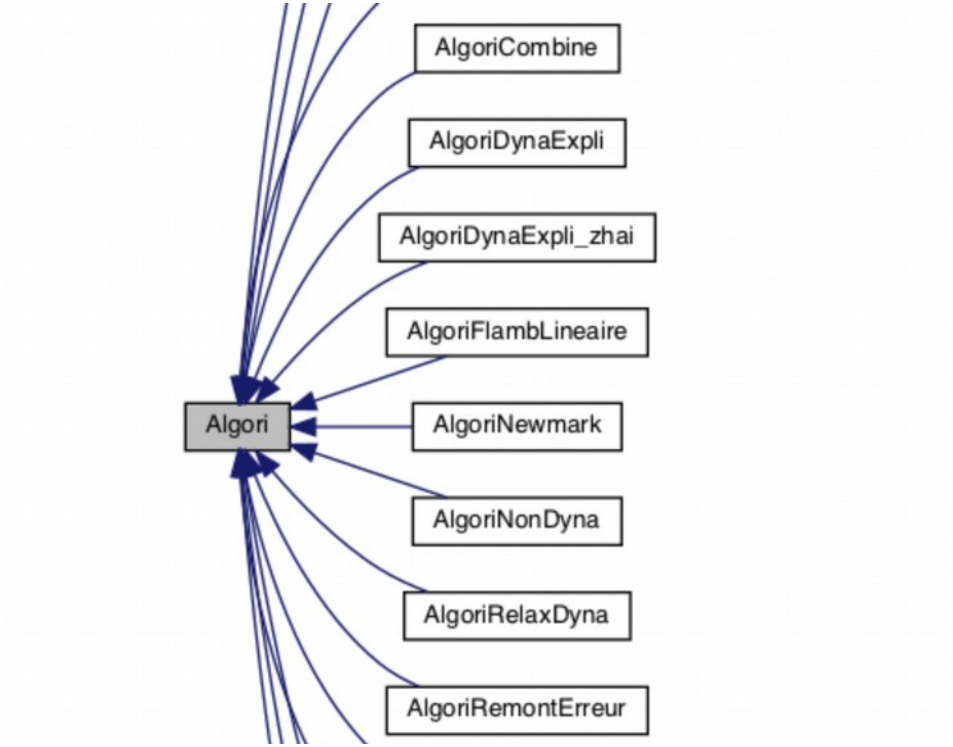
Par défaut, pour les lois hyperélastiques utilisant les invariants de type Mooney-Rivlin, on peut consulter en post-traitement le potentiel «W » qui est relatif au volume non déformé (par construction)
L'objectif proposé pour l'application est d'introduire la possibilité de fournir également le potentiel relatif au volume matière finale omega :

$$W = \omega \frac{\sqrt{g}}{\sqrt{g_0}} = V \omega$$

NB : Si on veut introduire une nouvelle grandeur disponible en sortie cela se passe au niveau des classes qui génèrent les grandeurs , par ex. pour les lois au travers de 2 méthodes :
- ListeGrandeurs_particulieres → donne la liste des grandeurs
- Grandeur_particuliere → assigne les valeurs aux grandeurs particulières

Les algorithmes globaux

Algori : classe d'interface virtuelle
Ex : relaxation dynamique



Idée du fonctionnement linéaire d'un calcul

```
Herezh    // main
 … Projet projet(retour,argc,argv); // def de l'objet de la classe de gestion de tous les projets en cours
    projet.Lecture();              // 1- lecture initiale
    projet.Calcul();
      algori->Execution // (Par exemple AlgoriNonDyna::Execution)
        AlgoriNonDyna::InitAlgorithme
        AlgoriNonDyna::CalEquilibre
            Boucle sur les  incréments de charge
              Boucle sur les itérations
                  Algori::RaidSmEner //Calcul de la raideur et second membre
                      Boucle sur les éléments
                  Algori::RaidSmEnerContact // conditions de contact
                  …lesCondLim→ImposeConLimtdt  // exe de conditions limites
                  Convergence( //  test de convergence
                  sol = &(tab_mato(niveau_substitution)->Resol_systID // résolution
              Ecriture_base_info( // écriture éventuelle .BI
              VisuAuFilDuCalcul        // visualisation éventuelle au fil du calcul
        AlgoriNonDyna::FinCalcul
     projet.FermetureFichiersVisualisation();
```

→ **classes particulières de gestion** : Projet, Algori,

….

Les grandes classes : conteneurs et macro-méthodes

Objets créés par le Projet

```
ParaGlob *      paraGlob;       // parametres globaux
UtilLecture *   entreePrinc;    // acces a la lecture du fichier principal
LesMaillages *  lesMaillages;   // description des maillages
Algori*         algori;         // algorithme de calcul
LesReferences*  lesRef;         // references des maillages
LesCourbes1D*   lesCourbes1D;   // courbes 1D
LesFonctions_nD* lesFonctionsnD; // les fonctions multi-dimensionnelles
LesLoisDeComp*  lesLoisDeComp;  // lois de comportement
DiversStockage* diversStockage; // stockage divers
Charge*         charge;         // chargement
LesCondLim*     lesCondLim;     // conditions limites
LesContacts*    lescontacts;    // le contact eventuel
Resultats*      resultats;      // sortie des resultats
VariablesExporter* varExpor;    // variables exportées globalement

Tableau <Algori* > tabAlgo;  // pour le schema XML
Temps_CPU_HZpp   tempsMiseEnDonnees;   // TempsCpu
```

Conteneur : Les maillages → LesMaillages

- **le plus important**

```
Tableau<Maillage *> tabMaillage ; // tableau de maillages
```

- et des stockages spécifiques

```
  // ---- stockage des intégrales de volumes sur des références d'éléments ---- : exemple :
   Tableau <TypeQuelconque>  integ_vol_typeQuel,  integ_vol_typeQuel_t;
  Tableau <const Reference*>   ref_integ_vol; // les références associées
  // si la référence est nulle, cela signifie que l'intégrale est figée: sa valeur ne change pas

  // idem pour  intégration de volume et en temps

  // ---- stockage des statistiques sur des références de noeuds ----
   Tableau <TypeQuelconque>  statistique_typeQuel,  statistique_typeQuel_t;
  Tableau <const Reference*>   ref_statistique; // les références associées
   //pour_statistique_de_ddl a la même dimension que ref_statistique
   Tableau < Ddl_enum_etendu > pour_statistique_de_ddl;
  // 2)  statistique avec cumul en temps: donc on commule le delta
  Tableau <TypeQuelconque>   statistique_t_typeQuel,   statistique_t_typeQuel_t;
  Tableau <const Reference*>  ref_statistique_t; // les références associées
```

Etc…..

Conteneur Maillage :

-le plus important :

```
Tableau<Noeud  *> tab_noeud; // tableau des noeuds du maillage
Tableau<Element *> tab_element; // tableau des elements du maillage
```

- puis beaucoup de choses….exemples :

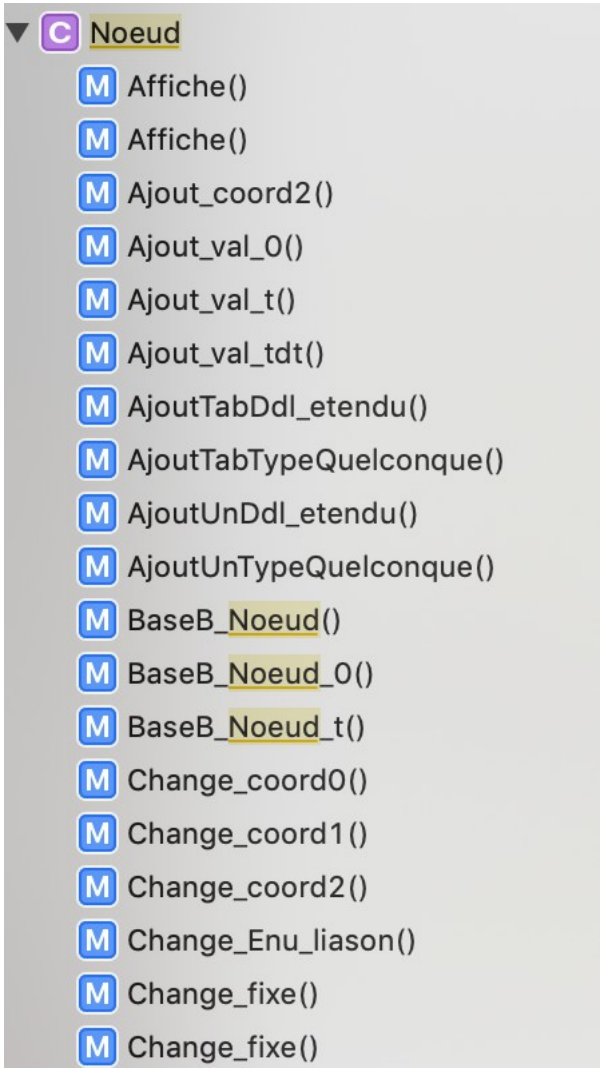
```
int idmail ; // numero de maillage
string nomDuMaillage;
LaLIST <Front> listFrontiere; // list des elements frontieres du maillage, c-a-d des frontières uniques

Tableau <Noeud *> tab_noeud_front;// tableau des noeuds des éléments frontières
Tableau < Tableau <Front> > mitoyen_de_chaque_element;

// en 3D liste de toutes les frontières lignes (non traités par CreeElemFront())
LaLIST <Front> listFrontiere_ligne_3D;

// définie la liste des types de degrés de liberté inconnus, qui vont être calculés par la résolution des problèmes
Tableau <Enum_ddl >  ddl_representatifs_des_physiques;
// idem au niveau des types de problèmes gérés par les éléments
Tableau <EnumElemTypeProblem > types_de_problemes;
```

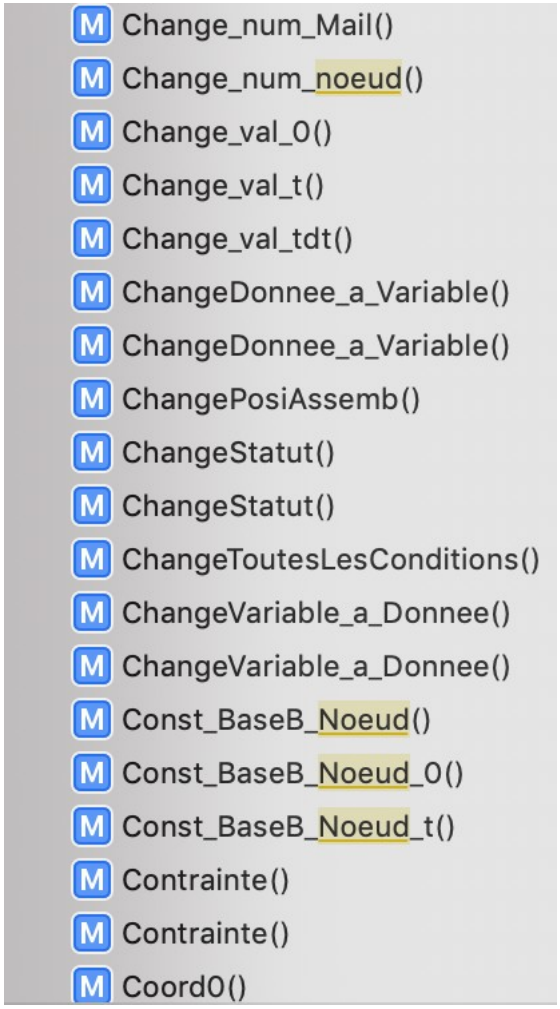
classe : Noeud

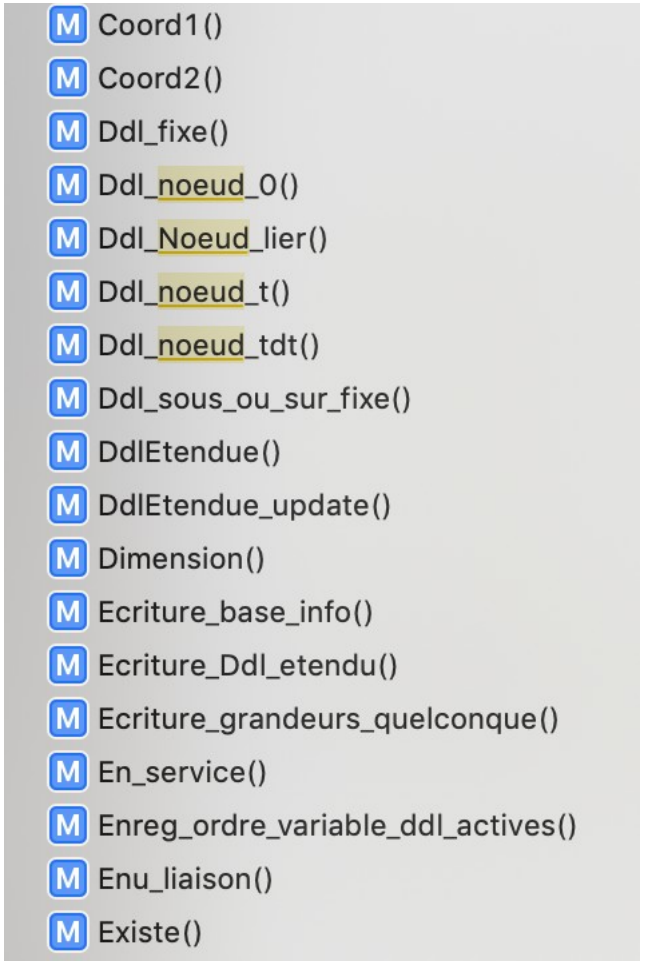
Noeud :
Conteneur : Globalement : stocke
- les coordonnées à t= 0, t et tdt
- les degrés de libertés : inconnues des problèmes
→ des « cas d'assemblage » chaque cas est associé à
. un problème particulier
. un jeux de degré de liberté particulier
- éventuellement des ddl étendu, par exemple comme donnée
- éventuellement des grandeurs quelconques : dans ce cas Noeud sert uniquement de conteneur avec des méthodes d'I/O (création, mise à jour etc.)
Méthodes : principalement :
- gestion : création/accès/modification des données (en particulier pour l'encapsulage)

Particularité :
Les données se construisent au fil du calcul

Exemple de méthodes associées à Noeud

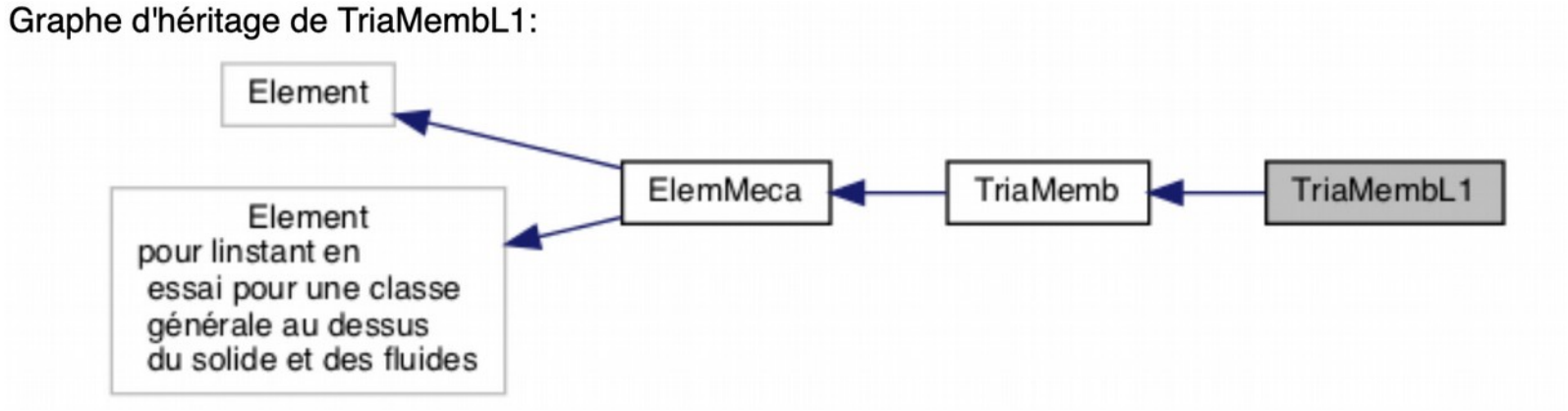






Classe Element

Exemple d'un triangle linéaire :



Element : classe virtuelle d'interface pour tout type d'élément
ElemenMeca : classe virtuelle pour les éléments mécanique
TriaMemb : classe virtuelle pour les éléments 2D triangulaire
TriaMembL1 : triangle à interpolation linéaire (nb pti : par défaut)

NB : 1) cf. doxygen_herezh pour l'ensemble des éléments
2) la génération initiale de la liste de chaque type d'élément utilise un mécanisme de constructeur static, qui agit avant le démarrage du programme : c'est spécifique aux éléments, les autres objets génériques sont créés via une classe particulière du type : lesFonctions_nD, Les_references, Les ….

Application : étude de PoutSimple1

l'idée est au travers d'un élément 2D de flexion, class PoutSimple1 : public PiPoCo actuellement non opérationnel

. de compléter les méthodes pour rendre fonctionne l'élément

→ compréhension et exemple du fonctionnement des classes :

- élément géométrique 1D : GeomSeg
- métrique générale : Met_abstraite
- puis métrique générale des poutres et plaques class Met_PiPoCo : public Met_abstraite
- puis métrique spécifique : class Met_Pout2D : public Met_PiPoCo
- la classe générique mére : class PiPoCo : public ElemMeca
- puis la classe mère : class ElemMeca : public Element

→ les classes :

LesPtIntegMecaInterne : les pti de l'élément
LesChargeExtSurElement : les chargements extérieurs calculés
Loi_comp_abstraite : la loi de comportement mécanique
CompThermoPhysiqueAbstraite : la loi thermophysique
CompFrotAbstraite : la loi de frottement
Deformation : la déformation
EnergieMeca : les énergies calculées

- puis la classe de base : Element avec les classes :

Signature
ElFrontiere
ConstrucElement et NouvelleTypeElement

Application : intégration d'un nouveau type d'élément : les éléments Pyramid

Méthodologie proposée :

- création d'un élément géométrique volumique en prenant modèle sur les pentaèdres :

→ création d'un élément commun : GeomPyraCom (cf. GeomPentaCom) qui hérite de ElemGeomC0

nécessite de récupérer : les fonctions d'interpolation, les points d'intégration

→ création d'un élément particulier : tout d'abord un élément à interpolation linéaire

→ GeomPyraL (cf. GeomPentaL)

Nécessite de rédiger en // une partie théorique/utilisateur qui indique les particularités de l'élément. Si possible faire de même pour tous les types d'interpolation/intégration

- création d'éléments finis : on se réfère aux pentaèdres

→ création d'un répertoire Pyramide sous Mecanique : /Herezh_pp/Elements/Mecanique/Pyramide

→ création d'un élément fini commun à toutes les pyramides, qui hérite d'ElemMeca PyraMemb à partir d'une copie de class PentaMemb : public ElemMeca

Faire les modifs ad hoc : il y a beaucoup de méthodes qui en fait utilisent une méthode déjà existante dans ElemMeca : l'objectif est principalement de paramétrer l'appel aux méthodes d'ElemMeca qui sont génériques.

→ création d'un élément fini particulier linéaire PyraL à partir d'une copie de class PentaL : public PentaMemb

Classe LesCondLim
gestion des CL sur les ddl

- les données principales

```
  Tableau <DdlLim> tabBloq; // tableau des ddl bloques
  Tableau <TorseurReac> tab_torseurReac; // tableau des torseurs de réaction
.. ..
  Tableau <DdlLim> tabInit; // tableau des ddl d'initialisation
  Tableau <ReactStoc> reaction; // les reactions pour les ddl bloqués
  Tableau <ReactStoc> reaction_CLin; // les reactions pour les conditions linéaires
  Tableau <ReactStoc> reactionApresCHrepere; // les reactions après CHrepere

  // tableau des conditions linéaires a imposer par les données d'entrée
  Tableau <I_O_Condilineaire > tab_iocondiline;
  Tableau < Tableau <Condilineaire > > tab_CLinApplique; // tableau des conditions linéaires
                                  //résultant de l'application de  tab_iocondiline
..  ..
   Tableau < CondLim > condlim; // les fonctions et données qui permettent d'imposer les cl
  // aux matrices, aux second membres, de remonter aux efforts apres
  // resolution etc ..., le tableau est indicé par le numéro de cas d'assemblage

  //------- temps cpu -----
  Temps_CPU_HZpp tempsCL; // temps cumulé pour imposer les CL imposées
  Temps_CPU_HZpp tempsCLL; //  temps cumulé pour imposer les CLL
```

Suite Classe LesCondLim
les méthodes principales

```
void Lecture1 // lecture des  conditions limites : ddl bloque
void Lecture2 // lecture des  conditions limites linéaires
void Lecture3 // lecture des  conditions limites : initialisation
 // introduction des données et variables pour leurs emplois futures, avec init par défaut
void IntroductionDonnees (..);

void Initial( // initialisation des ddl avec le tableau de ddl d'init
 // incrementation des coordonnees a t+dt et des ddl en fonctions des ddl imposes et de l'intensité de chargement
void MiseAJour_tdt (..)
// mise en place de la répercussion sur les noeuds des conditions linéaires imposées externes (par les données d'entrées),
void MiseAJour_condilineaire_tdt (..)
.. void ImposeConLimtdt  // mise en place des conditions limites sur les matrices et second membres
 // mise en place de condition externe lineaires expression de la raideur et du second membre dans un nouveau repere
bool CoLinCHrepere_ext(..);
bool Largeur_Bande // def de la largeur de bande  en fonction des conditions linéaire limite en entrée
//----- lecture écriture de restart -----
 void Lecture_base_info(..);
 void Ecriture_base_info(..);
```

Et .. divers affichages relatif aux cl

Suite Classe LesCondLim
exemple d'utilisation typique

```
AlgoriNonDyna::InitAlgorithme
   lesCondLim->InitNombreCasAssemblage // mise à jour du nombre de cas d'assemblage pour les conditions limites
  lesCondLim->Initial // init des ddl avec les conditions initials
……..
AlgoriNonDyna::CalEquilibre
      // -- initialisation des coordonnees et des ddl a tdt en fonctions des
       // ddl imposes et de l'increment du chargement: change_statut sera recalculé ensuite
       lesCondLim->MiseAJour_tdt
        //  mise en place des conditions linéaires (ne comprend pas les conditions linéaires de contact éventuelles)
       lesCondLim->MiseAJour_condilineaire_tdt
..
       lesCondLim->InitSauve(Ass.Nb_cas_assemb());// - initialisation des sauvegardes sur matrice et second membre
       lesCondLim→ReacAvantCHrepere // on récupère les réactions avant changement de repère et calcul des torseurs de
réaction
 ..

       // prise en compte des conditions imposée sur la matrice et second membres (donc dans le nouveau repère)
       lesCondLim->ImposeConLimtdt(lesMail,lesRef,(*matglob),vglobaal,Ass.Nb_cas_assemb()
                                   ,cas_combi_ddl,vglob_stat);
Etc.
```

Classe Charge 1) les principales données

```
// densite de force volumique dans le repère absolu
Tableau < BlocCharge< BlocDdlLim<BlocForces> > > tabFvol;
// densite de force surfacique dans le repère absolu
Tableau < BlocCharge< BlocDdlLim<BlocForces> > > tabFsurfac;
// pression uniformement reparti, appliquee normalement a la surface
Tableau < BlocCharge< BlocDdlLim<BlocIntensite> > > tabPresUnif;
// force ponctuelle sur un noeud
Tableau < BlocCharge< BlocDdlLim<BlocForces> > > tabPonctuel;
// pression unidirectionnelle type pression  des fluides
Tableau < BlocCharge< BlocDdlLim<BlocForces> > > PresUniDir;
// pression hydrostatique
Tableau < BlocCharge< BlocDdlLim<PHydro> > > PresHydro;
// pression hydrodynamique
Tableau < BlocCharge< BlocDdlLim<PHydrodyna> > > coefHydroDyna;
// densite de force lineique dans le repère absolu
Tableau < BlocCharge< BlocDdlLim<BlocForces> > > tabFlineique;
// densite de force lineique suiveuse, c'est-à-dire qui suit le repère
// locale (possible uniquement pour des éléments 2D)
Tableau < BlocCharge< BlocDdlLim<BlocForces> > > tabFlineiqueSuiv;
// torseur d'effort via une répartition de charges ponctuelles
Tableau < BlocCharge< BlocDdlLim<PTorseurPonct> > > tabTorseurPonct;
```

Avec les classes imbriquées :

```
class BlocCharge : public Bloc_particulier
  protected :
     string  co_charge; // nom d'une courbe de charge éventuelle
    string  f_charge; // nom d'une fonction nD utilisant des variables globales et autres
     double echelle;
    double t_min,t_max; // temps mini et maxi de durée des ddl imposés
    string nom_fnD_t_min; // nom éventuelle de la fonction associée
    Fonction_nD* fnD_t_min; // fonction nD associée, éventuelle
    string nom_fnD_t_max; // nom éventuelle de la fonction associée
    Fonction_nD* fnD_t_max; // fonction nD associée, éventuelle
    int precedent; // pour la description de l'évolution du ddlLim
    string attribut; // une chaine de caractère qui sert pour donner une
class BlocDdlLim : public Bloc_particulier
     Protected :  string*  nom_maillage; // nom d'un maillage
/// bloc spécialisé pour les forces
class BlocForces : public BlocScalVecTypeCharge
{public :// Constructeur
    BlocForces () : BlocScalVecTypeCharge(1,0) {}; // par defaut
class BlocScalVecTypeCharge : public BlocGeneEtVecMultType
 class BlocGeneEtVecMultType
      protected :
       Tableau <Coordonnee> vect_coorpt; // les coordonnées du vecteur
       Tableau <Tableau<string> >  ptnom_vect; //  les noms associés aux vecteurs
       Tableau<double>  tab_val ; //  les scalaires
       Tableau <string >  ptnom_tab_val; //  les noms associés aux scalaires
       string nomref; // nom de la ref
       string motClef; // nom du mot cle
```

Classe Charge 1) les principales méthodes

- Méthodes de : Lecture initiale, lecture/écriture sur .BI
- Méthodes Affichage, initialisation, incrémentation, etc.
- les 2 méthodes centrales :

bool ChargeSecondMembre_Ex_mecaSolid // application des efforts de chargement
bool ChargeSMembreRaideur_Im_mecaSolid // effort et opérateur tangent sur le chargement

Globalement :
- on parcourt tous les chargements enregistrés
  Pour chaque chargement actif
    On parcourt la référence associée au chargement
      On prépare les données nécessaires pour le calcul de la charge :
        - l'intensité de la charge en fonction de l'incrément en cours
        - paramètres spécifiques au type de chargement
        - récupération des éléments finis concernés
      On appelle chaque élément fini avec les paramètres ad hoc, qui :
        - calcul le chargement, et éventuellement l'opérateur tangent
        - retourne les vecteur/matrice locaux associés
      On assemble les vecteur/matrice locaux dans le vecteur et la matrice globale

Bilan :

Charge → gestionnaire des chargements
Element (en général ElemMeca) → calculs effectif du chargement

cf. fichier pour illustration:
Charge2.cc → Charge::ChargeSMembreRaideur_Im_mecaSolid

Charge : utilisation typique

```
AlgoriNonDyna::InitAlgorithme
      charge->Initialise(lesMail,lesRef,pa,*lesCourbes1D,*lesFonctionsnD);

AlgoriNonDyna::CalEquilibre
..
       //Boucle sur le chargement
       while (   ((!charge->Fin(icharge,!pas_de_convergence_pour_l_instant))
                        || pas_de_convergence_pour_l_instant ||(icharge == 1)
                      )
                       && (charge->Fin(icharge,true)!=2) // si on a dépassé le nombre d'incrément permis on s'arrête dans
tous les cas
                      && (charge->Fin(icharge,false)!=3) // idem si on a dépassé le nombre d'essai d'incrément permis
..-
      // mise en place du chargement impose, c-a-d calcul de la puissance externe
      // si pb on sort de la boucle
      if (!(charge->ChargeSMembreRaideur_Im_mecaSolid
                              (Ass,lesMail,lesRef,vglobex,*matglob,assembMat,pa,lesCourbes1D,lesFonctionsnD)))
         { Change_PhaseDeConvergence(-10);break;};
..-
```

Les paramètres globaux : ParaGlob // parametres globaux

Possède un pointeur static sur soi-même → accessible partout en lecture dans toutes les classes Herezh

```
static ParaGlob * param;  // pointeur sur le seul membre ouvert
```

Et plusieurs méthodes static d'utilité générale:

Ex d'utilisation : ParaGlob::Dimension() → donne la dimension en tout endroit

Objectif :

- fournir des paramètres généraux : ex : la dimension de l'espace en cours, le niveau d'impression ..
- fournir un pointeur sur ParaGlob (voir plus loin), et gestion interne du pointeur : ex : pour changement d'Algo en cours

```
const ParaAlgoControle & ParaAlgoControleActifs();
```

Données (exemple)

```
static EnumLangue langueHZ; // langue utilisée pour les entrées sorties
static int dimensionPb; // dimension du probleme = 1 unidimensionnel par defaut
static int nivImpression; // niveau d'impression
static string nbVersion; // numéro de version du logiciel
```

Etc.

Méthodes (exemples):

Encapsulage :

```
static const int Dimension () { return dimensionPb; } ;;// retourne la dimension du pb en lecture uniquement
```

Gestion du temps

```
static const VariablesTemps& Variables_de_temps() }; // récup des paramètres liés au temps
```

Gestion de grandeurs quelconques consultables partout ex : les constantes et variables utilisateurs

→ des méthodes pour : ajouter, consulter, modifier si authorisé etc. sous différentes formes

Classe : ParaAlgoControle

Globalement stockage et gestion des paramètres généraux :
→ correspond à tous les paramètres que l'utilisateur indique dans le .info au niveau des paramètres de contrôle et de pilotage (cf. utilisation_herezh++.pdf, douzième partie), mots clés :

controle
para_dedies_dynamique
para_energie
para_pilotage_equi_global
para_contact
para_affichage
para_calculs_geometriques

- sont accessibles partout en lecture via ParaGlob
- il peut y avoir plusieurs instances → via les algorithmes combinés → donc changement de paramètres en cours d'exécution

Méthodes :

- principalement les accès encapsulés
- I/O fichiers et interactif pour la construction de .info
- quelques méthodes spécifiques par exemple relatives au temps ex :

```
// indique si la sauvegarde au fil du calcul est autorisée,
bool SauvegardeFilCalculAutorisee(int incre,const double& temps_derniere_sauvegarde,bool dernier_calcul) const;
```

Annexe : Données Hyper_W_gene_3D :

```
protected :
    // données
    int sortie_post; // permet de stocker et ensuite d'accéder en post-traitement à certaines données
    int nb_para_loi; // nombre de paramètre de la loi:
    double I_B,I_BB;// 2 invariants primaires de B :  I_B, I_{B.B}
    double II_B,III_B; // les 3 invariants en I de B: I_1=I_B, I_2=II_B,I_3=III_B
    Vecteur J_r ; // les 3 invariants en J
    // variation des I II III par rapport aux composantes covariantes de la déformation
    Tenseur3HH d_I_B_epsBB_HH,d_II_B_epsBB_HH,d_III_B_epsBB_HH;
    // variation des J_r par rapport aux composantes covariantes de la déformation
    Tableau <Tenseur3HH> d_J_r_epsBB_HH;
    // variation seconde des J_r par rapport aux composantes covariantes de la déformation
    Tenseur3HHHH d_J_1_eps2BB_HHHH,d_J_2_eps2BB_HHHH,d_J_3_eps2BB_HHHH;
    // autres variables utiles pour la loi de comportement
     double V; // variation relative de volume
     Tenseur3HH BB_HH; // B . B
     Tenseur3HB B_HB; // B en mixte
     Tenseur3HB BB_HB; // B . B en mixte
     // variables tensorielles du second ordre intermédiaires
     Tenseur3HHHH IxI_HHHH,IxbarreI_HHHH,IxB_HHHH,BxI_HHHH;
```

Annexe :La classe générique pour les lois en mécanique :

Loi_comp_abstraite

Les données principales:

```
private: // non accessible par les classes dérivées → acces uniquement via des méthodes dédiées
   // liste des grandeurs locales qui peuvent être accédé localement via saveResul
  // cette liste nécessite d'être abondée par la méthode Activation_stockage_grandeurs_quelconques
    list <EnumTypeQuelconque > listQuelc_mis_en_acces_localement;
  // la liste totale qui est construite au moment de la définition de la loi
  // cette liste est différente de listQuelc_mis_en_acces_localement qui est celle des grandeurs
  // réellement demandés pour une mise en place : rempli par les classes dérivées
  list <EnumTypeQuelconque > listdeTouslesQuelc_dispo_localement;

  const PtIntegMecaInterne* ptintmeca_en_cours; // si différent de Null, peut-être utilisés par les méthodes internes
   int permet_affich_loi; // pour permettre un affichage spécifique dans les méthodes
   Fonction_nD * permet_affich_loi_nD; // fonction nD éventuelle pour l'affichage
  List_io <TypeQuelconque > li_quelconque; // stockage inter des grandeurs vraiment quelconques utilisables par permet_affich_loi_nD
  Tableau <const TypeQuelconque *  > tab_pt_li_quelconque;  // stockage de pointeurs de li_quelconque

protected : // accessible par les classes dérivées
  SaveResul * saveResul; // pointeur de travail utilise par les classes derivantes
  bool comp_tangent_simplifie;// indic pour définir si oui ou non on utilise un comportement tangent simplifié
  bool utilise_vitesse_deformation; // indication si l'on utilise ou pas la vitesse de déformation
  Enum_type_deformation type_de_deformation;// indication du type de déformation utilisée
  bool thermo_dependant; //  indique si oui ou non la loi dépend de la température
  double temperature_0,temperature_t,temperature_tdt; // variables valides que si l'on est thermo_dependant utilisée par les classes dérivées
  double* temperature; // pointeur sur la température de travail (à 0, à t ou tdt)
  bool dilatation; // variable interne, qui est mise en route par l'appel de certaine fonction, comme celles
  Deformation * def_en_cours; // si différent de NULL, indique une déformation qui peut être utilisée par les classes dérivées
  Temps_CPU_HZpp   temps_loi; // spécifique à ce type de loi: cumule tous les appels
```