Document de conception
TRIO_U Version 1

Juillet 1996

M. Farvacque, Ph. Emonot, O. Cueto

STR/LMTL/96-20

Table des matières

INTRODUCTION

1. Les concepts de l'approche objet

Ces concepts sont maintenant largement diffusés. On rappelle ici quelques définitions et on renvoie à la bibliographie pour une présentation complète de ces concepts.

Modélisation et conception orientées objet. J. RUMBAUGH (1995)
Éditions Masson. Prentice Hall

Les concepts

Un *objet* regroupe: des données,
des traitements,
des états, des comportements,
d'autres objets.

Une *classe* rassemble les objets ayant les mêmes caractéristiques et les mêmes comportements.

Un objet fournit des *services* à l'extérieur.

L'interface d'un objet (ou la classe) est l'ensemble des services proposés par l'objet (ou la classe).

La *mise en oeuvre* est la réalisation de l'interface.

Une *requête* est l'appel d'un service (par l'envoi d'un message).

Une *opération* représente la mise en oeuvre d'un service.

Surcharge d'opération: représente la mise en oeuvre d'un service par plusieurs opérations de signatures différentes.

L'*encapsulation* assure l'indépendance de l'interface vis-à-vis de la mise en oeuvre. La mise en oeuvre peut changer mais l'interface est conservée.
Pour supporter l'encapsulation, le paradigme objet prévoit le *masquage* de l'information : une partie de l'interface n'est disponible que pour certains clients.

On regroupe les classes de même caractéristique et de même comportement. Les classes sont ordonnées par deux mouvements contraires: la *généralisation* et la *spécialisation*.
L'*héritage* est le procédé par lequel une classe dite sous-classe reçoit une partie de sa définition d'une autre classe, dite super-classe. On dit aussi que la sous-classe *dérive* de sa super-classe.
Les caractéristiques de la classe ancêtre sont transmises à la classe dérivée.
Les mises en oeuvre d'opérations de la classe ancêtre sont applicables à la classe dérivée.
La classe ancêtre prévoit des comportements pour la classe dérivée.
La classe dérivée peut redéfinir un comportement hérité de la classe ancêtre: on dit qu'il y a *surcharge*.

Un *attribut* est un caractère propre à la classe.

On définit des *relations* (ou associations) entre les classes.

L'*agrégation* est une forme d'association forte dans laquelle un objet agrégat est fait de composants. Deux objets distincts sont englobés, l'un des deux est *une partie* de l'autre.

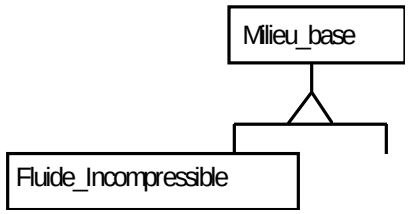
Polymorphisme fonctionnalité qui permet d'associer plusieurs implémentations de méthodes à un seul identificateur. Le polymorphisme est implémenté par liaison dynamique. Il confère autant de puissance que de souplesse au code qui en fait usage.

2. Modèle objet OMT

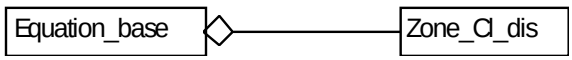
Les concepts objets sont utilisés dans TRIO_U. Le modèle objet OMT permet la représentation de la structure statique du système.
On définit rapidement ci-dessous le formalisme OMT utilisé dans ce document.

1.1 Représentation OMT utilisée dans le modèle objet:

- l'**héritage**:
ex: `Fluide_Incompressible` dérive (hérite) de `Milieu_base`
`Fluide_Incompressible` est une spécialisation de `Milieu_base`

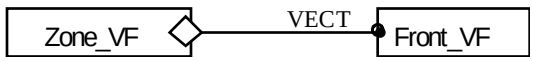


- les **agrégations simples** :
ex: `Equation_base` contient un objet `Zone_Cl_dis`

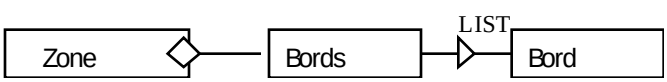


- les **agrégations multiples** : elles sont mises en oeuvre via un vecteur, ou une liste.

directement:
ex: `Zone_VF` contient plusieurs objets `Front_VF`.
`Zone_VF` porte un `VECT` de `Front_VF`:



ou par héritage:
ex: `Zone` contient l'objet `Bords`.
`Bords` hérite de la classe `LIST(Bord)`

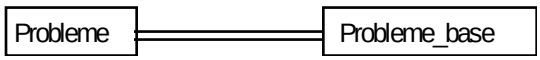


- les **associations** : En phase de conception on peut indiquer la direction du lien.

ex: `Equation_base` a une référence à un objet `Zone_dis`

- les opérations **virtuelles pures** notées oper{} qui rendent abstraite la classe qui les porte.

1.2 Les objets dits "**génériques**" dans TRIO_U: ces objets permettent la mise en oeuvre du polymorphisme et l'écriture d'opérations génériques.
Dans ce document cette relation (qui n'est pas formalisée par OMT) est représentée par un double trait.



Par exemple, il faut lire la relation entre les classes `Probleme` et `Probleme_base` de la façon suivante: "la classe `Probleme` représente une classe dérivée quelconque de la classe `Probleme_base`" : soit la classe `Pb_Hydraulique`, ou `Pb_Concentration`, ou `Pb_Conduction`, ou `Pb_Thermohydraulique` ou `Pb_Hyd_Therm_Concen`.

Ce lien est mis en oeuvre dans Trio_U par l'intermédiaire de la classe **`Deriv`** soit :

```
class Probleme : public DERIV(Probleme_base)
```

1.3 Les liens d'associations sont mis en oeuvre dans Trio_U par l'intermédiaire de la classe **`Ref`**.
exemple:

```
class Convection_Diffusion_std
{ ---
protected:
REF(Champ_Inc)  la_vitesse_transportante;
}
```

3. Structure de TRIO_U Version 1

TRIO_U est structuré en modules. Ce document décrit chaque module.

Noyau	Ensemble d'objets que peuvent se partager plusieurs applications.
ThHyd	Objets pour la thermo-hydraulique
ThSol	Objets pour le calcul de la température dans un solide
Operateurs	Opérateurs pour la thermo-hydraulique ou la thermique
Geometrie	Description du maillage
Champs_dis	Discrétisation des champs
Schemas_temps	Discrétisation en temps: Schémas en temps
Statistiques_temps	Traitement statistique des champs
VF	Objets communs aux méthodes de discrétisation "volumes finis"
VDF	Objets pour une discrétisation spatiale de type "volumes différences finies"
VEF	Objets pour une discrétisation spatiale de type "volumes éléments finis" (à écrire)
Utilitaires	Gestion des objets et des communications dans TRIO_U
Math	Module mathématique.

Ces deux derniers modules seront décrits dans un autre document.

4. Notations du document

Dans le texte, les noms des classes et de leurs membres sont écrits avec la police Courier : exemple, la classe `Probleme`.
Une classe représente une entité. Dans le texte, on assimile souvent le nom de la classe et l'entité qu'elle représente. On écrit par exemple : "un problème porte des équations" au lieu de "un objet de type `Probleme` porte des objets de type `Equation`", ce qui ne prête pas à confusion.

Ce document a été saisi avec WORD5; les diagrammes saisis avec Mac Draw Pro, sont insérés dans le texte.

5. Si vous voulez des précisions

Contactez l'équipe TRIO_U!

E-Mail : triou@alpes.cea.fr

TRIO_U : LE NOYAU

Le NOYAU de TRIO_U est un module qui contient un ensemble d'objets que peuvent se partager les différentes applications de la thermo-hydraulique : fluide incompressible ou compressible, monophasique ou diphasique, mono ou poly-constituants. D'autres types d'applications peuvent être aussi déduites de ce noyau : calculs thermiques, mécaniques, ...

On trouve surtout dans le noyau des classes de base et des classes génériques obtenues à partir des classes de base. Les classes instanciables appartiennent à des modules spécialisés.

Le Noyau contient les familles de classe suivantes:

1	Problème
2	Equation
3	Opérateur
4	Source
5	Solveur_Masse
6	Milieu
7	Discrétisation en temps
8	Discrétisation en espace
9	Conditions limites
10	Champ
11	Postraitement

1-NOYAU : LES PROBLEMES

Généralités: Le rôle d'un problème est la résolution sur un domaine des équations qui le composent. Un schéma de discrétisation en temps ainsi qu'un schéma de discrétisation en espace lui sont associés.

Classe générique: `Probleme`

Classe de base: `Probleme_base`

Méthodes virtuelles pures à implémenter par les classes dérivées de `Probleme_base`:

`nombre_d_equations() :` renvoie le nombre d'équations portées par une classe dérivée de `Probleme_base`

`equation(int i) :` renvoie l'équation de rang i. L'ordre des équations est fixé par la classe dérivée

Méthodes permettant de définir des algorithmes généraux, pouvant être éventuellement redéfinies par un problème particulier:
Les opérations suivantes sont déléguées à chaque équation composant le problème.

`calculer_pas_de_temps():` calcul de la valeur du prochain pas de temps

`sauvegarder(Sortie&):` sauvegarde du problème sur fichier.

`reprendre(Entree&):` lecture d'un fichier pour une reprise.

Méthodes d'acquisition et d'instanciation:

`completer():` remplissage des références, opération déléguée aux équations

`discretiser(Discretisation_base&):` discrétisation du domaine et de chaque équation

`associer():` envoie sa référence de `Probleme_base` aux équations

`associer(const Schema_Temps_base&):` remplit la référence `le_schema_en_temps` et la transmet aux équations

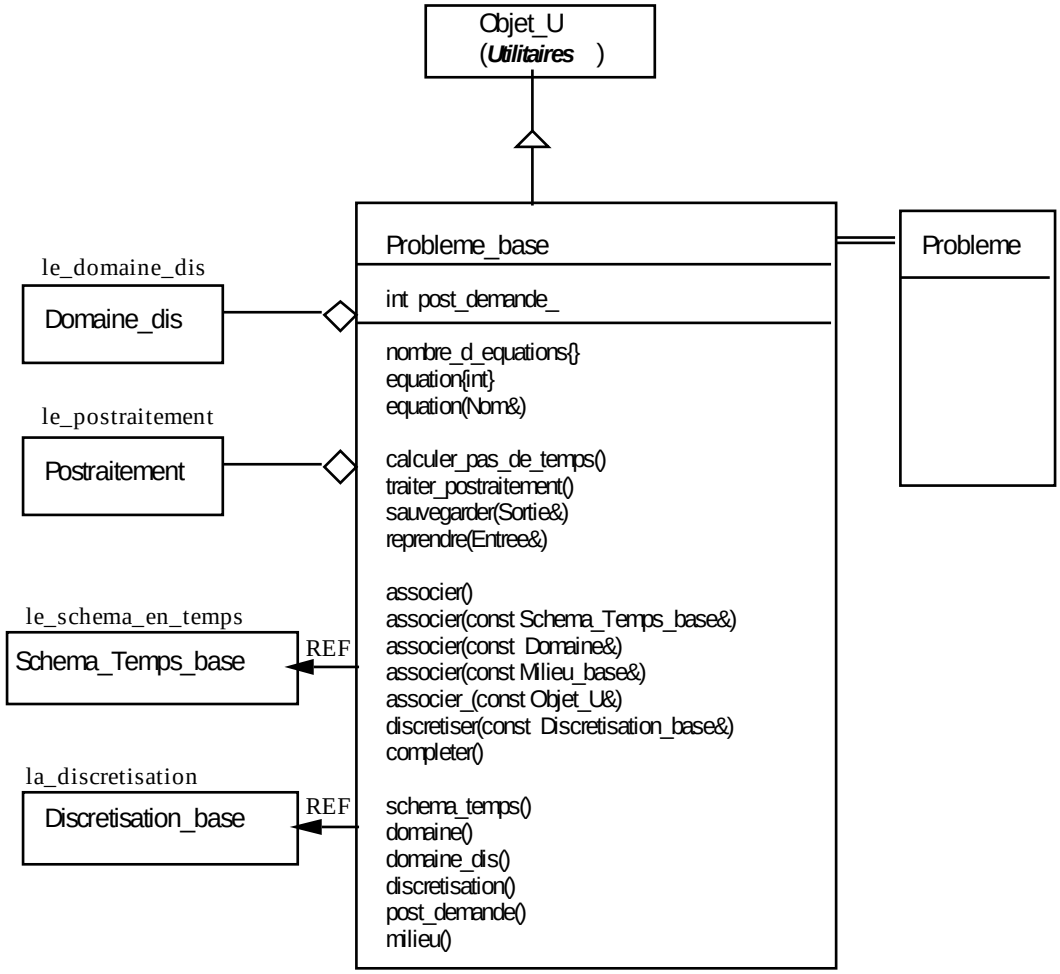
`associer(const Milieu_base&):` transmet la référence `Milieu_base` aux équations

`associer(const Domaine&):` transmet la référence `Domaine` au `Domaine_dis`

Autres méthodes:

`traiter_postraitement():` exécution du postraitement

Méthodesd'accèsauxmembresprivés: nondécritesdansce document



Noyau : Probleme

2-NOYAU : LES EQUATIONS

Généralités: Le rôle d'une équation est le calcul d'un ou plusieurs champs compte tenu des choix suivants:
- un schéma en temps,
- un schéma de discrétisation spatiale,
- des conditions aux limites,
- des termes sources et des opérateurs.

Une équation est portée par un Problème et possède une référence qui permet de remonter au problème qui la porte:

```
(Probleme_base le_probleme).
```

par la méthode:

```
Probleme_base&  probleme();
```

Classe générique: `Equation`

Classe de base: `Equation_base`

Méthodes virtuelles pures à implémenter par les classes dérivées de `Equation_base`:

`nombre_d_operateurs() :`	renvoie le nombre d'opérateurs portés par une classe dérivée de `Equation_base`
`operateur(int i) :`	renvoie l'opérateur de rang i. L'ordre des opérateurs est fixé par la classe dérivée
`inconnue()`	renvoie le `Champ_Inc` calculé par l'équation.
`milieu()`	renvoie l'objet `Milieu` associé à l'équation.
`associer(const Milieu_base&):`	associe un objet `Milieu` à l'équation

Méthodes permettant de définir des algorithmes généraux, pouvant être éventuellement redéfinies par une équation particulière:

`calculer_pas_de_temps():`	calcul du prochain pas de temps (opération déléguée aux opérateurs)
`sauvegarder(Sortie&):`	Sauvegarde du problème sur fichier. Opération déléguée au champ `inconnue()`.
`reprendre(Entree&):`	Lecture d'un fichier pour une reprise. Opération déléguée au champ `inconnue()`.
`derivee_en_temps_inco(...) :`	méthode générale pour le calcul de la dérivée du champ inconnue par rapport au temps. L'implémentation de la méthode est décrite ci-dessous:

```
DoubleTable& derivee_en_temps_inco(DoubleTable& derivee )
{
```

```
          derivee = 0;
          for (int i=0, i<nombre_d_operateurs(); i++)
                operateur(i).ajouter(derivee);
          les_sources.ajouter( derivee);
          return solveur_masse.appliquer(derivee);
    }
```

`mettre_a_jour() :` mise à jour de l'équation. L'opération est déléguée à tous ses composants (conditions aux limites , inconnue , opérateurs, solveur_masse et sources).

`preparer_calcul() :`

Méthodes d'acquisition et d'instanciation :

`discretiser()` remplissage de la référence `la_zone_dis` , typage du `solveur_masse`

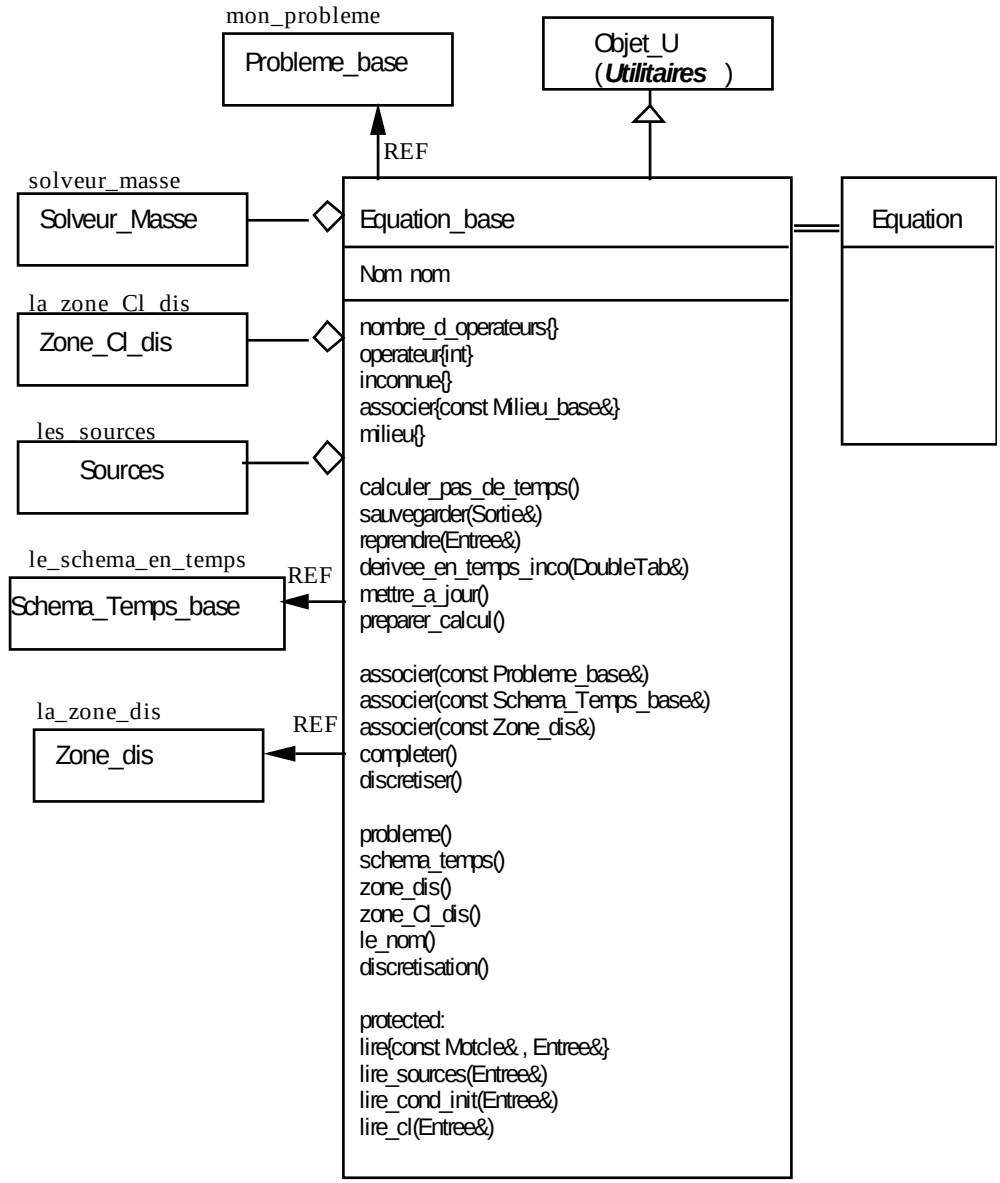
`associer(Probleme_base&)` remplissage de la référence `mon_probleme`, et envoi de sa référence d'`Equation_base` aux Sources et aux Operateurs.

`associer(cons Schema_Temps_base&):` remplit la référence `le_schema_en_temps`

`completer():` Opération déléguée aux Sources et aux Operateurs.

Autres méthodes :

`Méthodes d'accès aux membres privés :` non décrites ici



Noyau : Equation

3-NOYAU : LES OPERATEURS

Généralités: Les opérateurs sont des parties d'une équation. Ils héritent à la fois de la classe `Objet_U` et de la classe `MorEqn` (pour "morceau d'équation") qui est hors de la hiérarchie de TRIO_U.
La classe `MorEqn` possède une référence qui permet de remonter à l'équation qui la porte : (`Equation_base mon_equation`) et les méthodes:

```
const Equation_base& equation() const;
Equation_base& equation() ;
```

Classe générique: `Operateur`
La classe `Operateur` est une classe générique particulière car elle n'hérite pas de `DERIV(Operateur_base)`. Elle hérite uniquement de la classe `MorEqn`

Classe de base: `Operateur_base` qui hérite à la fois de la classe `Objet_U` et de la classe `MorEqn`.

La classe **`Operateur`** porte les opérations virtuelles pures suivantes:

`ajouter(...)`	ajout de la contribution de l'opérateur au second membre de l'équation.
`calculer(...)`	calcul de la contribution de l'opérateur.
`typer()`	instanciation de l'opérateur du type nécessaire au calcul. Le type est calculé, il dépend notamment des données de l'utilisateur, du choix de la discrétisation,...
`l_op_base()`	renvoi de l'`Operateur_base` correspondant.

Autres opérations générales:

`inconnue()`	renvoie le `Champ_Inc` de l'équation qui contient l'opérateur
`discretisation():`	renvoie la discrétisation supportée par l'équation

La classe **`Operateur_base`** porte les opérations virtuelles pures suivantes:

`ajouter(...)` : ajout de la contribution de l'opérateur au second membre de l'équation.
`calculer(...)` : calcul de la contribution de l'opérateur.
`associer(const Zone_dis&, const Zone_Cl_dis&, const Champ_Inc&)` : association des références (protected)

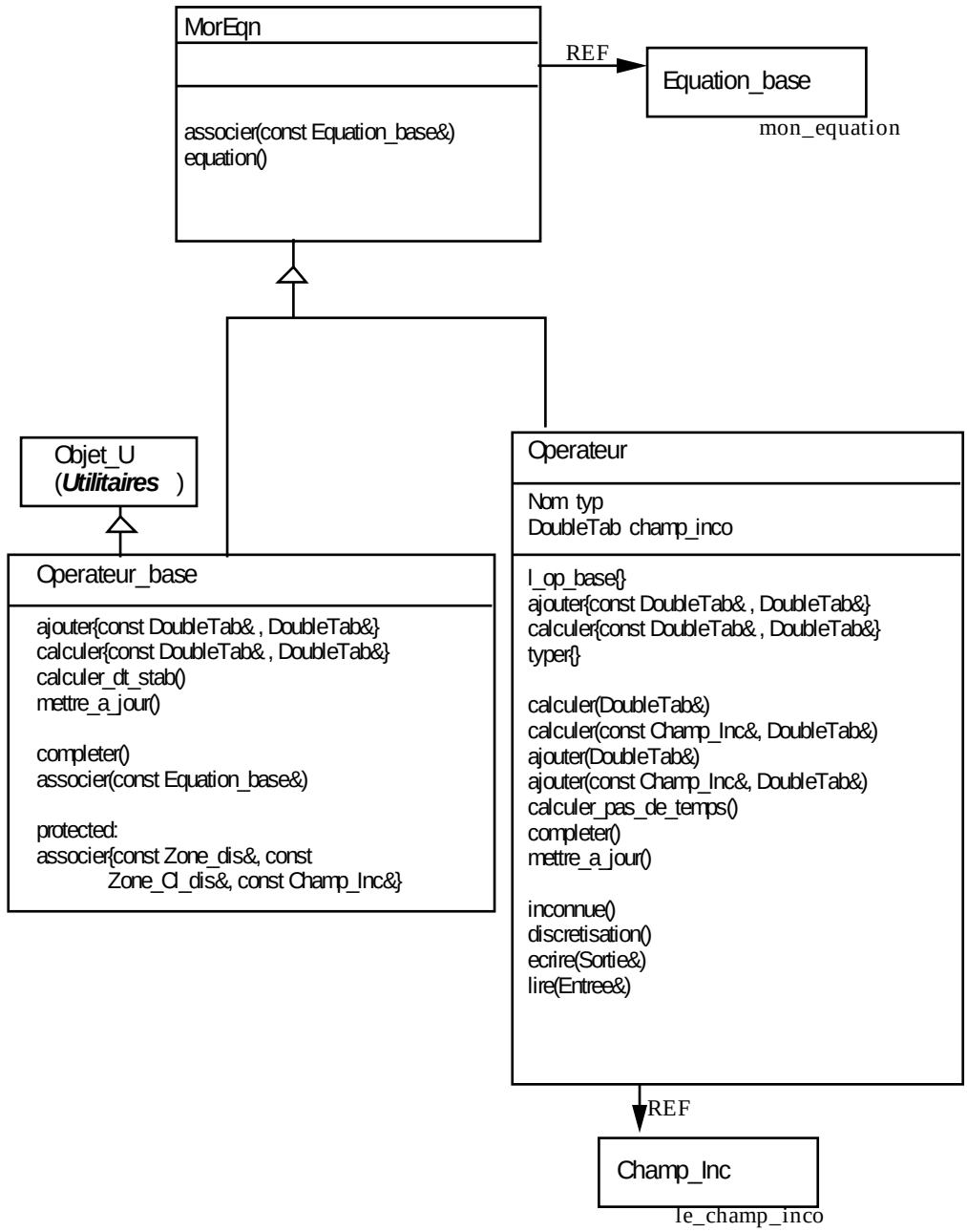
Autre opération:

`mettre_a_jour()`	mise à jour de l'opérateur.

Méthodes d'acquisition et d'instanciation:

`completer():`	appel de l'opération `associer(const Zone_dis&, ...)`

Remarque: Si un opérateur est implicité, il participe à la construction de la matrice du système. Leur contribution se fait alors par une méthode `assembler()`, non écrite dans TRIO_U Version 1.



Noyau : Operateur_base , Operateur , MorEqn

4-NOYAU : LES SOURCES

Généralités: Les sources sont des termes apparaissant au second membre d'une équation.

La plupart des termes sources sont définis dans le jeu de données par l'utilisateur, et quelques-uns sont générés automatiquement par les constructeurs des équations.

L'équation porte l'objet de type `Sources` , classe qui hérite de `LIST(Source)` et de la classe `MorEqn`.

Classe générique: `Source`

Classe de base: `Source_base`

La classe **`Source`** porte les opérations suivantes:

`typer(...) :` instanciation de la source du type nécessaire au calcul. Le type est calculé, il dépend notamment des données de l'utilisateur, du choix de la discrétisation,...

et les opérations d'accès aux opérations de la classe `Source_base.`

La classe **`Source_base`** porte les opérations virtuelles pures suivantes:

`ajouter(...) :` ajout de la contribution de la source au second membre de l'équation.

`calculer(...) :` calcul de la contribution de la source.

`associer_zones(const Zone_dis&, const Zone_Cl_dis&) :` association des références (méthode protégée)

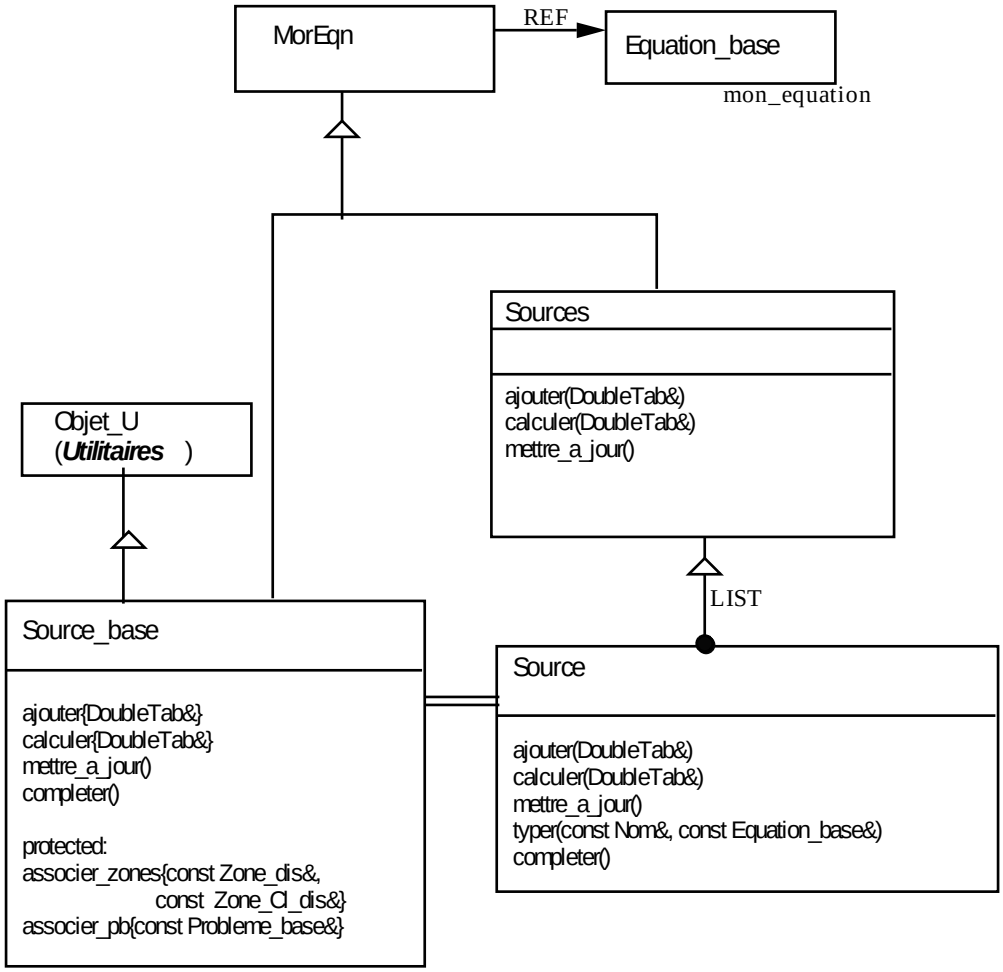
`associer_pb(const Probleme_base&):` association des références (méthode protégée)

Autre opération:

`mettre_a_jour() :` mise à jour de la source.

Méthodes d'acquisition et d'instanciation:

`completer():` appel des opérations `associer_zones( ...)` et `associer_pb(...)`



Noyau : Sources , Source , Source_base

5-NOYAU : SOLVEUR MASSE

Généralités: L'intégration des équations de bilan à l'aide du théorème de Gauss sur des volumes de contrôle conduit à un système linéaire dans lequel intervient une matrice de masse.

Masse* X = second_membre

Le type du champ X dépend de l'équation et de la discrétisation.
Suivant le type d'équation et de problème, la masse représentera vraiment la masse (ρ*V) ou se limitera au volume.

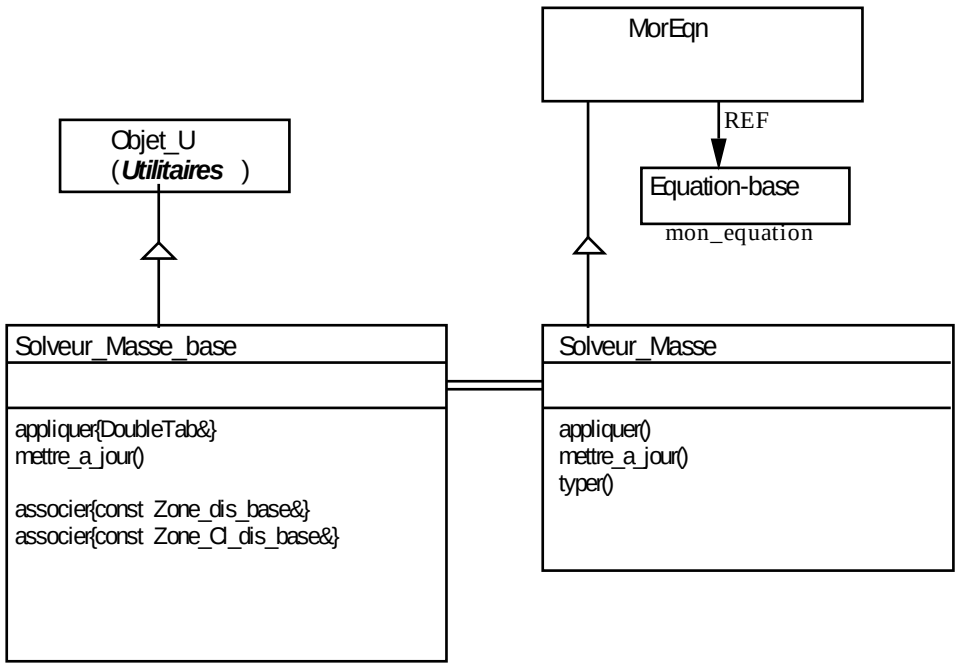
Le Solveur_Masse est porté par une équation.

Classe générique: `Solveur_Masse`
Cette classe hérite aussi de la classe `MorEqn`.
Possède la méthode `typer()` pour instancier la classe du type nécessaire au calcul.

Classe de base: `Solveur_Masse_base`

La classe **`Solveur_Masse_base`** porte les opérations virtuelles pures suivantes:

```
appliquer(DoubleTab& Y)            renvoie  le DoubleTab   (Masse-1)*Y
associer(const Zone_dis_base&) :   méthode d'acquisition
associer(const Zone_Cl_dis_base&)  idem
```



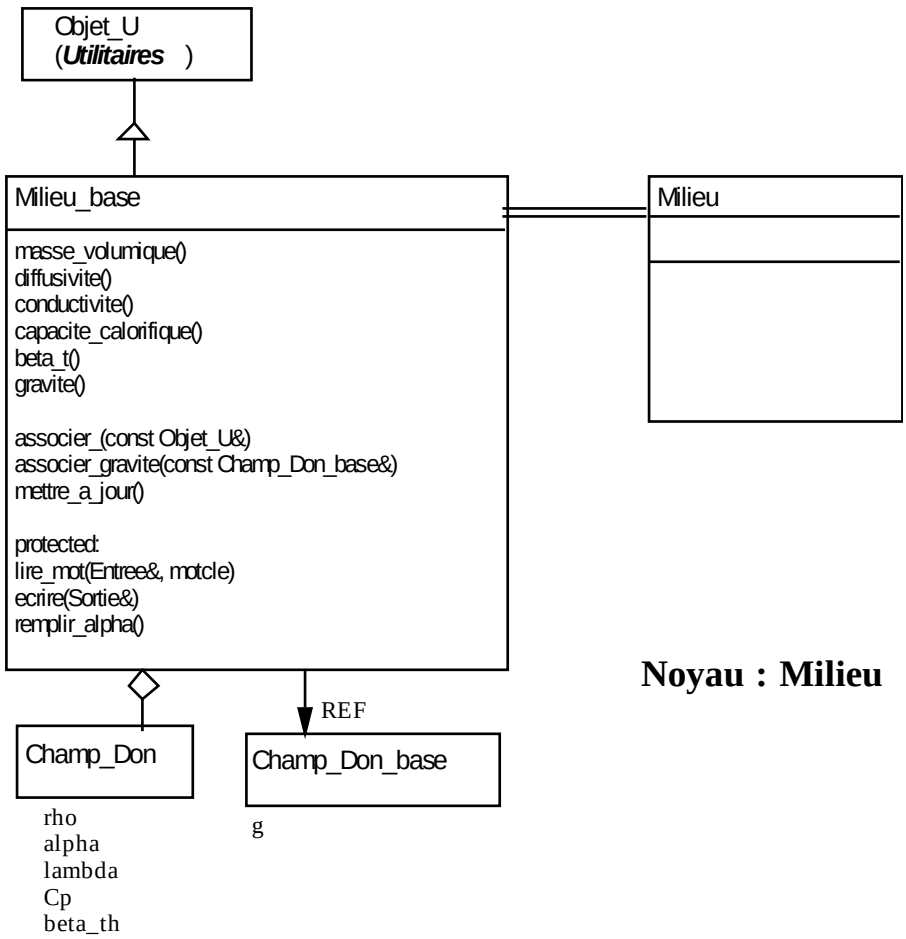
Noyau : Solveur_Masse

6-NOYAU : MILIEU

Généralités: Description du milieu fluide ou solide qu'on modélise
Le `Milieu` est associé au problème et aux équations.

Classe générique: `Milieu`

Classe de base: `Milieu_base`
Cette classe contient les "principaux" `Champ_Don` qui vont caractériser le milieu:
- la masse volumique,
- la diffusivité,
- la conductivité,
- la capacité calorifique,
- la variation de la masse volumique en fonction de la température (dilatabilité)



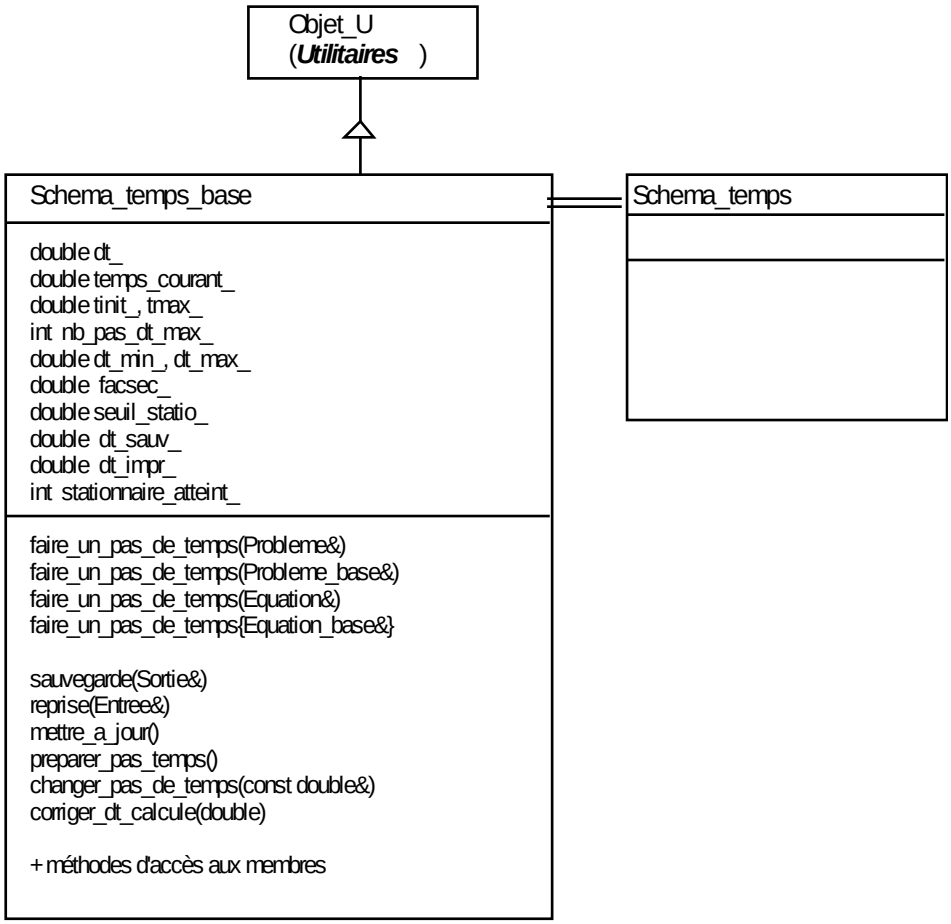
Noyau : Milieu

7-NOYAU : DISCRETISATION EN TEMPS

Généralités: Schémade discrétisationen tempschoisi par l'utilisateur.
Le `Schema_temps` estassociéau problèmeet aux équations.

Classegénérique: `Schema_temps`

Classede base: `Schema_temps_base`
Possèdela méthodevirtuellepure:
`faire_un_pas_de_temps(Equation_base&)`



Noyau : Schema_temps

8-NOYAU: DISCRETISATION EN ESPACE

8.1 La discrétisation

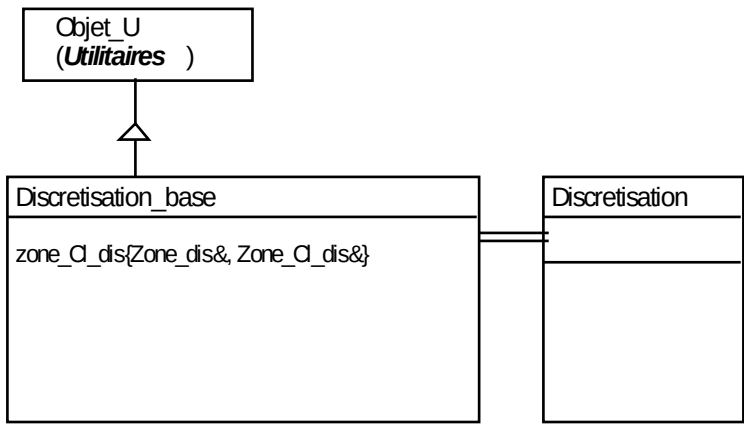
Généralités: Schémade discrétisationen espacechoisi par l'utilisateur.
La discrétisationest associéeau problème.

Classegénérique: `Discretisation`

Classede base: `Discretisation_base`

Les méthodes:

`zone_Cl_dis{---}:` méthode virtuelle pure à implémenter par les classes dérivées.
Discrétisationet typagede la `Zone_Cl_dis`



Noyau : Discretisation

8.2 Le Domaine discrétisé

Généralités: Le problème porte le domaine discrétisé `Domaine_dis`.

`Domaine_dis` La classe `Domaine_dis` représente le domaine discrétisé. Cette classe porte la classe `Zones_dis` qui contient elle même autant de `Zone_dis` qu'il y a de `Zone` dans le `Domaine`. (Rappel: dans TRIO_U Version 1 il n'y a qu'une Zone par domaine.)
Possède une référence au `Domaine`.

8.3 Discrétisation des zones :

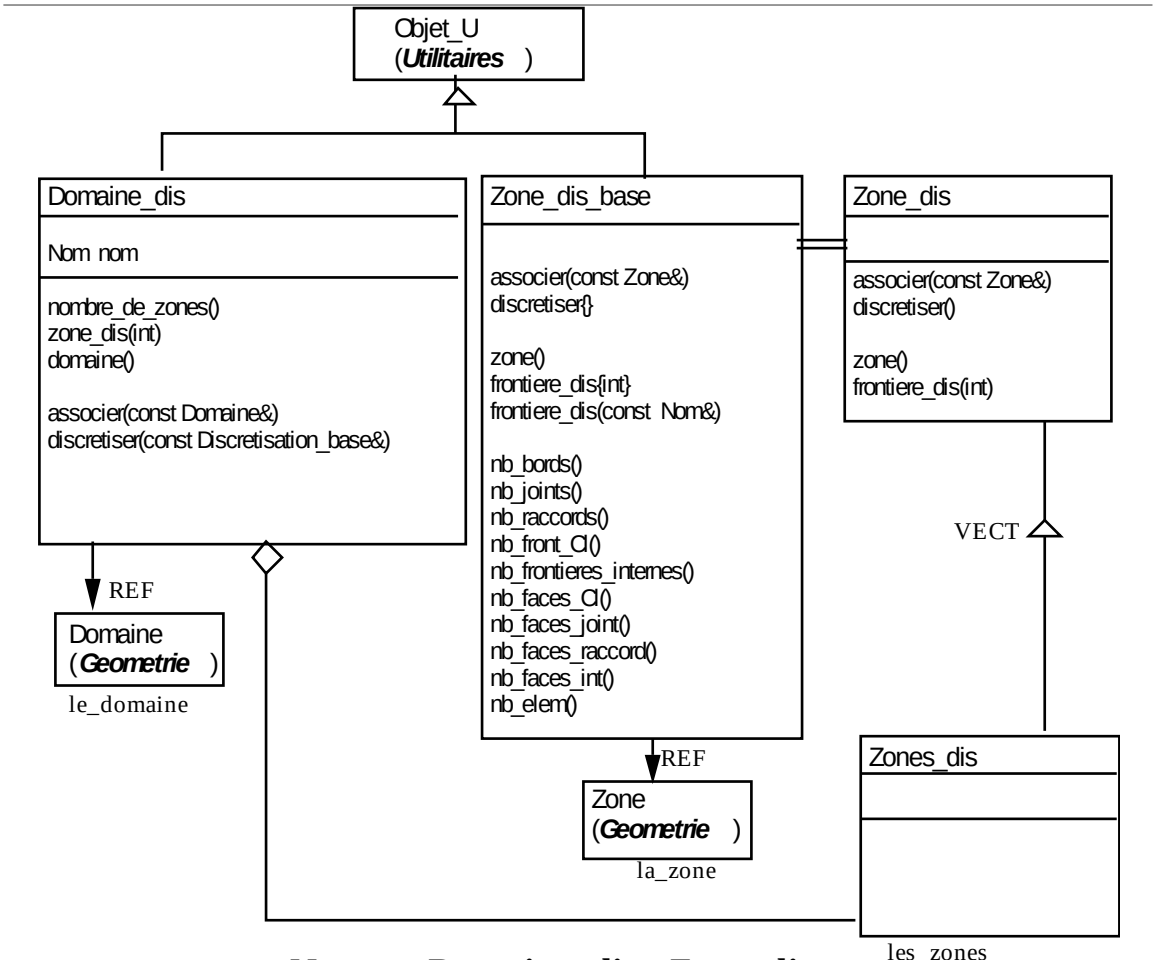
Classe générique: La classe `Zone_dis` représente une zone discrétisée.

Classe de base: `Zone_dis_base`
Classe de base représentant la discrétisation appliquée à une zone.
Possède une référence à la `Zone` correspondante.

Méthodes virtuelles pures de la classe `Zone_dis_base` :

`discretiser()` construction de tous les tableaux et les connectivités nécessaires au calcul pour la discrétisation choisie.

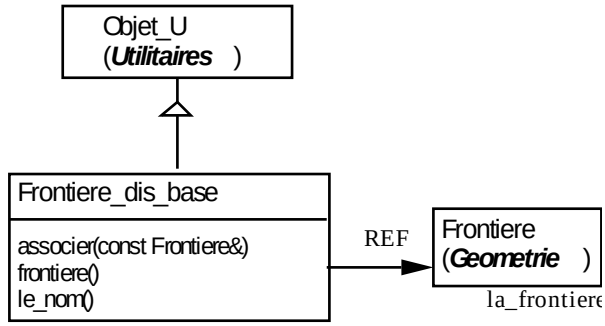
`frontiere_dis(int)` Envoie de la `Frontiere_dis_base` de rang i



Noyau : Domaine_dis , Zone_dis

8.4 Discrétisation des Frontières :

Classe de base: `Frontiere_dis_base`
Classe de base représentant la discrétisation appliquée à une frontière.



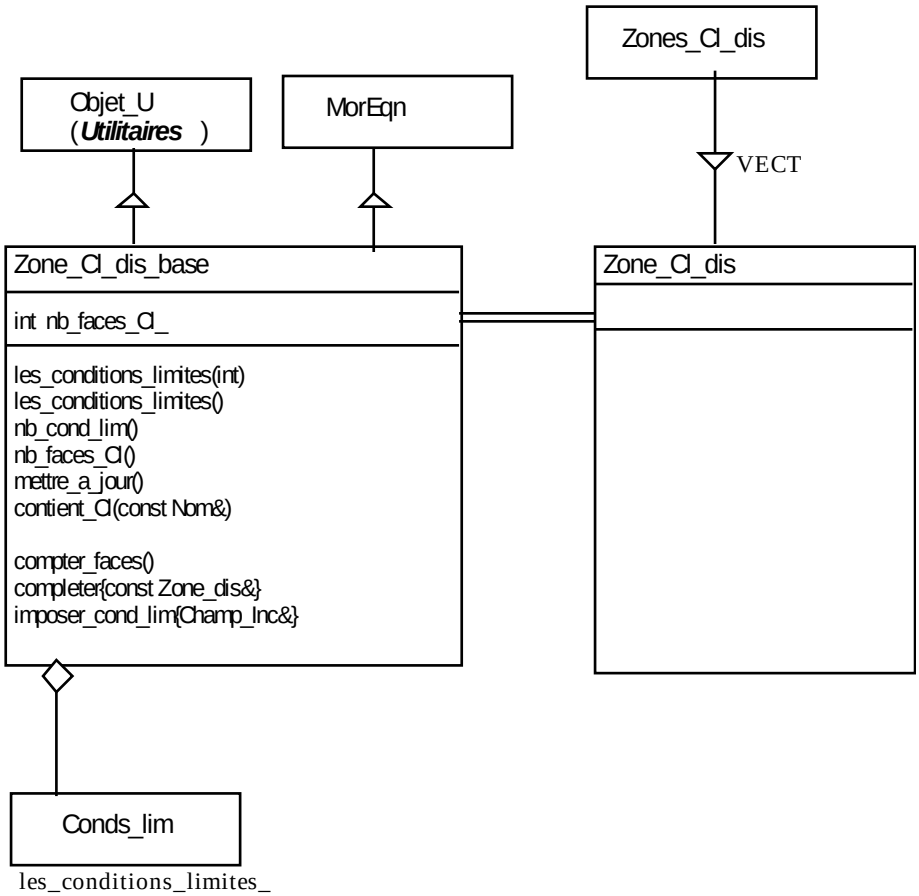
Noyau : Frontiere_dis_base

8.5 Discrétisation des conditions limites:

Généralités: Les conditions aux limites (`Cond_lim`) sont associées aux équations. Les objets `Zone_Cl_dis` contiennent les conditions limites et les connectivités nécessaires à la discrétisation des conditions limites.

Classe générique: `Zone_Cl_dis`

Classe de base: `Zone_Cl_dis_base`



Noyau : Zone_Cl_dis

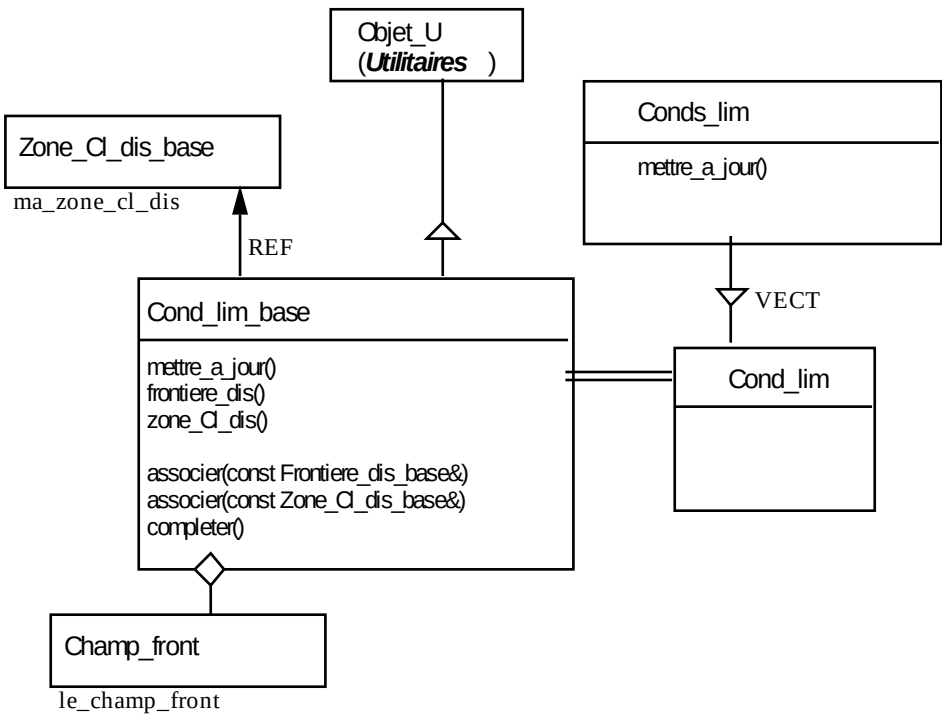
9-NOYAU : CONDITIONS LIMITES

Généralités: Un objet condition aux limites sert à définir, pour une équation donnée, des conditions limites sur une frontière d'un domaine.

Classe générique: `Cond_lim`

Classe de base: `Cond_lim_base`
Porte un objet de type `Champ_front` qui contient les valeurs imposées sur la frontière.

Remarque: les conditions aux limites (classe `Conds_lim`) sont portées par la classe `Zone_Cl_dis` affectée à une équation.



Noyau : Cond_lim

10-NOYAU : LES CHAMPS

Généralités: L'objet `Champ` représente un champ de valeurs.
Les données suivantes définissent un champ : un nom, une unité, un instant, des valeurs. Si le champ n'est pas uniforme en espace, les valeurs sont définies en référence à une géométrie discrétisée; la localisation des valeurs dépend de la discrétisation (faces, noeuds,...).

L'objet Champ contient en outre des méthodes telles que:
- des opérations arithmétiques,
- des méthodes d'accès aux valeurs,
- des méthodes de postraitement (formattage, interpolation...)

Classe de base: `Champ_base`
Cette classe possède les méthodes virtuelles pures `valeur_a(---)` , `valeur_aux(---)` qui interpolent les valeurs du champ en des points donnés. Toutes les classes de champ devront implémenter ces méthodes.

Prototype fonctionnel : `Champ_Proto` . Tous les champs vont hériter de cette classe

On distingue deux grands types de champ:

- les champs calculés par une équation: les `Champ_Inc`. On doit garder en mémoire un certain nombre d'états de ces champs: t^{n-1}, t^n, t^{n+1}, ... C'est le schéma en temps qui gère ce nombre d'états.

- tous les autres champs: les `Champ_Don` champ constant,
champ fonction du temps,
champ fonction d'un ou plusieurs champs
Ces champs vont posséder la méthode `me_calculer()` pour en déterminer les valeurs.

10.1 Champ_Inc

Généralités: Les `Champ_Inc` sont des champs calculés par une équation.
Une `Roue` leur est associée. Cette `Roue` permet de gérer le nombre de valeurs du temps pour lesquels le champ reste en mémoire. Dans le cas d'un schéma en temps explicite, il suffit de conserver les valeurs du champ aux temps t^n et t^{n+1}.

Classe générique: `Champ_Inc`

Classe de base: `Champ_Inc_base`

10.2 Champ_Don

Généralités: Ce sont les champs qui, entre autres, fournissent une caractéristique physique du milieu.

Classe générique: `Champ_Don`

Classe de base: `Champ_Don_base`

10.3 Champ_Fonc

Généralités: `Champ_Don` particuliers qui sont fonction d'autres champs.

Classe générique: `Champ_Fonc`

Classe de base: `Champ_Fonc_base`

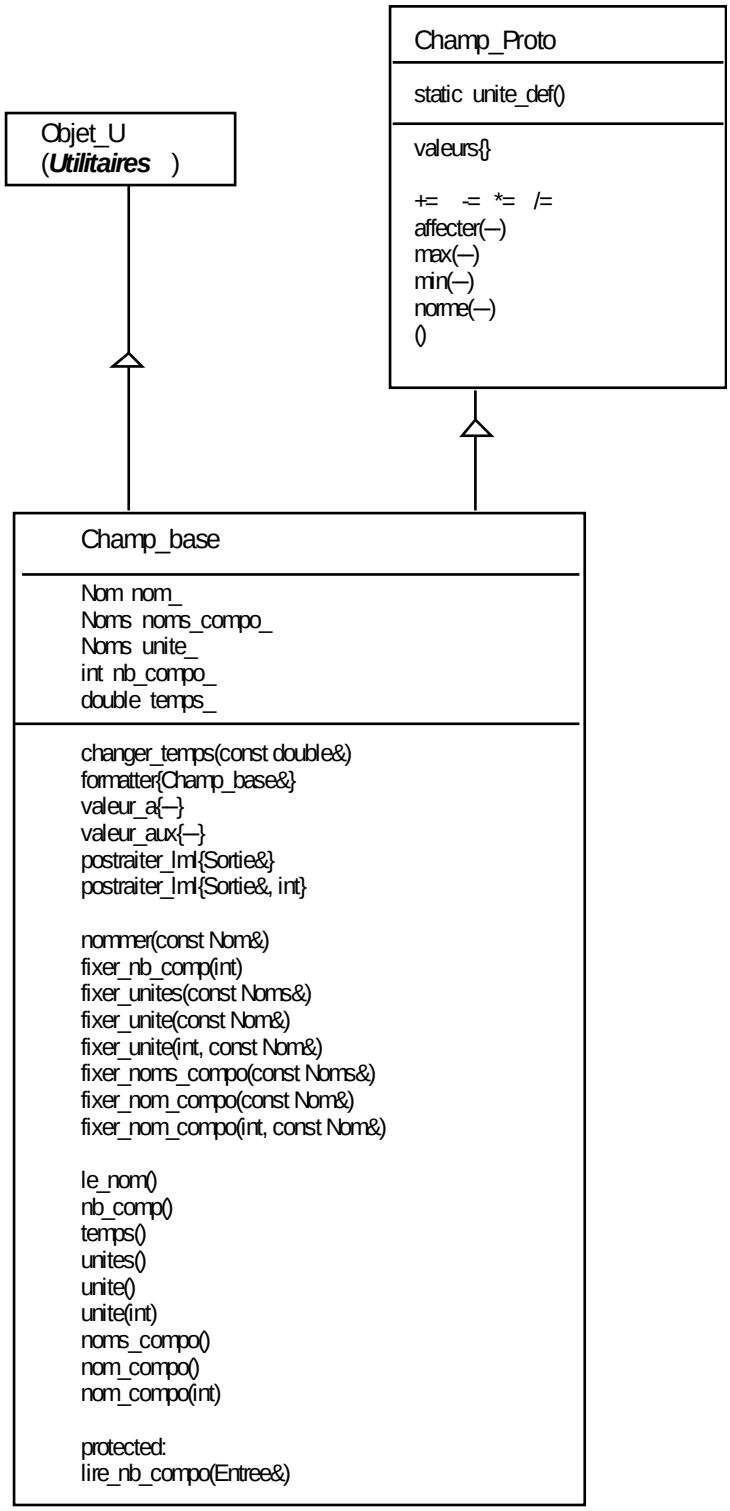
10.4 Champ_front

Généralités: Un `Champ_front` est un objet qui définit un champ sur une frontière d'un domaine. Cet objet sera associé par exemple à une condition limite.
Les `Champ_front` héritent de la classe prototype `Champ_Proto`.

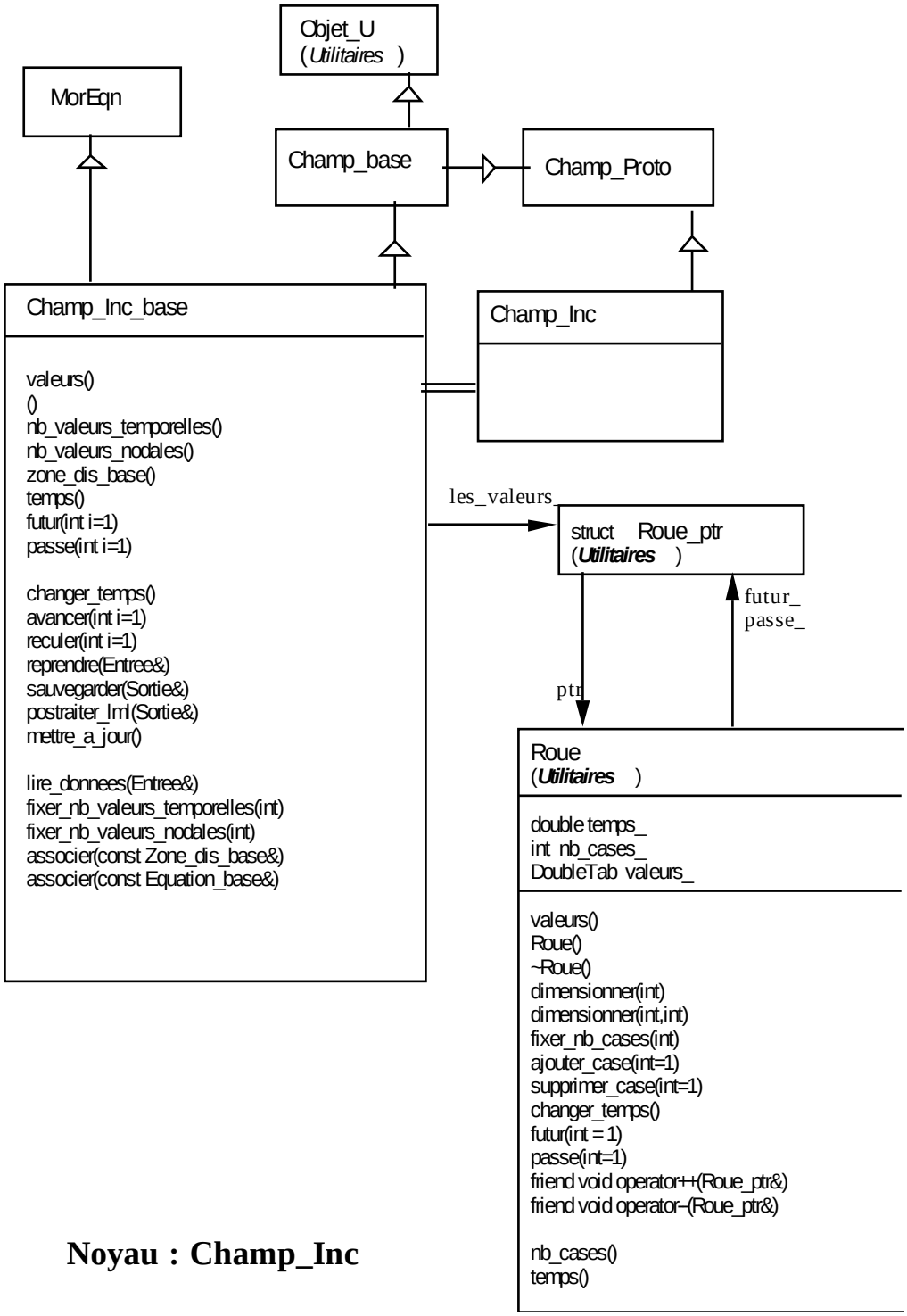
Classe générique: `Champ_front`

Classe de base: `Champ_front_base`

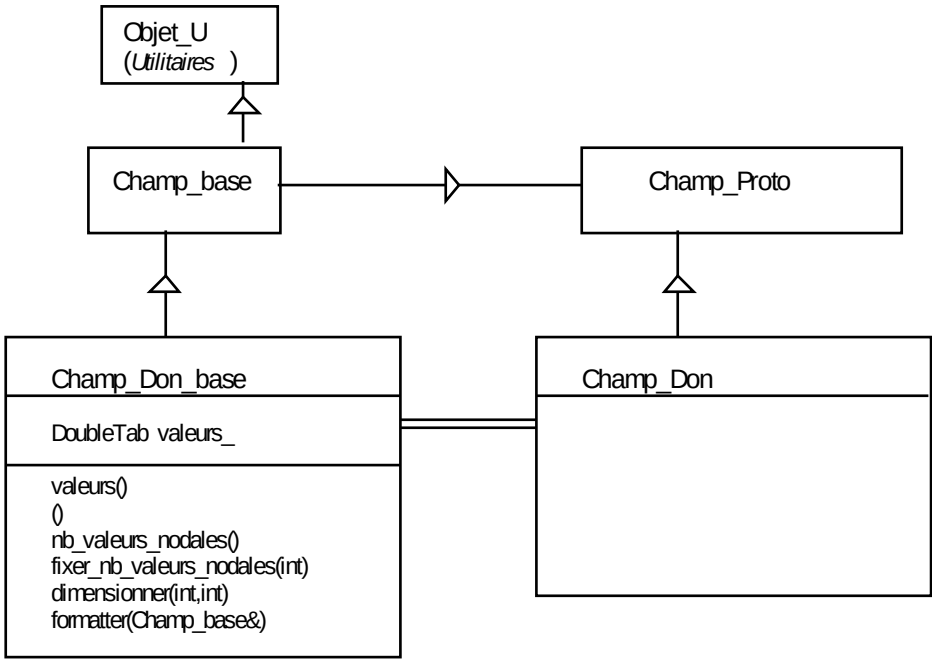
Relations avec les autres classes: L'utilisation des champs est multiple. Néanmoins on peut donner les utilisations les plus fréquentes dans TRIO_U.
Les `Champ_Inc` sont portés par les équations.
Les `Champ_Don` sont portés par le Milieu.
Pour l'instant on trouve des `Champ_Fonc` dans les modèles de turbulence.
Les `Champ_front` sont portés par les conditions aux limites.



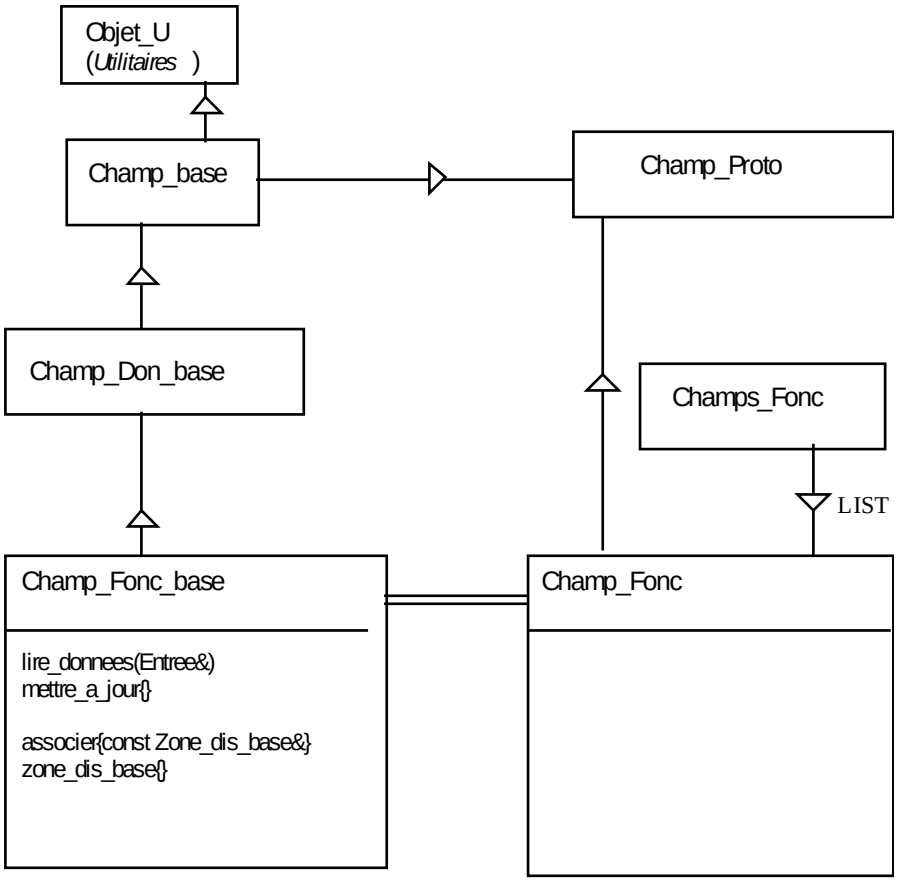
Noyau : Champ_base , Champ_Proto



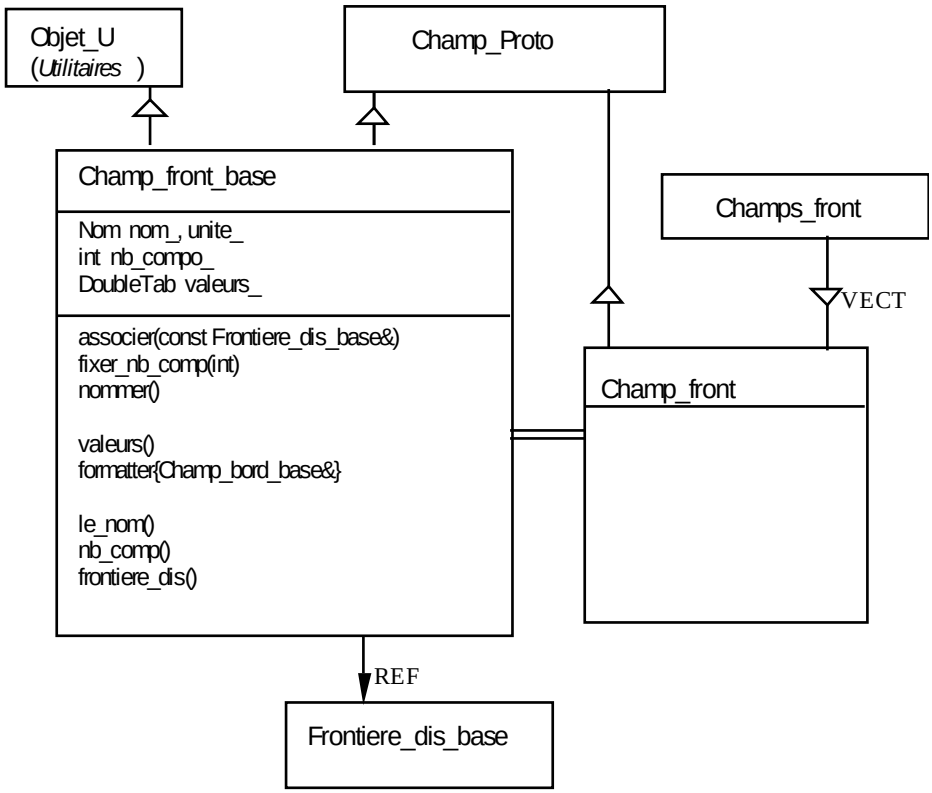
Noyau : Champ_Inc



Noyau : Champ_Don



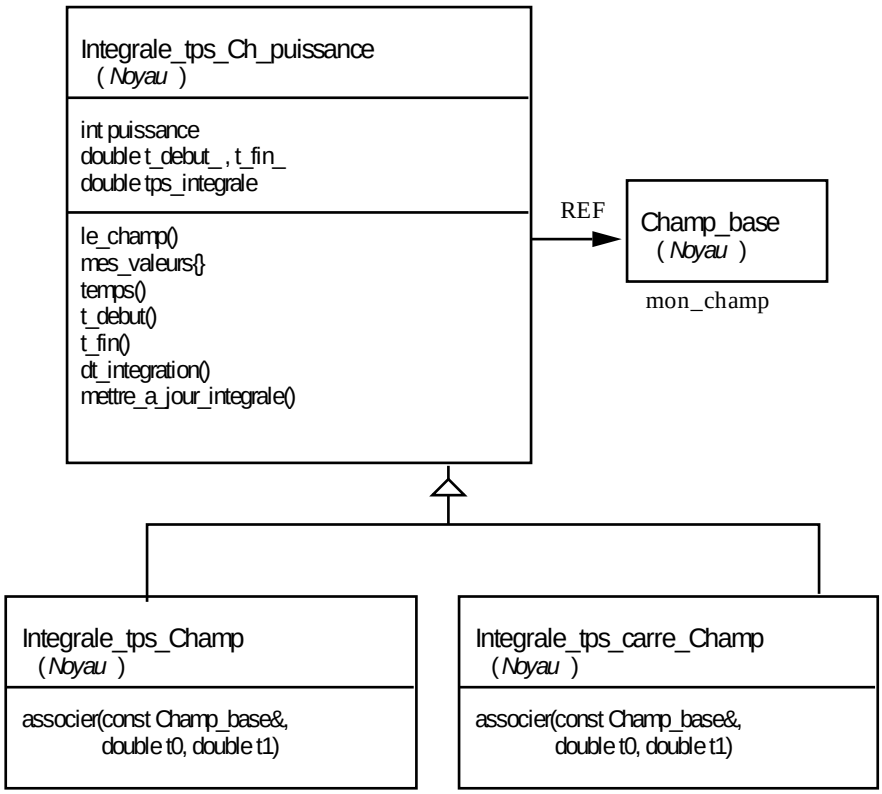
Noyau : Champ_Fonc



Noyau : Champ_front

Intégrale d'un champ

`Integrale_tps_Ch_puissance` : représente l'intégrale en temps d'un champ élevé à une puissance entière et positive entre un temps initial et un temps courant. Cette classe est hors hiérarchie.
`Integrale_tps_Champ` : cas particulier de `Integrale_tps_Ch_puissance` pour une puissance de 1
`Integrale_tps_carre_Champ` : cas particulier de `Integrale_tps_Ch_puissance` pour le champ élevé au carré.



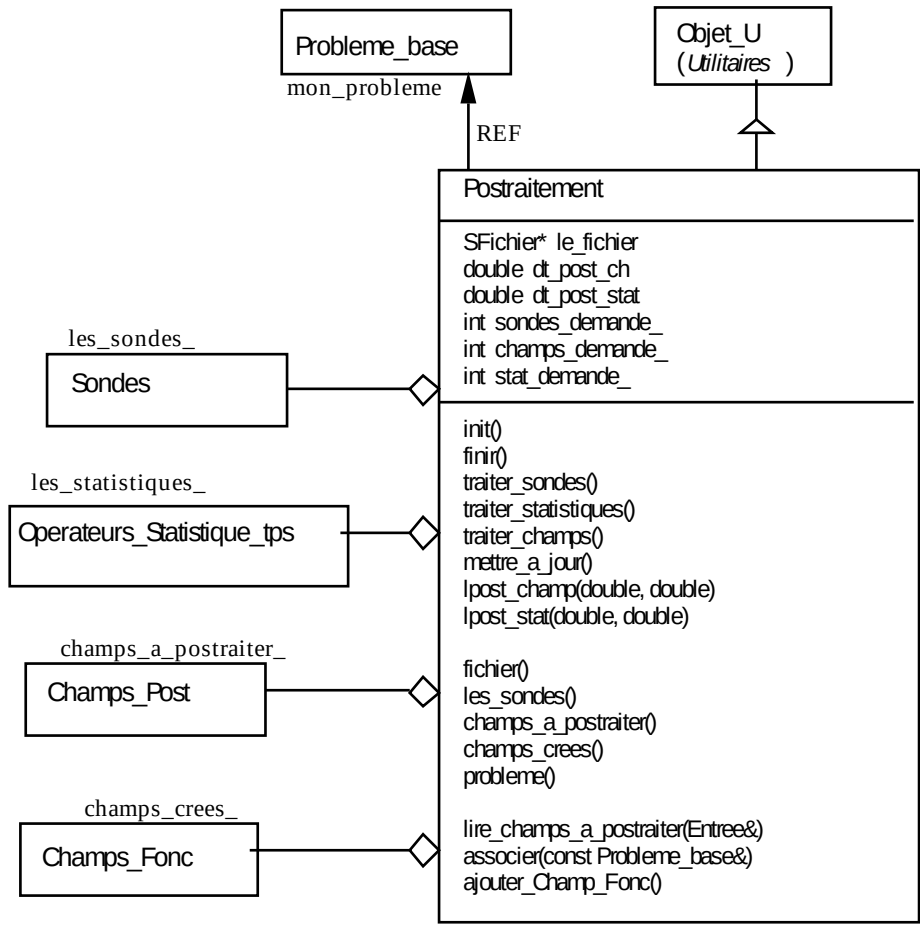
Noyau : Integrale_tps_Champ_puissance

11-NOYAU : POSTRAITEMENT

11.1 Postraitement

`Postraitement` : Cette classe permet le postraitement suivant deux approches:
- à l'aide de sondes (valeurs d'un champ en un ou plusieurs points en fonction du temps)
- à l'aide de champs sélectionnés

Lien avec les autres objets: L'objet `Postraitement` est porté par un objet `Probleme_base`

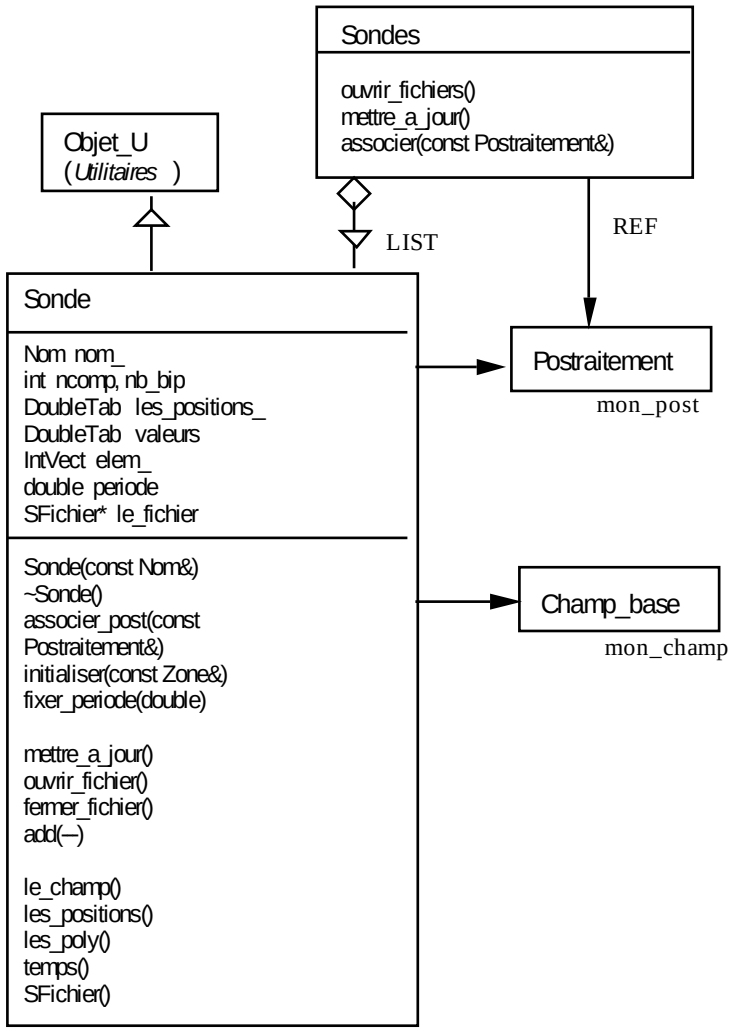


Noyau : Postraitement

11.2 Sonde

`Sonde` : Ensemble de points sur lesquels on veut connaître un champ en fonction du temps. La méthode `mettre_a_jour()` écrit sur le fichier.

Lien avec les autres objets : L'objet `Postraitement` porte les `Sondes`.

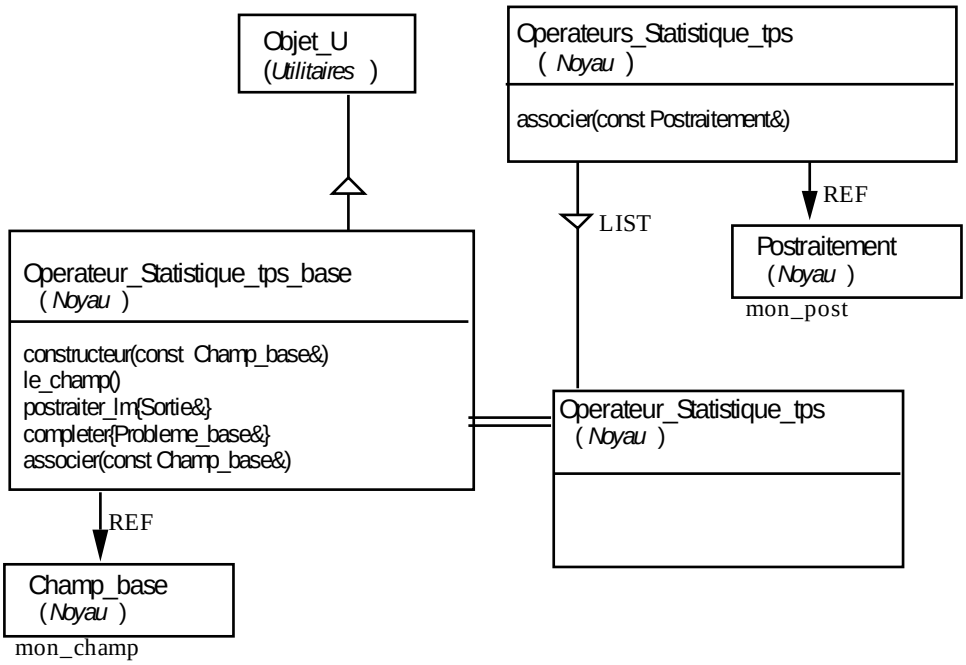


Noyau : Sondes

11.3 Operateur_Statistique_tps

`Operateur_Statistique` : classe générique
`Operateur_Statistique_tps_base` : classe de base des opérateurs statistiques sur les champs.

Lien avec les autres objets : L'objet `Postraitement` porte les `Operateurs_Statistiques_tps`.

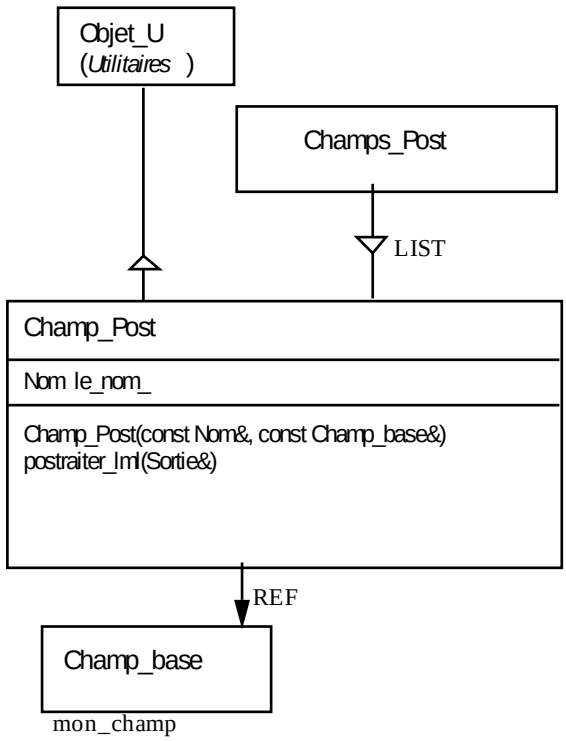


Noyau : Opérateurs statistiques

11.4 Champ_Post

`Champ_Post` : Un `Champ_Post` est un objet qui définit un champ à postraiter. Il porte donc une référence à ce champ.

Lien avec les autres objets : L'objet `Postraitement` porte les `Champs_Post`.



Noyau : Champ_Post

TRIO_U : Champs

Le Module **Champs** contient des classes contenant des champs particuliers, non discrétisés.

On a défini les familles d'objets suivantes:

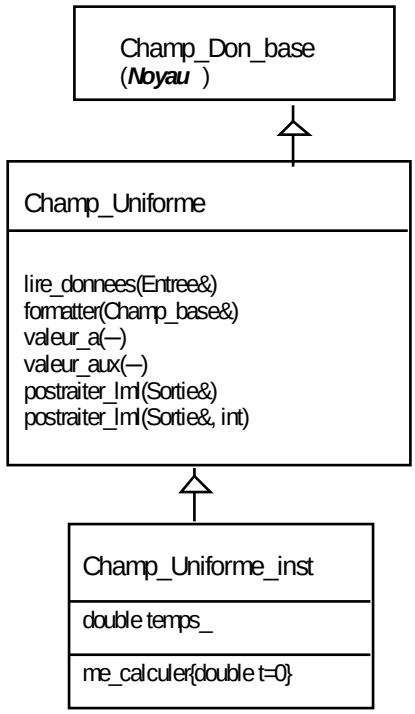
1 Champ_Uniforme
2 Champ_front_uniforme

1- Champ_Uniforme

Hiérarchie:		
Classe de base:	`Champ_Don_base`	(Noyau)
Classe générique:	`Champ_Don`	(Noyau)

`Champ_Uniforme` : champ constant dans l'espace et dans le temps

`Champ_Uniforme_inst` : champ constant dans l'espace mais dépendant du temps

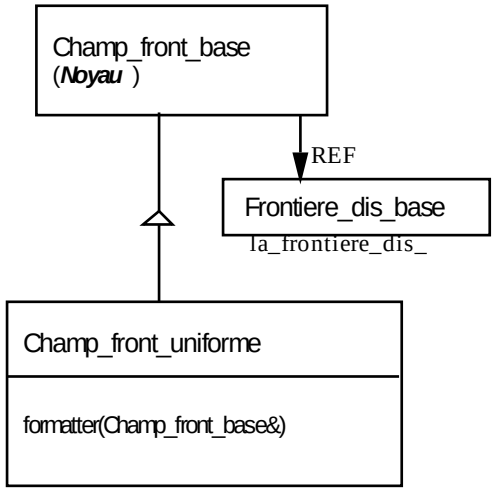


Champs : Champ_Uniforme

2- Champ_front_uniforme

Hiérarchie:	
Classe de base:	`Champ_front_base` (Noyau)
Classe générique:	`Champ_front` (Noyau)

`Champ_front_uniforme` : champ constant dans l'espace et dans le temps défini sur une frontière



Champs : Champ_front_uniforme

TRIO_U : Champs_dis

Le Module **Champs_dis** contient les classes de base pour la discrétisation des champs.

On a défini les familles d'objets suivantes:

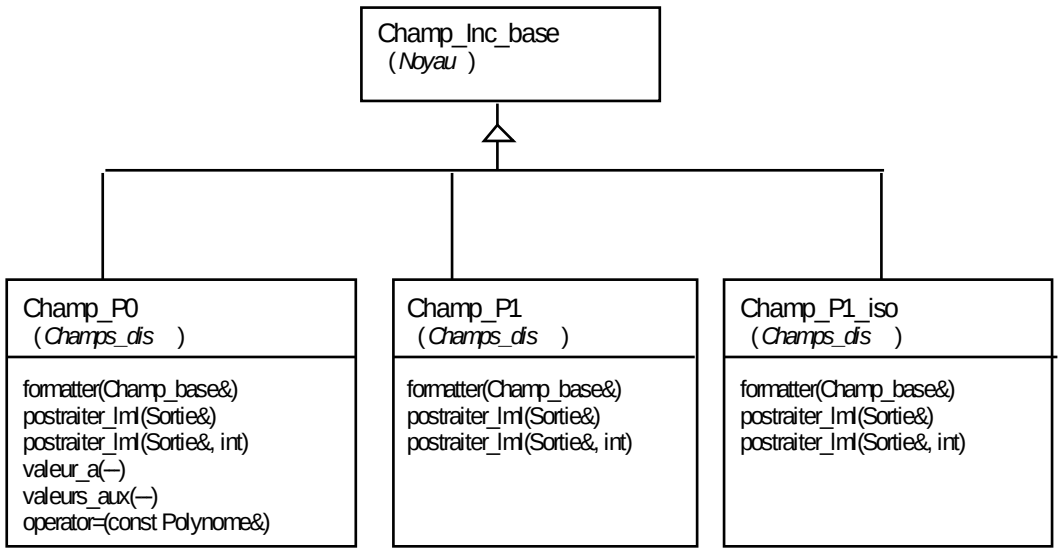
1	Discrétisation des Champ_Inc
2	Discrétisation des Champ_Fonc

1- Discrétisation des Champ_Inc

Hiérarchie:		
Classe de base:	`Champ_Inc_base`	(Noyau)
Classe générique:	`Champ_Inc`	(Noyau)

`Champ_P0` : classe de base pour un `Champ_Inc` de type P0 (champ constant par élément). Pour l'instant il existe deux classes dérivées instanciables : `Champ_P0_VDF` (Module VDF) et `Champ_P0_VEF` (Module VEF).

`Champ_P1` : classe de base pour un `Champ_Inc` de type P1

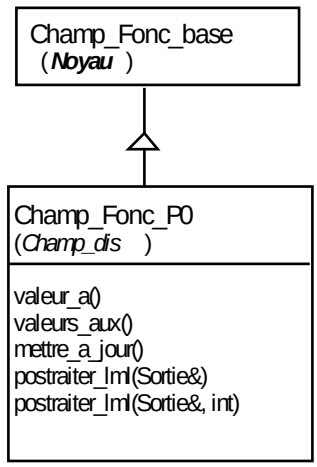


Champs_dis : Champ Inc

2- Discrétisation des Champ_Fonc

Hiérarchie:		
Classe de base:	`Champ_Fonc_base`	(Noyau)
Classe générique:	`Champ_Fonc`	(Noyau)

`Champ_Fonc_P0` : classe de base pour un `Champ_Fonc` de type P0



Champs_dis : Champ Fonc

TRIO_U : Statistiques_temps

Le Module **Statistiques_temps** contient les classes de base pour les calculs statistiques relatifs aux champs.

On calcule :

Moyenne d'un champ
Ecart type d'un champ

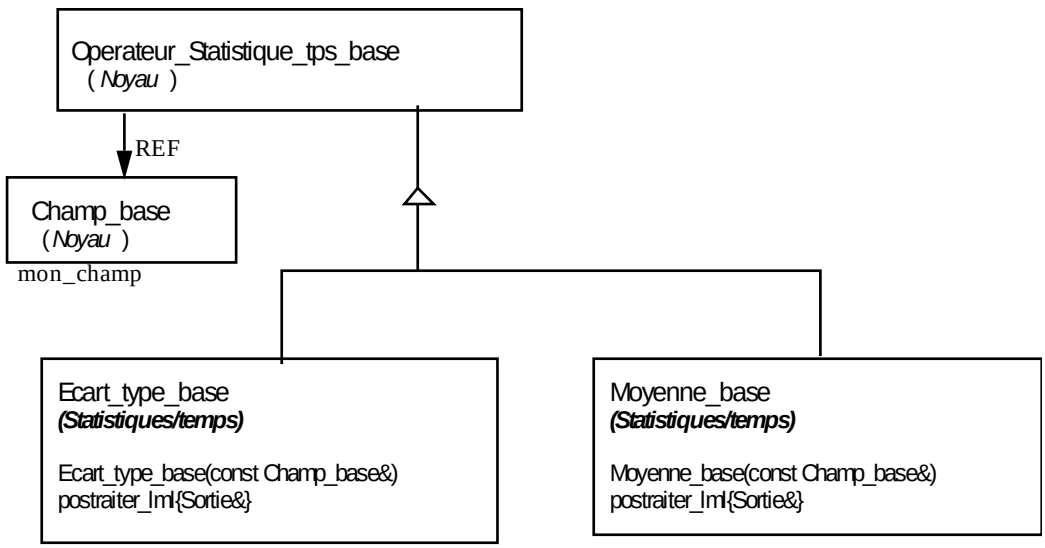
Moyenne et Ecart type d'un champ

Hiérarchie:		
Classe de base:	`Operateur_Statistique_tps_base`	(Noyau)
Classe générique:	`Operateur_Statistique_tps`	(Noyau)

`Moyenne_base` : classe de base pour le calcul de la moyenne d'un champ. Il faut connaître la discrétisation du champ pour définir la classe instanciable correspondante.

`Ecart_type_base` : classe de base pour le calcul de l'écart type d'un champ. Il faut connaître la discrétisation du champ pour définir la classe instanciable correspondante.

Lien avec les autres classes: La classe `Postraitement` porte des `Operateur_Statistique_tps`.



Statistiques_temps : Moyenne , Ecart_type

TRIO_U : ThHYD

Le Module ThHyd contient les objets nécessaires au calcul de la thermo-hydraulique.

On a défini les familles d'objets suivantes:

1. Problème
   - 1.1 Écoulement laminaire
   - 1.1 Écoulement turbulent
2. Équation
   - 2.1 Navier_Stokes
   - 2.2 Convection Diffusion
   - 3.3 Transport_K_Eps
3. Milieu
4. Discrétisation
5. Conditions limites
   - 5.1 Dirichlet
   - 5.2 Neumann
   - 5.3 Symétrie
   - 5.4 Échange imposé
6. Solveur Pression
7. Modèles de turbulence
   - 7.1 Modèles de turbulence (hyd)
   - 7.2 Modèles de turbulence (scalaire)
   - 7.3 Lois de paroi
8. Sources
   - 8.1 Boussinesq
   - 8.2 Puissance thermique

1- ThHyd : LES PROBLEMES

Hiérarchie:		
Classe de base:	`Probleme_base`	(Noyau)
Classe générique:	`Probleme`	(Noyau)

1.1 Problèmes thermohydrauliques traités en laminaire :

`Pb_Hydraulique` : problème d'hydraulique standard dans lequel on résout les équations de Navier_Stokes en régime laminaire pour un fluide incompressible.
La formulation est de type vitesse pression.

`Pb_Thermohydraulique` : problème d'hydraulique standard dans lequel on résout en régime laminaire pour un fluide incompressibe et faiblement dilatable:
- les équations de Navier_Stokes,
- l'équations d'énergie.
L'hydraulique et la thermique sont couplées par des forces de volume (approximation de Boussinesq).

`Pb_Hydraulique_Concentration` : problème d'hydraulique avec transport d'un ou plusieurs constituants. On résout en régime laminaire pour un fluide incompressible:
- les équations de Navier_Stokes,
- les équations de convection-diffusion pour un ou plusieurs constituants.
Si on transporte plusieurs constituants, on utilise une seule équation de convection diffusion avec une inconnue vectorielle.

En général on couple les deux équations par l'intermédiaire du terme source des forces de volume de Navier_Stokes dans lequel on prend en compte de petites variations de la masse volumique en fonction du (ou des) constituant(s).

`Pb_Thermohydraulique_Concentration` : problème de thermohydraulique avec transport d'un ou plusieurs constituants. On résout en régime laminaire pour un fluide incompressible:
- les équations de Navier_Stokes,
- l'équations d'énergie.
- les équations de convection-diffusion pour un ou plusieurs constituants.
Si on transporte plusieurs constituants, on utilise une seule équation de convection diffusion avec une inconnue vectorielle.

1.2 Problèmes thermohydrauliques traités avec un modèle de turbulence :

`Pb_Hydraulique_Turbulent` : problème d'hydraulique standard dans lequel on résout les équations de Navier_Stokes en régime turbulent pour un fluide incompressible.
La formulation est de type vitesse pression.

`Pb_Thermohydraulique_Turbulent` : problème d'hydraulique standard dans lequel on résout en régime turbulent pour un fluide incompressible:

- les équations de Navier_Stokes,
- les équations d'énergie.

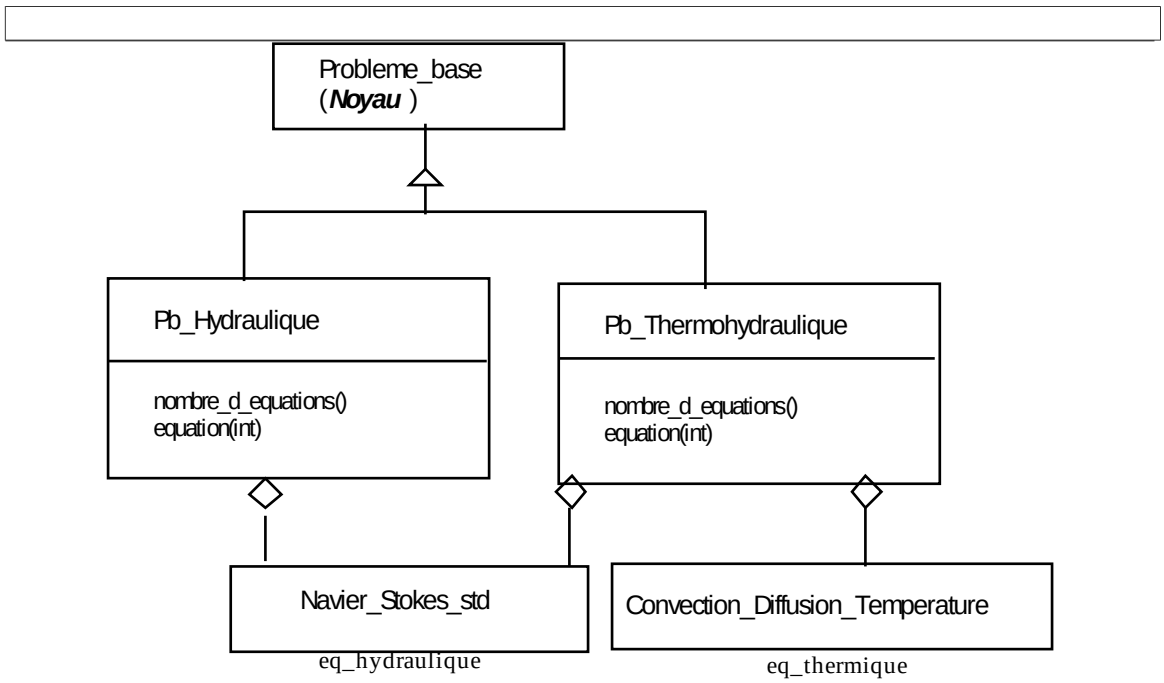
L'hydraulique et la thermique sont couplées par des forces de volume (approximation de Boussinesq).

`Pb_Hydraulique_Concentration_Turbulent` : problème d'hydraulique avec transport d'un ou plusieurs constituants. On résout en régime turbulent pour un fluide incompressible:

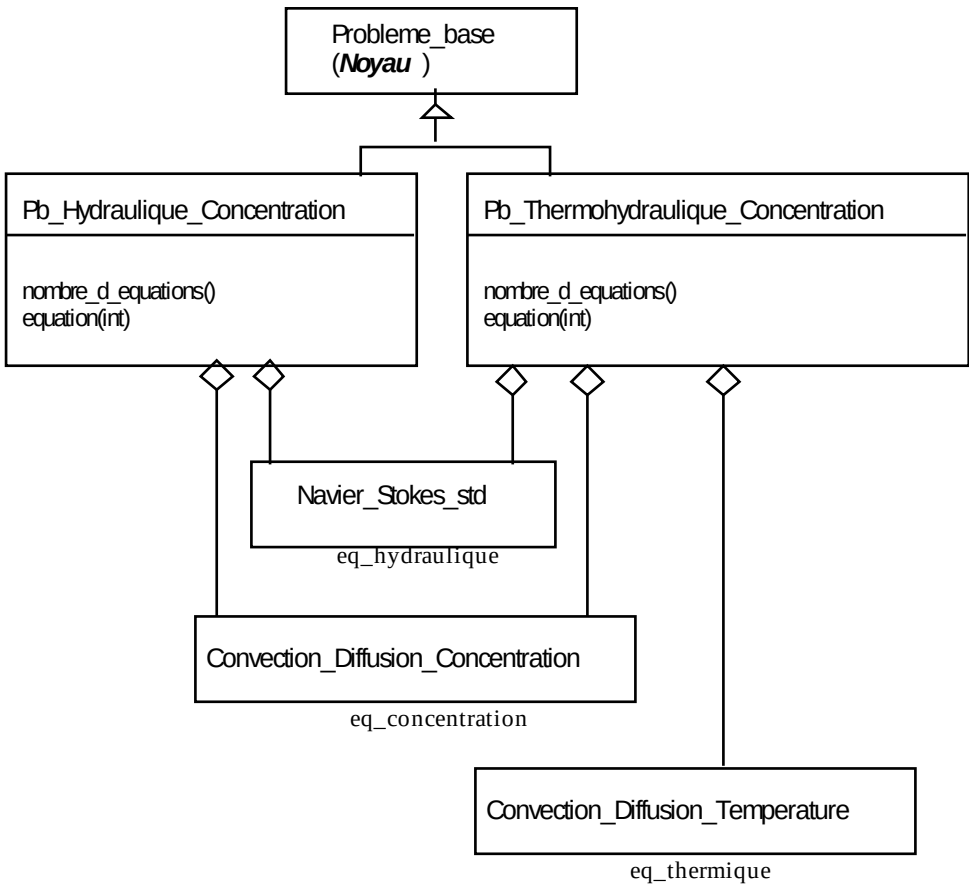
- les équations de Navier_Stokes,
- les équations de convection-diffusion pour un ou plusieurs constituants.

`Pb_Thermohydraulique_Concentration_Turbulent` : problème d'hydraulique standard dans lequel on résout en régime turbulent pour un fluide incompressible:

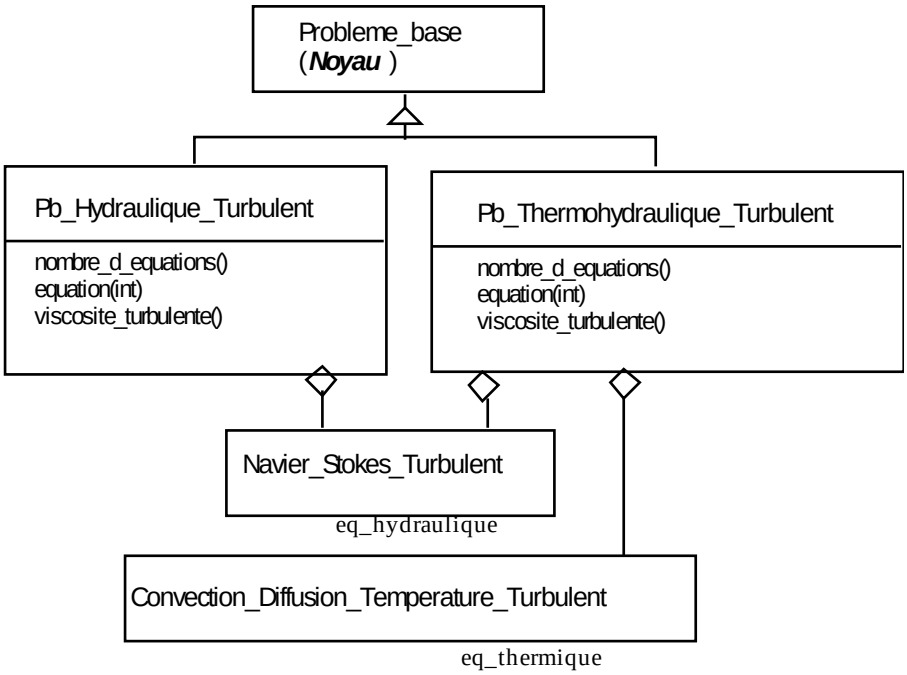
- les équations de Navier_Stokes,
- les équations d'énergie
- les équations de convection-diffusion pour un ou plusieurs constituants.



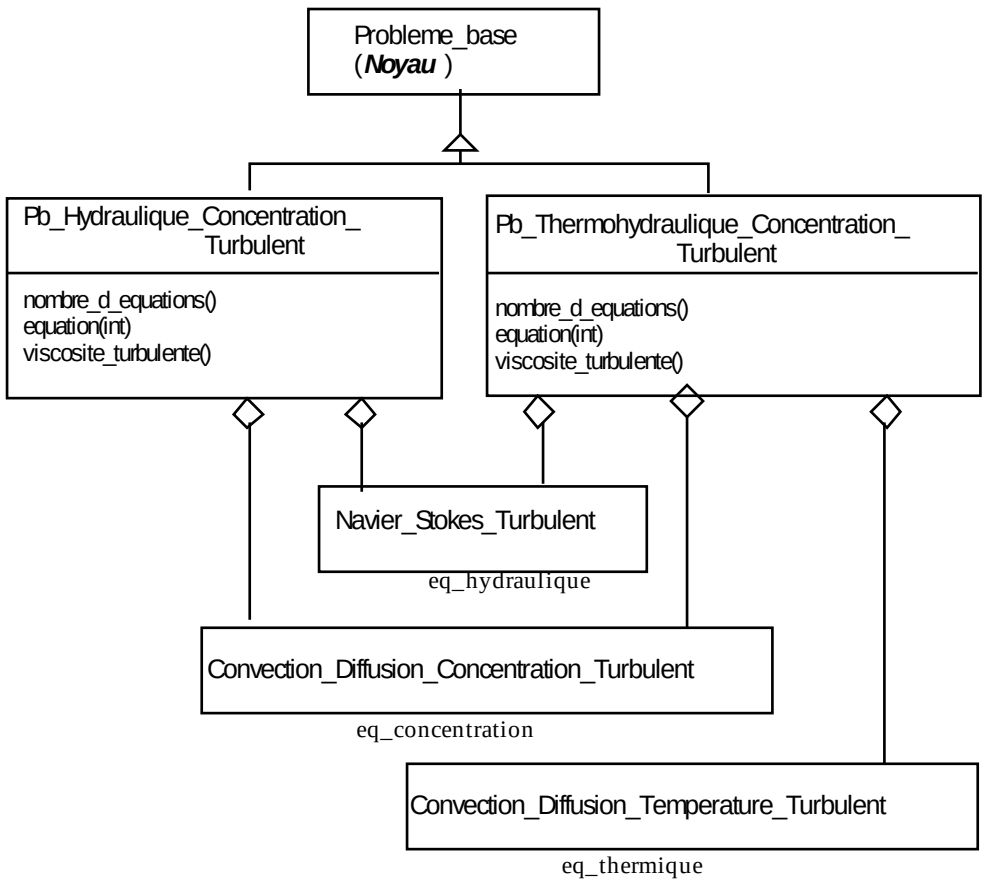
ThHyd : Pb_Hydraulique , Pb_Thermohydraulique



ThHyd : Pb_Hydraulique_Concentration
Pb_Thermohydraulique_Concentration



ThHyd : Pb_Hydraulique , Pb_Thermohydraulique traités en turbulent



ThHyd : Pb_Hydraulique_Concentration_Turbulent , Pb_Thermohydraulique_Concentration_Turbulent

2- ThHyd : LES EQUATIONS

Hiérarchie:		
Classe de base:	`Equation_base`	(Noyau)
Classe générique:	`Equation`	(Noyau)

Les équations sont décrites dans le document de présentation de TRIO_U [PRES1])

2.1 Les équations de Navier_Stokes

L'équation Navier_Stokes décrit les équations de quantité de mouvement.

$$\frac{\partial \vec{U}}{\partial t} + \vec{\nabla}\left(\vec{U}\times\vec{U} + \overline{\vec{u}\times\vec{u}} + \frac{P^*}{\rho_0}\Pi - \nu\vec{\nabla}\vec{U}\right) - \vec{F} = 0$$

(voir TRIO_U Document de présentation [PRES1])

$\vec{U}\times\vec{U}$	terme convectif
$\nu\vec{\nabla}\vec{U}$	terme diffusif
$\overline{\vec{u}\times\vec{u}}$	composantes du tenseur de Reynolds
$\vec{F}$	Les forces de volume
P^*	la somme de la pression statique et de sa composante hydrostatique

On considère le fluide comme **incompressible** (div U =0). On considère la masse volumique constante (égale à ρ_0), sauf dans le terme des forces de gravité (hypothèse de Boussinesq).
Sous ces hypothèses, on utilise la forme suivante des équations de Navier_Stokes:

$$\frac{DU}{dt} = \text{div (terme\_visqueux)} - \frac{\text{grad}P}{\rho_0} + \beta(T-T0)g + \frac{\text{sources}}{\rho_0} = 0 \quad \text{et} \quad \text{div } U = 0$$

$\frac{DU}{dt}$	dérivée particulaire de la vitesse
ρ_0	masse volumique de référence
β	coefficient de dilatabilité du fluide
g	gravité

Les classes:

`Navier_Stokes_std` : Equation de la dynamique pour un fluide incompressible en régime **laminaire** sans modélisation de la turbulence.

L'inconnue est le champ de vitesse.

`Navier_Stokes_Turbulent:` Cette classe représente l'équation Navier_Stokes pour un écoulement **turbulent**. Elle va donc porter un modèle de turbulence et instancier un opérateur de diffusion qui tient compte de la viscosité laminaire et turbulente.

Liens avec les autres classes
`Navier_Stokes_std` est porté par les classes des problèmes:
`Pb_Hydraulique`
`Pb_Thermohydraulique`
`Pb_Concentration`.
`Navier_Stokes_Turbulent` est porté par les classes des problèmes:
`Pb_Hydraulique_Turbulent`
`Pb_Thermohydraulique_Turbulent`.

2.2 Les équations de Convection_Diffusion

Les classes de la hiérarchie Convection_Diffusion représentent l'équation de transport d'un scalaire dans un écoulement de fluide visqueux incompressible.

$$\frac{\partial T}{\partial t} + \vec{\nabla} (T\vec{U} + \overline{\theta\vec{u}} - \alpha\vec{\nabla}T) - S_T = 0$$

Si T est le scalaire transporté (la température par exemple),

$\overline{\theta\vec{u}}$	est le flux de chaleur turbulent
S_T	est le terme source Q/rho/Cp
Q	est la puissance volumique dégagée au sein du fluide
α	la diffusivité de la température dans le fluide

`Convection_Diffusion_std` Classe de base pour l'équation de transport d'un scalaire en régime **laminaire**.
La classe porte une référence au champ de la vitesse transportante.

`Convection_Diffusion_Temperature` : cas particulier de `Convection_Diffusion_std` pour le transport de la température. L'inconnue est le champ de température.

`Convection_Diffusion_Concentration` : cas particulier de `Convection_Diffusion_std` pour le transport d'un ou plusieurs constituants. Dans le cas de plusieurs constituants, le `Champ_Inc` de concentration et le `Champ_Don` de la diffusivité ont plusieurs composantes. L'inconnue est le champ de concentration.

`Convection_Diffusion_Turbulent` Classe de base pour l'équation de transport d'un scalaire en régime **turbulent**. Cette classe est définie hors hiérarchie `Objet_U`.
La classe porte un modèle de turbulence.

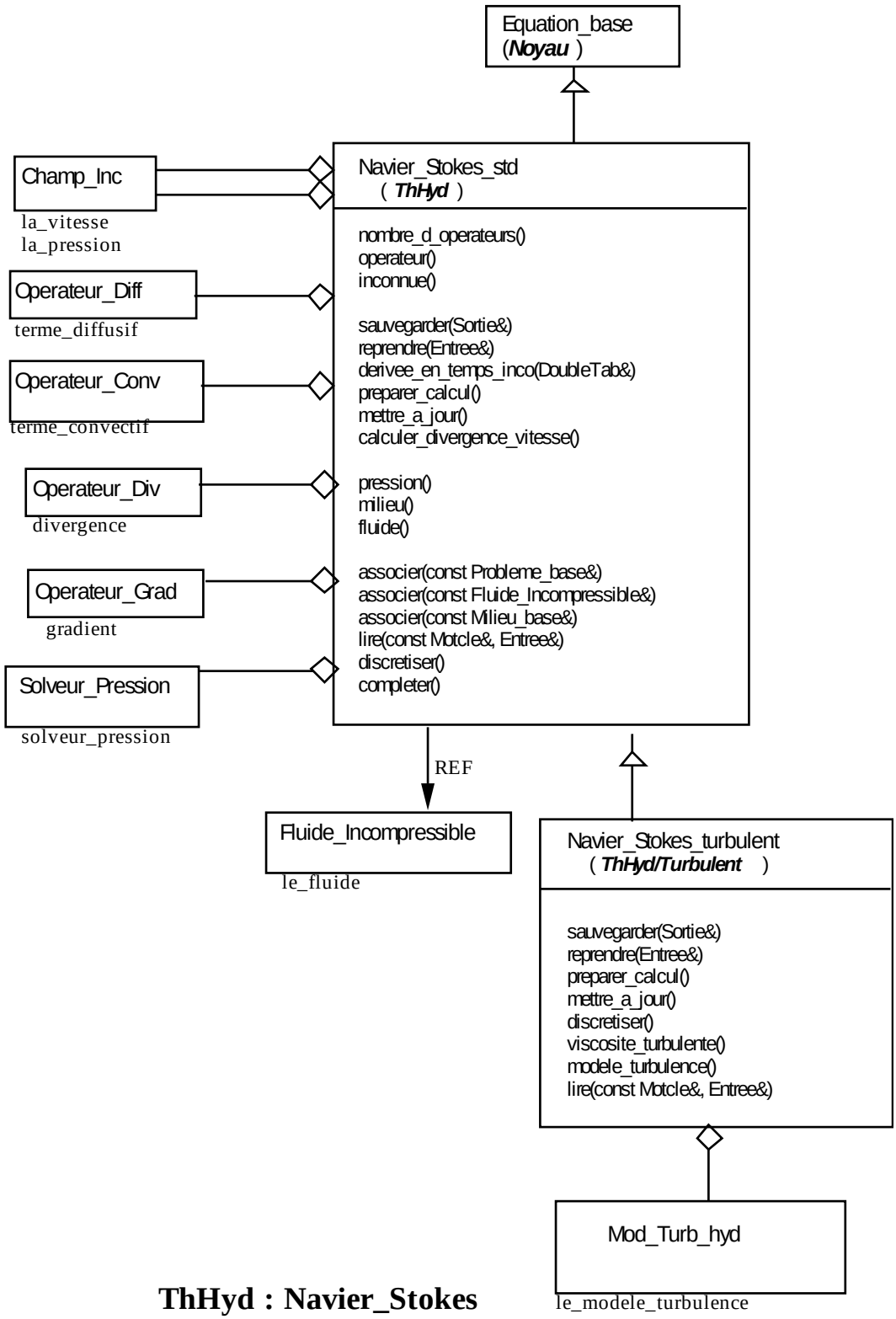
`Convection_Diffusion_Temperature_Turbulent` : équation pour le transport de la température en régime turbulent. L'inconnue est le champ de température.

`Convection_Diffusion_Concentration_Turbulent_` : équation pour le transport de constituants en régime turbulent. L'inconnue est le champ de concentration.

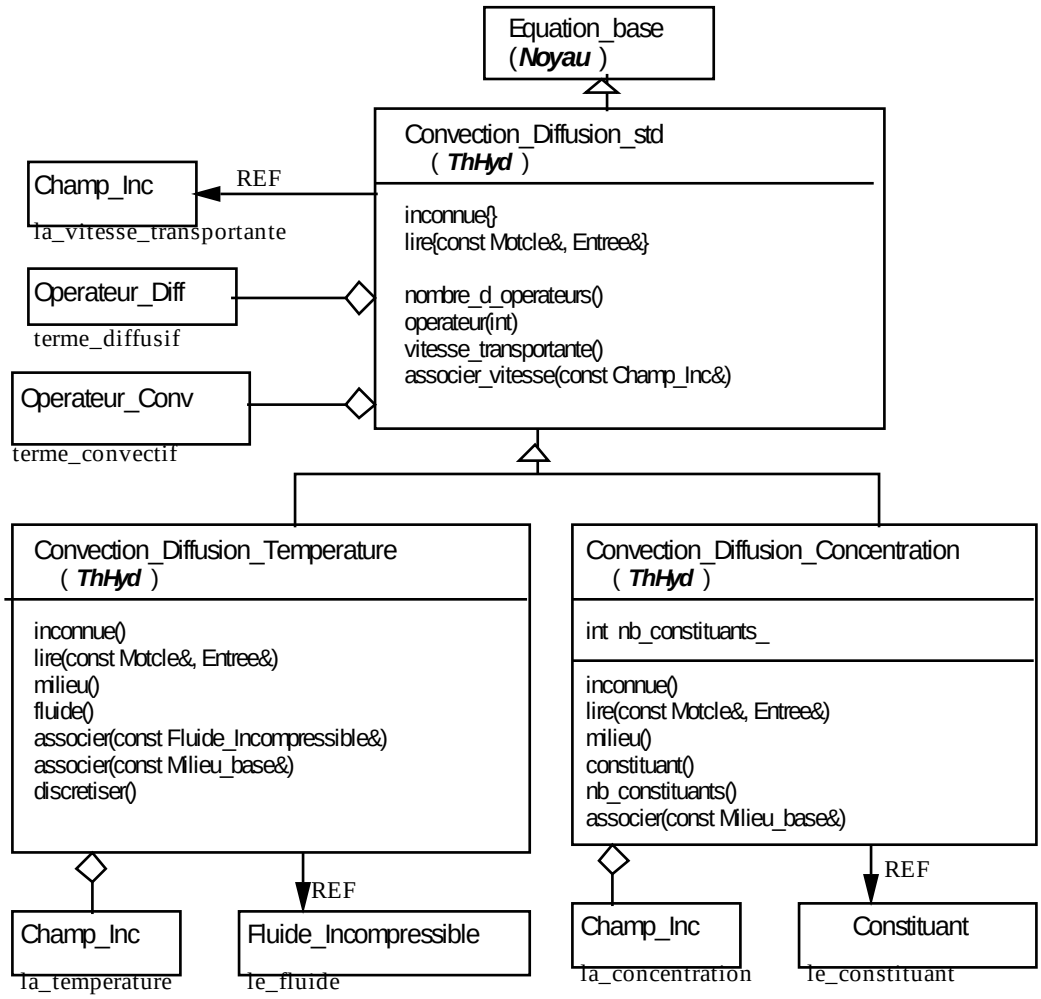
2.3 Equations de transport pour K, ε :

`Transport_K_Eps`: équation de transport de l'énergie cinétique turbulente K et du taux de dissipation ε, associée au modèle de turbulence K-ε. Le champ "inconnue" possède 2 composantes scalaires: K et ε .

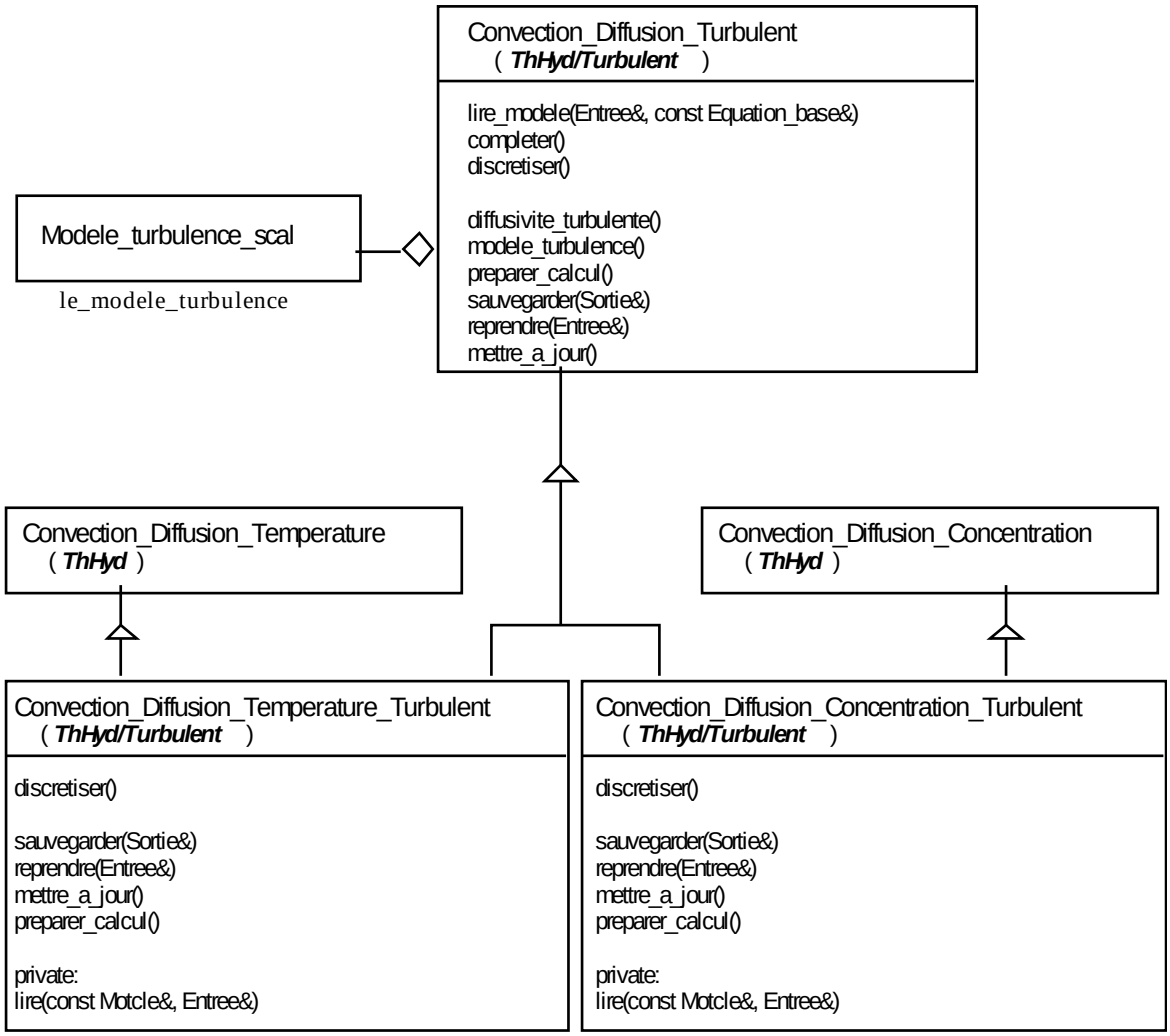
Liens avec les autres classes
`Transport_K_Eps` est porté par la classe: `Modele_turbulence_hyd_K_Eps` .



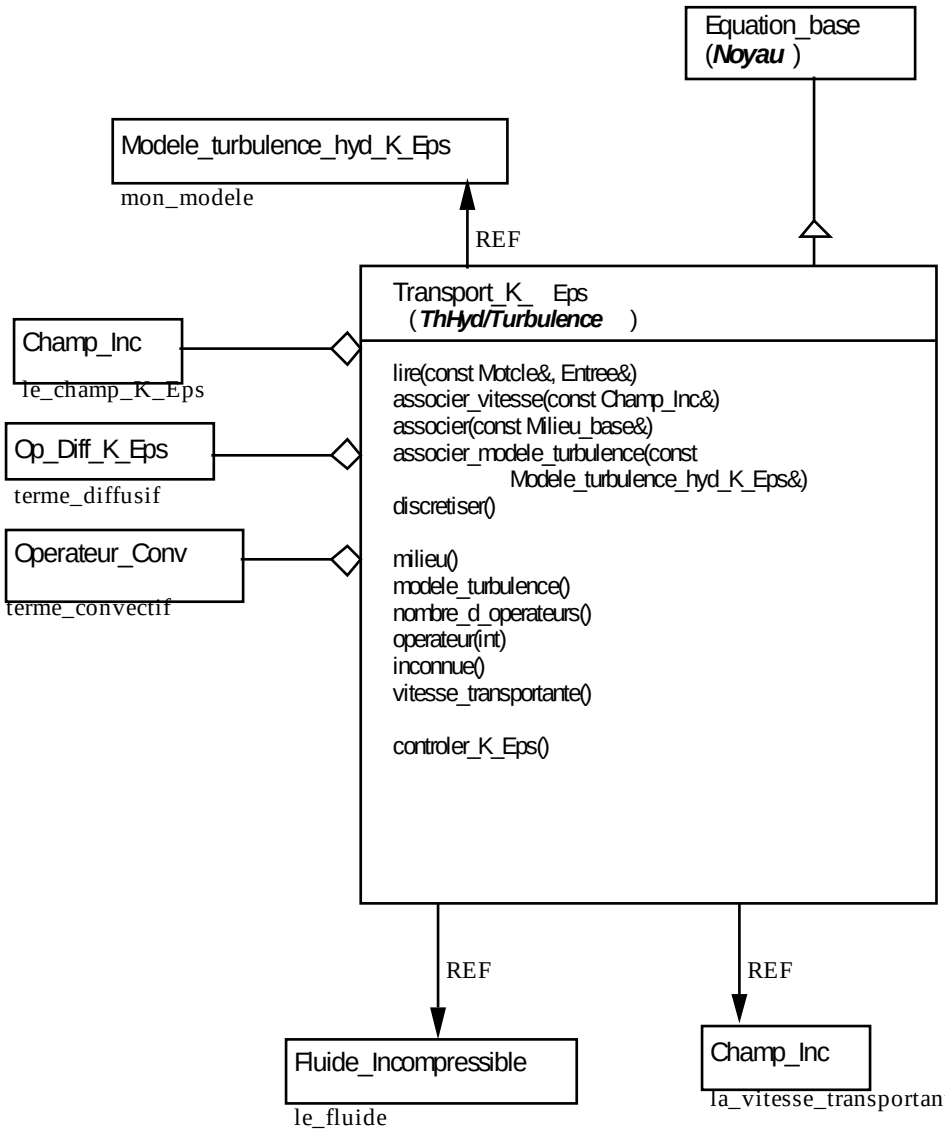
ThHyd : Navier_Stokes



ThHyd : Convection_Diffusion



ThHyd /Turbulence: Convection_Diffusion



ThHyd / Turbulence : Transport_K_Eps

3- ThHyd : MILIEU

Hiérarchie:		
Classe de base:	`Milieu_base`	(Noyau)
Classe générique:	`Milieu`	(Noyau)

Généralités: Les classes `Milieu` portent les `Champ_Don` contenant les propriétés du milieu fluide ou solide qu'on modélise.

`Fluide_Incompressible` : Cette classe contient des champs utiles au calcul hydraulique:

Le champ `mu` : viscosité dynamique (donnée utilisateur)

Le champ nu : viscosité cinématique (

)
Le champ `beta_co` : dilatabilité du constituant (donnée utilisateur) ou le coefficient de variation de la masse volumique en fonction de la concentration d'un constituant.

`Constituant :` Cette classe contient le champ `alpha` de diffusivité d'un constituant. Quand l'équation calcule la diffusion de plusieurs constituants, ce champ a autant de composantes qu'il y a de constituants.

Liens avec les autres classes

`Fluide_Incompressible` est référencé par les équations de type:
`Navier_Stokes_std`
`Convection_Diffusion_Temperature`
`Transport_K_Eps`

`Constituant` est référencé par les équations de type:
`Convection_Diffusion_Concentration`

4- ThHyd : Discrétisation

Hiérarchie:		
Classe de base:	`Discretisation_base`	(Noyau)
Classe générique:	`Discretisation`	(Noyau)

`Discret_Thyd` : Classe de base décrivant la discrétisation spatiale appliquée aux problèmes thermo-hydrauliques.

`Discret_Thyd_Turb` : Classe de base décrivant la discrétisation spatiale appliquée aux problèmes thermo-hydrauliques turbulents.

Liens avec les autres classes

`Discretisation_base` est associé à la classe `Probleme_base` .

Les méthodes virtuelles pures sont à implémenter pour chaque type de discrétisation pour typer et discrétiser les `Champ_Inc` et les `Champ_Fonc` portés par les équations de la thermo-hydraulique.

*5- **ThHyd : CONDITIONS LIMITES***

Hiérarchie:		
Classe de base:	`Cond_lim_base`	(Noyau)
Classe générique:	`Cond_lim`	(Noyau)

Liens avec les autres classes
`Conds_lim` , vecteur des conditions aux limites concernant une équation est porté par la classe `Zone_Cl_dis_base` .

5.1 Conditions limites de type Dirichlet

Les conditions aux limites de type **Dirichlet** imposent sur une frontière la valeur du champ inconnue. Les valeurs sont contenues dans `le_champ_front` , porté par la classe de base `Cond_lim_base`.

`Dirichlet_paroi_defilante` : impose la vitesse de paroi dans une équation de type `Navier_Stokes`.

`Dirichlet_paroi_fixe` : représente une paroi immobile dans une équation de type `Navier_Stokes`.

`Entree_fluide_vitesse_imposee` : impose la vitesse d'entrée du fluide dans une équation de type `Navier_Stokes`.

`Entree_fluide_temperature_imposee` : impose la température d'entrée du fluide dans une équation de type `Convection_Diffusion_Temperature`.

`Temperature_imposee_paroi` : impose la température dans la paroi dans une équation de type `Convection_Diffusion_Temperature`.

`Entree_fluide_concentration_imposee` : impose la concentration d'entrée du fluide dans une équation de type `Convection_Diffusion_Concentration`.

`Entree_fluide_K_Eps_imposee` : impose les valeurs de K et ε d'entrée du fluide dans une équation de type `Transport_K_Eps`.

5.2 Conditions limites de type Neumann

2) Les conditions aux limites de type **Neumann** imposent un flux sur un bord. Les valeurs du flux imposé sont contenues dans `le_champ_front` , porté par la classe de base `Cond_lim_base`.

`Neumann_paroi` : correspond à un flux imposé pour l'équation de transport d'un scalaire : par exemple, une paroi chauffante pour l'équation de transport de la température.

`Neumann_paroi_adiabatique` : définit une paroi adiabatique dans une équation de type `Convection_Diffusion_Temperature`.. Le flux de température est nul à la frontière.

`Neumann_sortie_libre` : représente une frontière ouverte. Pour la classe `Neumann_sortie_libre` et les classes qui en dérivent, les flux diffusifs sont nuls à la frontière. Par contre pour traiter les flux convectifs, il faut distinguer pour chaque face de bord les deux cas:
- le fluide est sortant,
- le fluide est entrant parce qu'il y a recirculation. Dans ce cas, le schéma de convection impose de connaitre le scalaire convecté ou la vitesse du fluide à l'extérieur. C'est pourquoi la classe porte un `champ_ext`.

`Neumann_sortie_libre` : représente une frontière ouverte pour une équation de transport d'un scalaire. On met dans `champ_ext` la valeur du scalaire calculé à l'extérieur du domaine.

`Sortie_libre_pression_imposee` : La classe représente une frontière ouverte avec condition de pression imposée dans une équation de `Navier_Stokes`.
`le_champ_front` contient la pression et la fonction `flux_impose` renvoie les valeurs de cette pression.
`champ_ext` contient une valeur de la vitesse du fluide à l'extérieur accessible par la méthode `val_ext()`.

5.3 Conditions limites de Symetrie

`Symétrie` : Dans une équation de transport d'un scalaire (température, concentration, K, ε), la condition `Symétrie` met les gradients des grandeurs scalaires à 0.
Dans une équation de `Navier_Stokes`, elle impose la composante normale de la vitesse nulle (condition de glissement).

5.4 Conditions limites de type Echange_impose

4) Les conditions de type `Echange_impose` utilisées pour l'équation d'énergie imposent un échange de chaleur avec l'extérieur du domaine. Cet échange se fait en imposant une température extérieure T_ext et un coefficient d'échange h_imp.
Le terme de flux calculé à partir du couple (h_imp, T_ext) s'écrit:

h_t* (T_ext - T_int) * surface

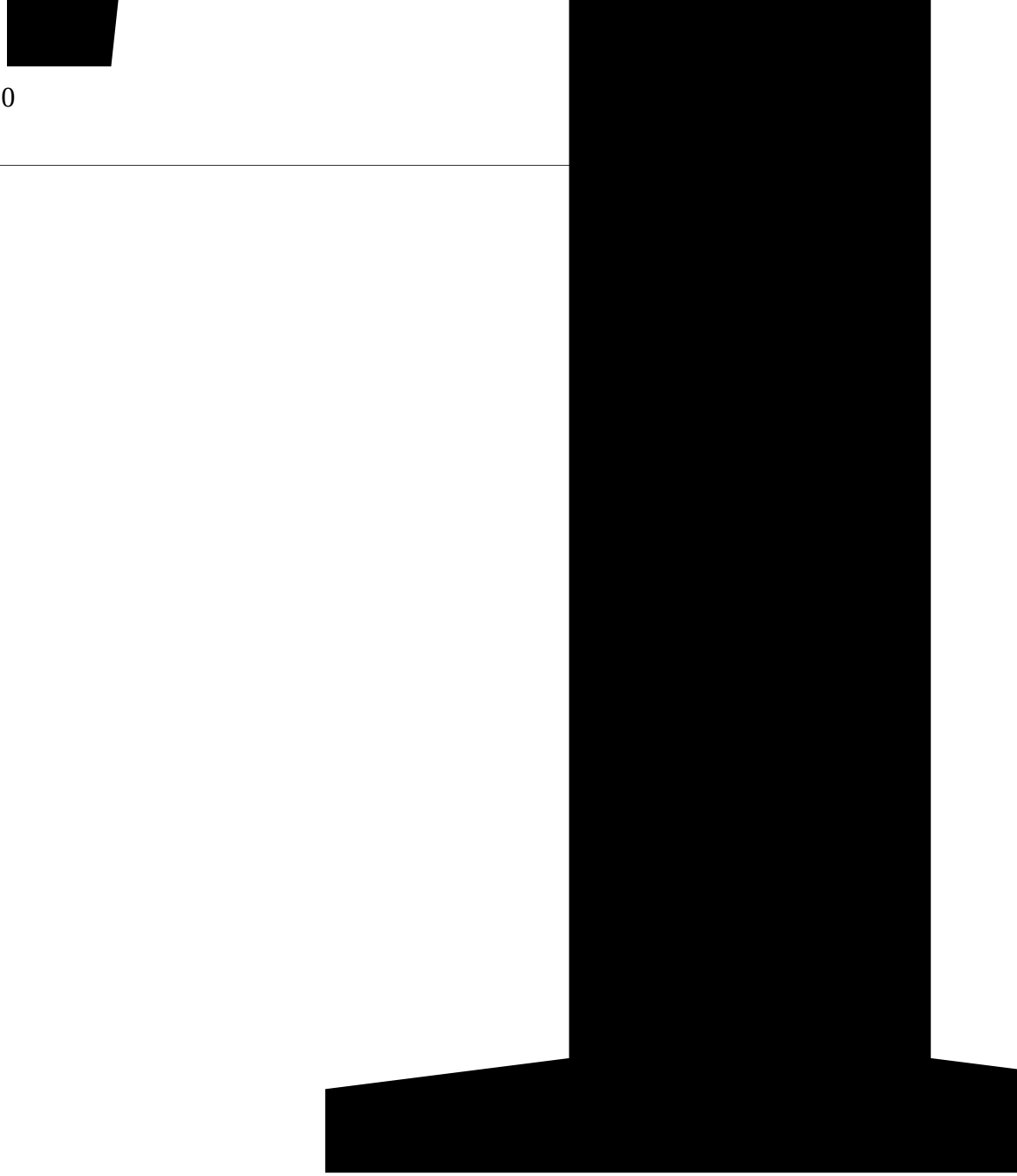
Il figure au second membre de l'équation d'énergie.
On dérive ensuite des classes qui calculent chacune à sa manière le coefficient d'échange h_t.

`Echange_externe_impose` : sert à calculer un coefficient d'échange h_t qui tient compte du coefficient d'échange de la paroi donné par l'utilisateur via la conductivité, et la diffusion dans la demi-maille près de la paroi.
Les champs `h_imp_` et `T_ext_` sont uniformes.

[redacted]

En laminaire, dist = distance du centre de la maille à la paroi.
En turbulent, il faut calculer une distance équivalente d' qui dépend de la distance à la paroi et de la vitesse de frottement U*.

`Echange_global_impose` : contient directement le coefficient d'échange global h_t.
Les champs h_ et `T_ext_` sont uniformes.



6- ThHYd : SOLVEUR PRESSION

Généralités: Les solveurs pression vont résoudre le système:
Matrice * P = second_membre
La pression est un champ scalaire.

Classe générique: `Solveur_Pression`

Classe de base: `Solveur_Pression_base`

Liens avec les autres classes
`Solveur_Pression` est porté par la classe équation de type `Navier_Stokes_std`.

`Solveur_Pression_GCP_ssor` : gradient conjugué avec pré-conditionnement SSOR. Cette classe porte la matrice (de type `Matrice_Morse_Sym`).
Cette classe n'est pas instanciable. On doit dériver des classes pour chaque type de discrétisation.

7- ThHyd : MODELES DE TURBULENCE

On distingue:
- les modèles de turbulence associés à l'équation de Navier_Stokes
- les modèles de turbulence associés à l'équation de transport d'un scalaire.

7.1 Modèles de turbulence (hyd)

Généralités: Les modèles de turbulence utilisés par l'équation de Navier_Stokes contiennent:
- la viscosité turbulente ν_t,
- les lois de paroi turbulente,
- une référence à l'équation hydraulique.

Classe générique: `Mod_turb_hyd`

Classe de base: `Mod_turb_hyd_base`

Modèles de turbulence codés: [PRES1]

Modèle **K-ε** :

`Modele_turbulence_hyd_K_Eps` : représente la modélisation K-ε de la turbulence pour une équation hydraulique.
Cette classe porte une équation de transport de l'énergie cinétique K et du taux de dissipation turbulente ε . La résolution de cette équation permet de calculer la viscosité turbulente.

Modèle **Simulation de grandes échelles** (dit SGE ou LES):

`Mod_turb_hyd_ss_maille` : Ce modèle repose sur le calcul d'une fonction de structure qui permet de calculer la viscosité turbulente.
Cette classe n'est pas instanciable. On doit dériver des classes pour chaque type de discrétisation.

7.2 Modèles de turbulence (scalaire)

Généralités: Les modèles de turbulence portés par une équation de convection diffusion contiennent:
- la diffusivité turbulente α_t,
- une référence aux lois de paroi turbulente,
- une référence à l'équation thermique.

Classe générique: `Modele_turbulence_scal`

Classe de base: `Modele_turbulence_scal_base`

Modèles de turbulence codés

Modèle de **Prandtl** : $\alpha_t = \nu_t$ / Prdtl
La constante Prdtl vaut 0.9.
Les classes possèdent donc une référence au champ de viscosité turbulente.

`Modele_turbulence_scal_Prandtl` : classe instanciable, utilisée pour l'équation de convection diffusion quand l'hydraulique est calculée par le modèle de turbulence K-ε.

`Modele_turb_scal_Prandtl_sous_maille` : c'est le modèle de Prandtl pour l'équation de convection diffusion quand l'hydraulique est calculée par le modèle de turbulence sous-maille.
Cette classe n'est pas instanciable. On doit dériver des classes pour chaque type de discrétisation.

7.3 Lois de paroi

Généralités: Traitement de la turbulence au voisinage de la paroi.

Classe générique: `Turbulence_Paroi`

Classe de base: `Turbulence_Paroi_base`

Liens avec les autres classes
`Mod_turb_hyd` est porté par la classe `Navier_Stokes_Turbulent`.
`Modele_turbulence_scal` est porté par la classe `Convection_Diffusion_Turbulent`.

Les lois de paroi `Turbulence_Paroi` sont portées par le modèle turbulent hydraulique et référencées par le modèle turbulent scalaire.

8- ThHyd : Termes SOURCE

8.1 Boussinesq

Généralités: Les classes de Boussinesq représentent le terme de gravité (qui figure dans l'équation de la dynamique divisée par la masse volumique de référence). On est dans le cas de l'hypothèse de Boussinesq: la masse volumique est supposée constante et égale à sa valeur de référence sauf dans le terme des forces de volume où on prend en compte une petite variation de la masse volumique en fonction d'un ou plusieurs scalaires transportés par l'écoulement.

Classe de base: `Terme_Boussinesq_base`. Cette classe est hors hiérarchie `Objet_U`.

`Terme_Boussinesq_Temperature :` cas particulier du terme de Boussinesq pour la température. Le terme de gravité a pour expression:

```
beta_t * (T-T0)
```

où `beta_t` représente la dilatabilité et `T0` la température de référence.

`Terme_Boussinesq_Concentration :` cas particulier du terme de Boussinesq pour la concentration. Le terme de gravité a pour expression:

`beta_c[0] * (C[0]-C0[0]) +...+beta_c[i] * (C[i]-C0[i])`

où `beta_c[i]` représente la variation de la masse volumique en fonction de `C[i]` .

`Terme_Boussinesq:` terme de Boussinesq pour la concentration et la température.

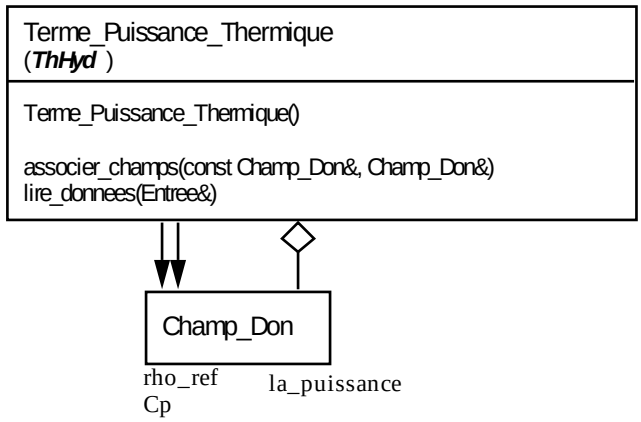
Remarque: Ces 3 classes ne sont pas des classes instanciables. Elles contiennent seulement tout ce qui est nécessaire au calcul mais pas l'implémentation des termes qui dépendra de leur discrétisation.

8.2 Puissance thermique

Généralités: Ces classes représentent le dégagement volumique de puissance thermique.

`Terme_Puissance_Thermique :` contient la puissance (donnée utilisateur) et les références à la masse volumique et la chaleur spécifique.

Remarque: Cette classe n'est pas instanciable. Elle contient seulement tout ce qui est nécessaire au calcul mais pas l'implémentation des termes qui dépendra de leur discrétisation.



TRIO_U : ThSol

Le Module ThSol contient les objets nécessaires au calcul de la thermique dans un milieu solide.

On a défini les familles d'objets suivantes:

1	Problème
2	Equation
3	Milieu
4	Discrétisation

1- ThSol : LES PROBLEMES

Hiérarchie:		
Classe de base:	`Probleme_base`	(Noyau)
Classe générique:	`Probleme`	(Noyau)

`Pb_Conduction` : représente un problème de conduction standard

2- ThSol : LES EQUATIONS

Hiérarchie:		
Classe de base:	`Equation_base`	(Noyau)
Classe générique:	`Equation`	(Noyau)

`Conduction` : Equation d'évolution de la température dans un solide de conductivité k :
L'inconnue est le champ de température.

Liens avec les autres classes
`Conduction` est porté par la classe `Pb_Conduction`

3- ThSol : MILIEU

Hiérarchie:		
Classe de base:	`Milieu_base`	(Noyau)
Classe générique:	`Milieu`	(Noyau)

Généralités: Les classes `Milieu` portent les `Champ_Don` contenant les propriétés du milieu fluide ou solide qu'on modélise.

`Solide` : Cette classe définit un milieu solide. La classe `Milieu_base` porte déjà les champs utiles au calcul de la conduction dans un solide.

Liens avec les autres classes
`Solide` est référencé par la classe équation de type `Conduction`.

4- ThSol : Discrétisation

Hiérarchie:		
Classe de base:	`Discretisation_base`	(Noyau)
Classe générique:	`Discretisation`	(Noyau)

`Discret_Thermique`: Classe de base décrivant la discrétisation spatiale appliquée aux problèmes thermiques.

Liens avec les autres classes
`Discretisation_base` est associé à la classe `Probleme_base` .

Méthodes:
`temperature{---} :` méthode virtuelle pure pour typer et discrétiser les `Champ_Inc` de température

TRIO_U : OPERATEURS

Le Module "Operateurs" contient les objets qui définissent les opérateurs, indépendamment du type de discrétisation spatiale.

On a défini les familles d'objets suivantes:

1	Opérateurs de diffusion
2	Opérateurs de convection
3	Opérateurs de divergence
4	Opérateurs de gradient

Toutes les classes héritent de la classe de base `Operateur_base`.

Hiérarchie:		
Classe de base:	`Operateur_base`	(Noyau)
Classe générique:	`Operateur`	(Noyau)

1- Les opérateurs de diffusion:

Généralités: Représente le terme de diffusion dans une équation. Le choix du terme dépend de la modélisation laminaire ou turbulente de l'écoulement, de la discrétisation et du type du champ de diffusivité.
Les classes des opérateurs de diffusion héritent de la classe `MorEqn` (voir `Operateur` ou `Operateur_base` dans Noyau).

Classe générique: `Operateur_Diff` qui hérite des classes `Operateur` et `DERIV(Operateur_Diff_base)`.

Classe de base: `Operateur_Diff_base` est la classe de base de tous les opérateurs de diffusion.

Classe `Op_Diff_negligeable`: classe instanciable dérivant de `Operateur_Diff_base` pour négliger dans une équation les effets de la diffusion.

Classe `Op_Diff_Turbulent_base`: classe de base pour les opérateurs de diffusion pour un écoulement turbulent.

Classe `Op_Diff_K_Eps_base`: classe de base pour les opérateurs de diffusion pour l'équation de transport des champs K et ε.

Classe `Op_Diff_K_Eps`: classe générique pour les opérateurs de diffusion pour l'équation de transport des champs K et ε.

Liens avec les autres classes
`Operateur_Diff` est porté par les classes des équations:
`Navier_Stokes_std` (ThHyd),
`Convection_Diffusion_std` (ThHyd)
`Conduction` (ThSol)
La méthode `typer()` qui calcule le type de la classe à instancier est portée par la classe générique `Operateur_Diff` .
Ces classes ont une référence au champ de diffusivité (ou viscosité) laminaire (porté par le `Milieu`) et au champ de diffusivité (ou viscosité) turbulente (porté par le modèle de turbulence).

`Op_Diff_K_Eps` est porté par la classe : `Transport_K_Eps`

2- Les opérateurs de convection:

Généralités: Représente le terme de convection dans une équation. Le choix du terme dépend du choix du modèle de convection et de la discrétisation.
Les classes des opérateurs de convection héritent de la classe `MorEqn` (voir `Operateur` ou `Operateur_base` dans Noyau).

Classe générique: `Operateur_Conv` qui hérite des classes `Operateur` et `DERIV(Operateur_Conv_base)`.

Classe de base: `Operateur_Conv_base` est la classe de base de tous les opérateurs de convection.

Classe `Op_Conv_negligeable`: cet opérateur permet de négliger dans une équation les effets de la convection.

Liens avec les autres classes: `Operateur_Conv` est portée par les classes des équations:
`Navier_Stokes_std`,
`Convection_Diffusion_std`
`Transport_K_Eps`
La méthode `typer()` qui calcule le type de la classe à instancier est portée par la classe générique `Operateur_Conv` .

3- Les opérateurs de divergence:

Généralités: Calcul de la divergence d'un champ.

Classe générique: `Operateur_Div` : qui hérite des classes `Operateur` et `DERIV(Operateur_Div_base)`.

Classe de base: `Operateur_Div_base`

Liens avec les autres classes: La divergence est portée par la classe `Navier_Stokes_std` pour le calcul de div(U).

4- Les opérateurs de gradient:

Généralités: Calcul du gradient d'un champ scalaire.

Classe générique: `Operateur_Grad:` qui hérite des classes `Operateur` et `DERIV(Operateur_Grad_base)`

Classe de base: `Operateur_Grad_base`

Liens avec les autres classes: Le gradient est porté par la classe `Navier_Stokes_std` pour le calcul de grad(P).

TRIO_U : GEOMETRIE

Le module "Geometrie" contient les informations que fournit un mailleur.

1. Domaine
2 Zone
3. Frontières
4. Sous_Zone, Frontière utilisateur
5. Elem_geom

1- Geometrie / Domaine

Un **Domaine** est un espace géométrique maillé.

On va résoudre un **Probleme** sur un **Domaine**. On peut vouloir résoudre conjointement des problèmes de type différent sur plusieurs domaines. Géométriquement, les domaines sont alors couplés par des surfaces qu'on appelle **Raccord**.

Si les problèmes sont résolus dans TRIO_U les raccords sont dits **locaux**. Si un problème est résolu dans TRIO_U et l'autre dans un autre code, le raccord est dit **distant**.

Exemple de couplage de domaines: un domaine fluide sur lequel on veut résoudre un problème de type Pb_Hydraulique, est couplé à un domaine solide auquel on associe un problème de type Pb_Conduction.

Un domaine contient une ou plusieurs objets de type **Zone**. Par définition une zone est une partie du domaine à laquelle on va associer un schéma de discrétisation spatial. Une zone possède un et un seul schéma de discrétisation. Si par exemple, on veut résoudre un problème avec à la fois une discrétisation de type structuré et une autre de type dé-structuré, il faudra définir autant de zones que de types de discrétisation.
Remarque: dans TRIO_U Version 1, les domaines sont constitués d'une seule zone.

On appelle **Bord** la surface qui borde le domaine. Pour une équation donnée, l'utilisateur doit associer une condition aux limites à chaque bord.

L'objet de type **Joint** : Au niveau géométrique, la mise en oeuvre d'une méthode de décomposition de domaine pour la parallélisation des calculs consiste à:
- effectuer une partition du maillage (avec un découpeur),
- distribuer un morceau à chaque processeur.

Chaque processeur possède alors un domaine, une zone avec des bords et des joints.

Les **faces internes** définissent des surfaces internes au maillage.

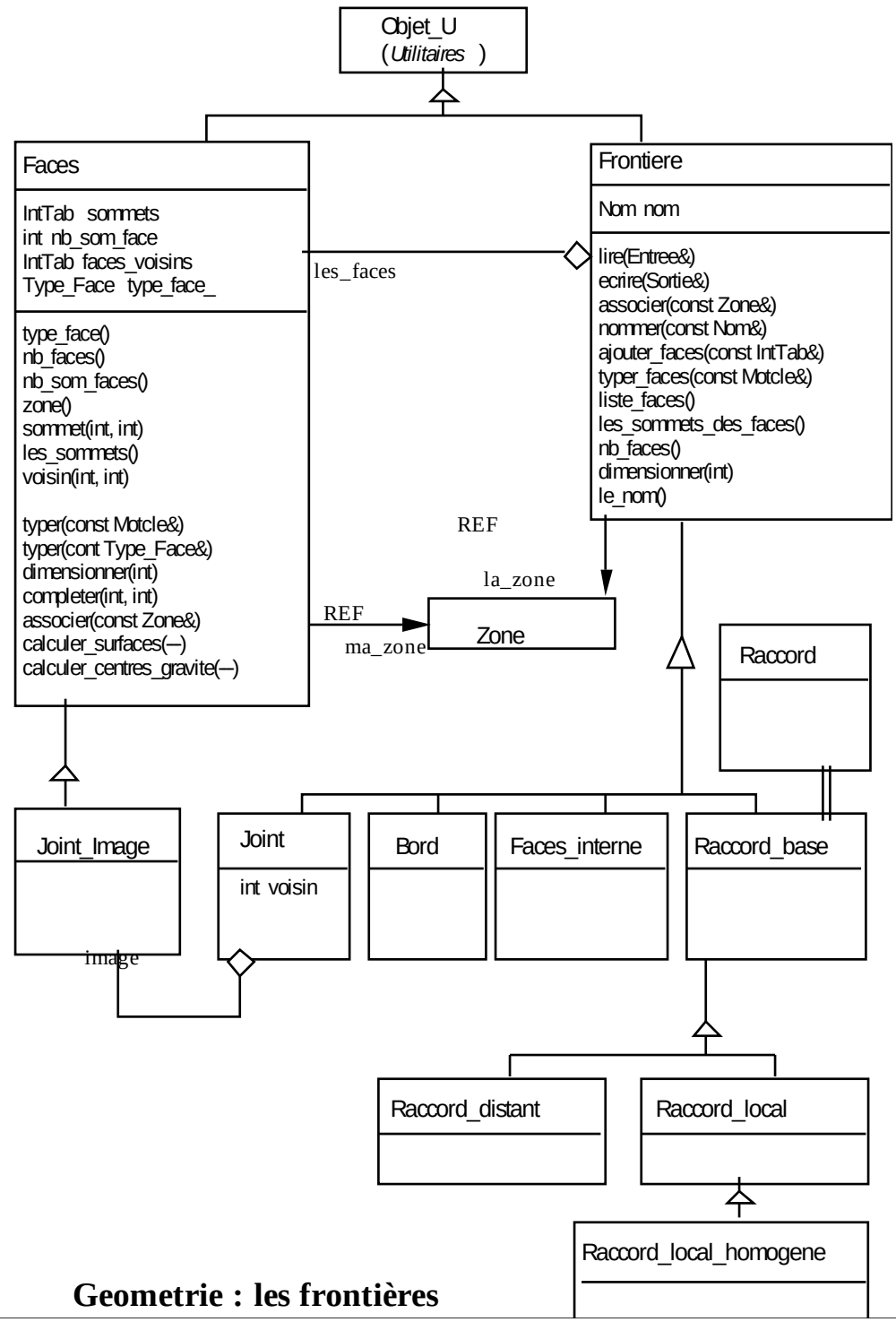
Une **Sous_Zone** est un ensemble de mailles dans une zone.

Bord_utilisateur: Regroupement de bords élémentaires pour faciliter l'entrée des données de Trio_U.

2- Geometrie / Zone

3- Geometrie / Frontiere

La classe **`Frontiere`** est la classe de base pour représenter un ensemble de **faces** dans un maillage, soit :
- les bords : limite du maillage avec l'extérieur,
- les joints : surface de découpage du maillage pour du calcul parallèle avec méthode de décomposition de domaine
- les raccords : surface de séparation de deux problèmes
- les faces internes : faces à l'intérieur d'un maillage

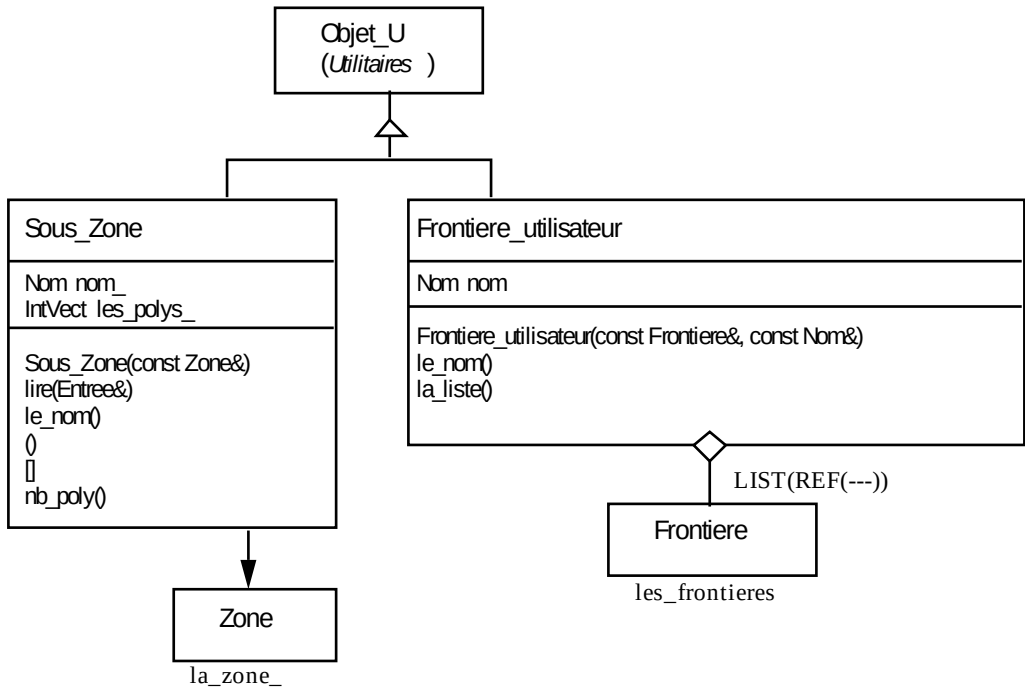


Geometrie : les frontières

4- Geometrie / Sous_Zone, Frontiere_utilisateur

Une **Sous_Zone** est un ensemble de mailles dans une zone.

Bord_utilisateur: Regroupement de bords élémentaires pour faciliter l'entrée des données de Trio_U.



Geometrie : sous zone , frontiere utilisateur

5- Geometrie / Elem_geom

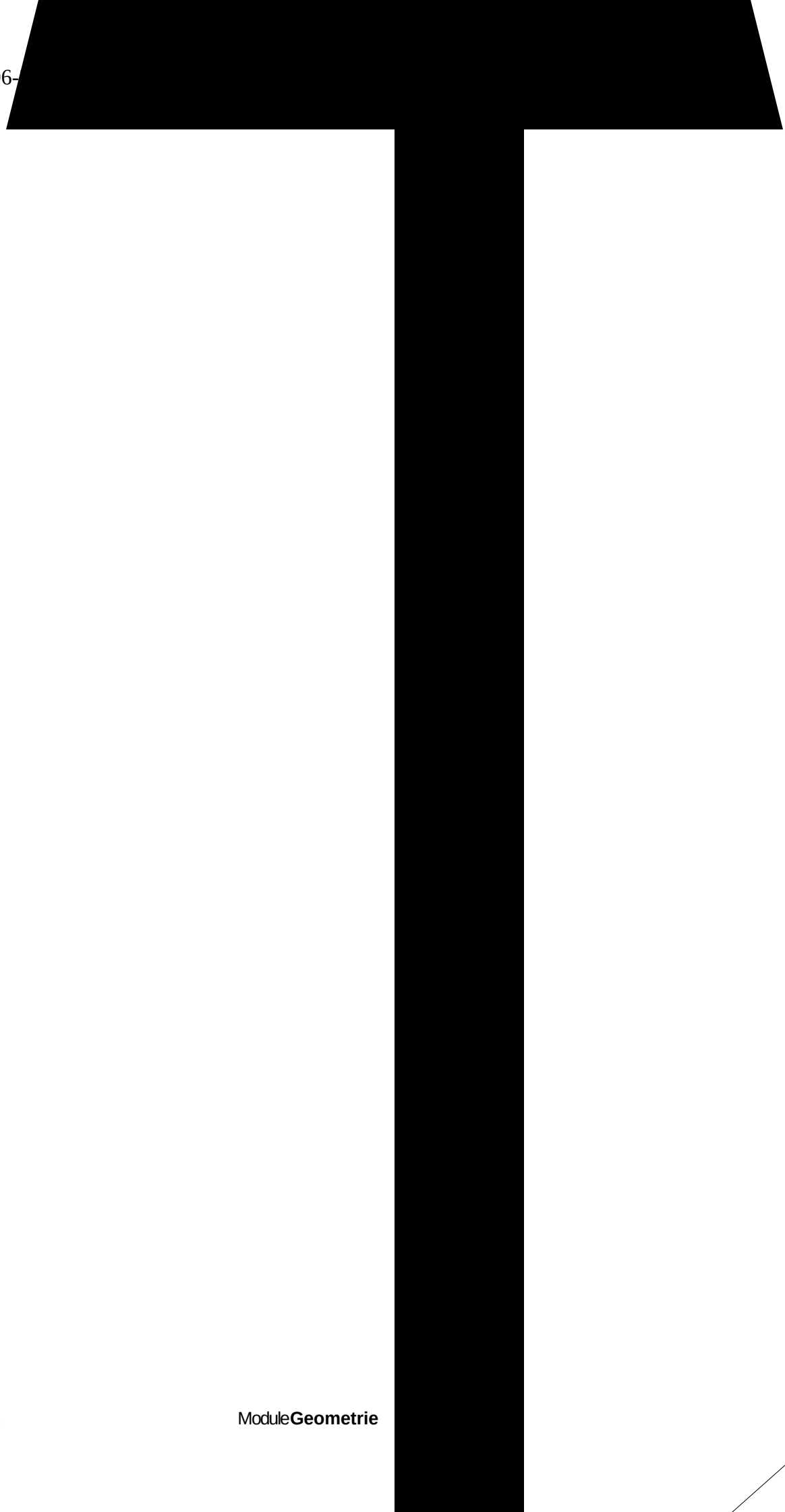
La classe `Elem_geom` définit le type d'éléments qui constituent une `Zone`.

Chaque classe instanciable doit implémenter les méthodes:

`face_sommet(int i, int j)` :	ième sommet de la face j
`nb_som()` :	nombre de sommets de l'élément
`nb_face()` :	nombre de faces de l'élément
`nb_som_face()` :	nombre de sommets d'une face
`est structure()` :	=1 si l'élément est de type structuré
`type_face()` :	renvoie le type des faces
`nom_lml()` :	nom du type de l'élément pour le fichier de sortie lml
`contient(const DoubleVect& pos, int elem)` :	le point de coordonnées `pos` est-il situé dans l'élément `elem`?
`calculer_volumes(DoubleVect& vol)` :	remplissage du tableau `vol` , du volume des éléments de la zone
`reordonner()` :	définit la renumérotation des sommets d'un élément.

Rectangle : élément utilisé pour définir un maillage structuré en 2D.
Convention de numérotation des sommets:

Héxaèdre : élément utilisé pour définir un maillage structuré en 3D.
Convention de numérotation des sommets:



TRIO_U : Schemas_Temps

Le Module **Schemas_Temps** contient les classes des schémas de discrétisation en temps.

Une seule classe existe dans TRIO_Version1 :

1 Schema_Euler_explicite

1- Schemas_Temps/ Schema_Euler_explicite

Hiérarchie:		
Classe de base:	`Schema_temps_base`	(Noyau)
Classe générique:	`Schema_temps`	(Noyau)

`Schema_Euler_explicite` : définit le schéma d'Euler explicite
U(n+1) = U(n) + dt * (dU/dt)(n)

Les méthodes:

`nb_valeurs_temporelles()`: renvoie le nombre de pas de temps pour lesquels on stocke les `Champ_Inc` .
Renvoie le nombre 2, soit : U(n) et U(n+1)

`faire_un_pas_de_temps(Equation_base& eqn)` : Exécution d'un pas de temps pour l'équation `eqn`.

Codage de la méthode `faire_un_pas_de_temps` :

```
int Schema_Euler_explicite :: faire_un_pas_de_temps(Equation_base& eqn)
{
      DoubleTab& futur =
                        eqn.derivee_en_temps_inco(eqn.inconnue().futur());
      double acc_max_abs = futur.max_abs();
      stationnaire_atteint_ *= (acc_max_abs  < seuil_statio_);

      DoubleTab& present = eqn.inconnue();
      futur *= dt_;
      futur += present;
      eqn.mettre_a_jour()
      return 1;
}
```

TRIO_U : VF

Le Module **VF** contient les objets communs aux schémas de discrétisation de type *volumes finis*.

On a défini les familles d'objets suivantes:

1	Zones
2	Eléments

1- VF / Zones

Hiérarchie:		
Classe de base:	`Zone_dis_base`	(Noyau)
Classe générique:	`Zone_dis`	(Noyau)

La classe **Zone_VF** est la classe de base des `Zone_dis` pour toutes les discrétisations volumes finis (VDF, VEF, ...).

Elle contient les tableaux relatifs à l'ensemble des faces de la zone:

`face_voisins_` : les deux éléments voisins des faces (*)
`face_sommets_` : les sommets des faces (*)

et les tableaux relatifs aux éléments:

`elem_faces_` : les faces des éléments
`volumes_` : volumes des éléments (*)

Les tableaux (*) sont remplis par la méthode `Zone_VF::discretiser()`.

La classe dérivée applique ses conventions de rangement pour remplir le tableau `lem_faces_`.
Elle peut aussi décider de permuter les valeurs du tableau `face_voisins_` pour satisfaire à ses conventions.

La classe `Zone_VF` contient également un Vecteur de `Front_VF` qui représentent les bords et les raccords discrétisés, ainsi que les `Zone_joint_VF` qui représentent les joints discrétisés.

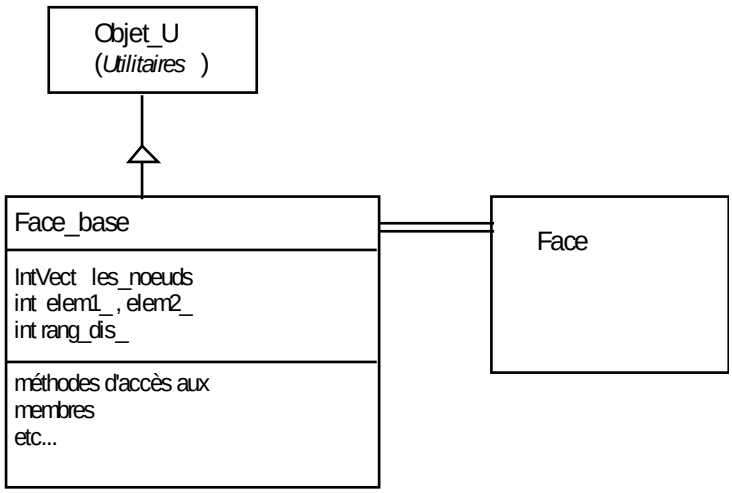
2- VF / Eléments

La classe `Face_base` décrit une face pour une discrétisation de type Volumes_finis (VDF, VEF, ...). Cette classe sert temporairement pour la construction des zones discrétisées.

Les membres:

`elem1_` et `elem2_` : numéro des éléments voisins de la face

`les_noeuds` : numéros des sommets de la face.



VF / Elements : Face

TRIO_U : VDF

Le Module **VDF** contient les classes pour le schéma de discrétisation Volumes Différences finis.

La discrétisation Volumes Différences Finies (VDF) de TRIO_U est le schéma qui a fait ses preuves dans TRIO VF. Référence [PRES1].

Quelques améliorations ont été apportées lors de la ré-écriture:
- on est passé de la notation i-j-k à une notation de type "éléments finis", plus souple et plus facile à mettre en oeuvre.
- on peut calculer du vrai 2D.

Le maillage est structuré : mailles rectangle en 2D, parallélipipède en 3D pour des coordonnées cartésiennes. Le maillage en coordonnées cylindriques n'est pas décrit dans cette version du document.
Au maillage, correspond un maillage décalé, centré autour des faces. Dans la suite, on appellera ces volumes "**volumes entrelacés**" et leurs faces des "**facettes**".

Les équations de transport de scalaire sont discrétisées sur les éléments, tandis que Navier_Stokes est discrétisé sur le maillage entrelacé.

Le champ de vitesse est de type `Champ_Face` : on ne calcule que la composante normale aux faces.

Les champs scalaires (pression, température, concentration, K, Epsilon) sont de type `Champ_P0_VDF` (constants dans la maille).

Le Module VDF contient les familles de classe suivantes:

1	Zone_VDF
2	Zone_Cl_VDF
3	Eléments
4	Discrétisation
5	Opérateurs
6	Sources
7	Solveur Masse
8	Solveur Pression
9	Modèles de turbulence
10	Conditions limites
11	Champ
12	Statistiques temps

1- VDF / ZONES : Zone_VDF

Hiérarchie:		
Classe de base:	`Zone_dis_base`	(Noyau)
	`Zone_VF`	(VF / Zones)
Classe générique:	`Zone_dis`	(Noyau)

La classe **Zone_VDF** contient les informations géométriques que demande la méthode des Volumes différences finis.
Cette discrétisation correspond à un maillage structuré (mailles rectangle en 2D, parallélipipède en 3D pour des coordonnées cartésiennes).

La `Zone_VDF` hérite de la convention de rangement des faces établies par `Zone_VF`. Les faces sont numérotées de la façon suivante:
- les faces qui sont sur une **Zone_joint** apparaissent en premier (dans l'ordre de la liste `mes_faces_joint` de la `Zone`).
- les faces qui sont sur une **Zone_bord** apparaissent ensuite (dans l'ordre de la liste `mes_faces_bord` de la `Zone`)
- les faces qui sont sur un **Raccord** apparaissent ensuite (dans l'ordre de la liste `mes_faces_raccord` de la `Zone`). Les raccords étant, dans le calcul, assimilés à des bords avec une condition aux limites adéquate, on ne les distingue plus dans Zone_VDF: on parle donc de `Frontiere_dis` qui sont associées à des conditions limites.
- les faces **internes** apparaissent enfin.

La Zone_VDF réordonne les faces de façon à les ranger suivant leur orientation.
A chaque face on fait correspondre un entier qui indique son **orientation**. On suppose qu'à l'intérieur de chaque famille de faces (joint, bord, interne) on trouve:

- le bloc des faces d'équation x=cte (face d'orientation 0),
- le bloc des faces d'équation y=cte (face d'orientation 1),
- le bloc des faces d'équation z=cte (face d'orientation 2),

On n'a pas besoin d'une numérotation particulière des éléments.

Figure: Numérotation des faces

La Zone_VDF ordonne le rangement des faces à l'intérieur d'un élément. Le tableau **elem_faces_** (de Zone_VF) qui contient les numéros des faces d'un élément est rangé de la façon suivante:

- en 2D :

- en3D:

Dans le tableau `face_voisins_` (dans `Zone_VF`) qui contient les numéros des éléments voisins d'une face, on range les 2 voisins

```
elem0 = face_voisins_(face,0)
```

et `elem1 = face_voisins_(face,1)` de la façon suivante:

face d'orientation 0 : `xp_(elem0, 0) < xp_(elem1,0)`
face d'orientation 1 : `xp_(elem0, 1) < xp_(elem1,1)`
face d'orientation 2 : `xp_(elem0, 2) < xp_(elem1,2)`

`xp_(elem, i)` étant la ième coordonnée du centre de gravité de l'élément `elem`.

On a introduit la notion d'**arête** pour le calcul des flux diffusifs et convectifs dans la conservation de la quantité de mouvement.

Figure: les arêtes

Le tableau Qdm contient la connectivité arête/faces, à savoir les faces entourant une arête. Dans ce tableau les arêtes apparaissent dans l'ordre suivant:

- bloc des arêtes joint,
- bloc des arêtes coin,
- bloc des arêtes bord,
- bloc des arêtes mixte,
- bloc des arêtes internes,

A l'intérieur de chaque bloc les arêtes apparaisent dans l'ordre suivant: arêtes XY, arêtes XZ et arêtes YZ.

Les membres:

`axi_` : = 1 en coordonnées cylindriques, =0 sinon

Tableaux associés aux **faces**:

`face_surfaces_`: surface de chaque face
`volumes_entrelaces_` : volume entrelacé associé à chaque face
`xv_` : coordonnées du centre de gravité des faces.
`porosite_face_` : porosité surfacique des faces.
`orientation_` : orientation des faces.

Tableaux associés aux **éléments**:

`xp_` : coordonnées du centre de gravité des éléments.
`porosite_elem_` : porosité volumique des éléments.

Tableaux associés aux **arêtes**:

`Qdm_` : numéro des faces entourant une arête

`h_x_` : pas du maillage en X. la plus petite distance entre 2 centres de faces dans la direction X=cte
`h_y_` , `h_z_` : idem dans les directions y et z.

Les méthodes qui calculent:

Ces méthodes supposent l'ordre de rangement des faces:

```
inline int premiere_arete_coin() const;
inline int premiere_arete_bord() const;
inline int premiere_arete_mixte() const;
inline int premiere_arete_interne() const;
```

Méthodes calculant des distances:

Distance entre les centres de gravité de 2 éléments n1 et n2 dans la direction k:

```
inline double dist_elem(int n1, int n2, int k) const;
```

Dimensiond'unélémentn1 dansla directionk:

```
inline double dim_elem(int n1, int k) const;
```

Distance entre les faces f1 et f2 dans la direction k:

```
inline double dist_face(int f1, int f2, int k) const;
                                        // coordonnées cartésiennes
inline double distance_face(int f1, int f2, int k) const;
                                        // cartésiennes ou cylindriques
```

Méthodes appelées par le schéma de convection Quick:

```
inline int amont_amont(int num_face, int i) const;
```

Renvoie elem1 ou elem2 suivant la valeur de i (i vaut 0 ou 1).

`inline int` **`face_amont_princ`**`(int num_face, int i) const;`
Renvoie la face f1 ou f2 suivant la valeur de i (i vaut 0 ou 1).

```
inline int face_amont_conj(int num_face, int k, int i) const;
```
Renvoie la face f3c (resp f4c) quand num_face vaut f3 (resp f4).
i vaut 0 ou 1, k est l'orientation de la face f1.

Distancenormalepo
```
inline d
```

Distancenormalepour une face de bord (coordonnées cartésiennes)
```
inline double dist_norm_bord(int num_face) const;
```

Distance normale pour une face quelconque (coordonnées cartésiennes ou cylindriques): vaut **`dist_norm`**`(int num_face)` pour une face interne, ou **`dist_norm_bord`**`(int num_face)` pour une face de bord.

```
inline double distance_normale(int num_face) const;
```

Méthode utile au calcul de la fonction de structure pour le modèle sous-maille.

```
inline double delta_C(int elem) const;
```

Liens avec les autres classes

La `Zone_dis` est portée par le `Domaine_dis` , lui même porté par le `Probleme_base` .

Pratiquement toutes les classes VDF ont une référence à la Zone_VDF. Certaines classes, pour avoir un accès rapide aux tableaux de Zone_VDF, partagent avec la Zone_VDF la référence à ces tableaux.

Exemple:

```
class Evaluateur_VDF
{ ...
REF(Zone_VDF)  la_zone ;
DoubleVect  porosite;
... }
```

La référence est remplie de la façon suivante:

```
Evaluateur_VDF::associer_zones(const Zone_VDF& zone_VDF,...)
{...
porosite.ref(zone_VDF.porosite_face());
}
```

2- VDF / ZONES : Zone_Cl_VDF

Hiérarchie:		
Classe de base:	`Zone_Cl_dis_base`	(Noyau)
Classe générique:	`Zone_Cl_dis`	(Noyau)

Membres:

`type_arete_bord_`: type des arêtes séparant 2 faces frontières
type = 0 : l'arête sépare 2 faces de paroi
type = 1 : l'arête sépare 2 faces "fluide"
type = 2 : l'arête sépare une face de paroi et une face fluide
type = 3 : l'arête sépare 2 faces de symétrie

Liens avec les autres classes

La `Zone_Cl_dis` est portée par l' `Equation_base` .
Pratiquement toutes les classes VDF ont une référence à la `Zone_Cl_VDF`.

3- VDF / Elements

Ces classes permettent la discrétisation de la géométrie.

`Faces_VDF` : Décrit un ensemble de faces pour la discrétisation VDF.
`Aretes` : Décrit un ensemble d'arêtes.

4- VDF / Discrétisation

Hiérarchie:		
Classe de base:	`Discretisation_base`	(Noyau)
Classe générique:	`Discretisation`	(Noyau)

`VDF_discretisation` : classe instanciable pour la discrétisation VDF. Les méthodes réalisent la discrétisation de la `Zone_Cl_dis` et des `Champ_Inc` portés par les équations.

5- VDF / Operateurs

Hiérarchie:		
Classe de base:	`Operateur_base`	(Noyau)
Classe générique:	`Operateur`	(Noyau)
	`Operateur_Diff`	(Operateur)
	`Op_Diff_K_Eps`	(ThHyd/Turbulent)
	`Operateur_Conv`	(Operateur)
	`Operateur_Grad`	(Operateur)
	`Operateur_Div`	(Operateur)

Les opérateurs VDF sont le fruit de l'association de 3 hiérarchies. Sur un exemple montrons d'abord l'intérêt de ce découpage.

Exemple: opérateur de convection, de type schéma Amont [PRES1], VDF, équation de transport d'un scalaire S, méthode `ajouter(...)` :

Voici un algorithme qui calculerait le terme de convection et l'ajouterait au second membre:

```
1- calcul des faces internes:

 boucle sur les faces internes:
       pour chaque face:
       calcul du flux:Flux = S_amont*U(face)*Surf(face)*porosite(face)

       distribution du flux:
       sm(i) -= Flux;      sm(j) += Flux;
 fin de la boucle sur les faces internes

2- calcul des faces bords: il faut calculer les bords qui portent les
conditions limites suivantes:
       bords avec Cl Dirichlet _entree_fluide,
       bords avec CL Neumann_sortie_libre
```

On s'aperçoit qu'il y a 2 types de tâches dans cet algorithme:
- un calcul de flux,
- une gestion des boucles, ainsi que la distribution des flux au second membre.

On va retrouver cette répartition dans tous les opérateurs VDF, VEF, etc... d'où l'idée de définir 3 hiérarchies:
- des *évaluateurs* de flux chargés du calcul de flux,
- des *itérateurs* responsables des boucles, réutilisables dans tous les opérateurs,
- des *opérateurs*, portés par les équations, contenant un itérateur et un évaluateur.

Pour résumer:
- Les opérateurs, portés et instanciés par les équations, vont porter un itérateur.

- Il est nécessaire de définir deux types d'itérateurs: un premier pour gérer les calculs d'équations scalaires (boucles sur les faces et répartition des flux sur les éléments voisins), un second pour les équations de quantité de mouvement pour gérer les boucles sur les arêtes et répartition des flux sur les faces.

- Les itérateurs seront définis de façon générique en utilisant les templates (ou patron). Remarque, pour des raisons liés aux compilateurs des ordinateurs différents de HP, on a remplacé dans Version 1 les templates par des MACRO. Quand les compilateurs accepteront tous des template, on reviendra aux itérateurs template qui sont beaucoup plus lisibles que les macros.

- Les itérateurs vont porter un évaluateur de flux.

- Les évaluateurs de flux vont gérer le type des conditions aux limites pour lesquelles ils doivent calculer un flux.

- Les opérateurs ne portent qu'un constructeur qui doit instancier l'itérateur choisi et le type d'évaluateur de flux porté par l'itérateur.

Intérêt de la hiérarchie:
L'écriture d'un opérateur particulier devient sûre et rapide. Les boucles sur les faces, les facettes ou les éléments existent déjà dans la hiérarchie des itérateurs. Il suffit de coder l'évaluateur de flux ad-hoc, c'est à dire juste le calcul du flux .

5.1 Les itérateurs VDF :

`Iterateur_VDF_Elem` : Le calcul des flux sur les faces d'un élément revient à gérer les boucles sur les faces et à répartir le flux sur les éléments voisins de la face.

Méthodes

`ajouter_interne(---)` : boucle sur les faces internes
Pour chaque face,
appel de la fonction `flux_faces_interne()` de l'évaluateur
répartition du flux sur les éléments voisins de la face.

`ajouter_bords(---)` : boucle sur les frontières. Suivant le type de condition limite `Cl` que porte la frontière:
Pour chaque face,
appel de la fonction `flux_faces(---, cl, ---)` de l'évaluateur
envoi du flux sur l'élément voisin de la face.

`Iterateur_VDF_Face` : Le calcul des flux sur les faces d'un volume entrelacé contient :
- la gestion des boucles sur les arêtes pour les facettes (1) à cheval sur deux faces
- et la gestion des boucles sur les facettes (2) à l'intérieur d'un élément.
Le flux est réparti sur les faces voisines

Méthodes

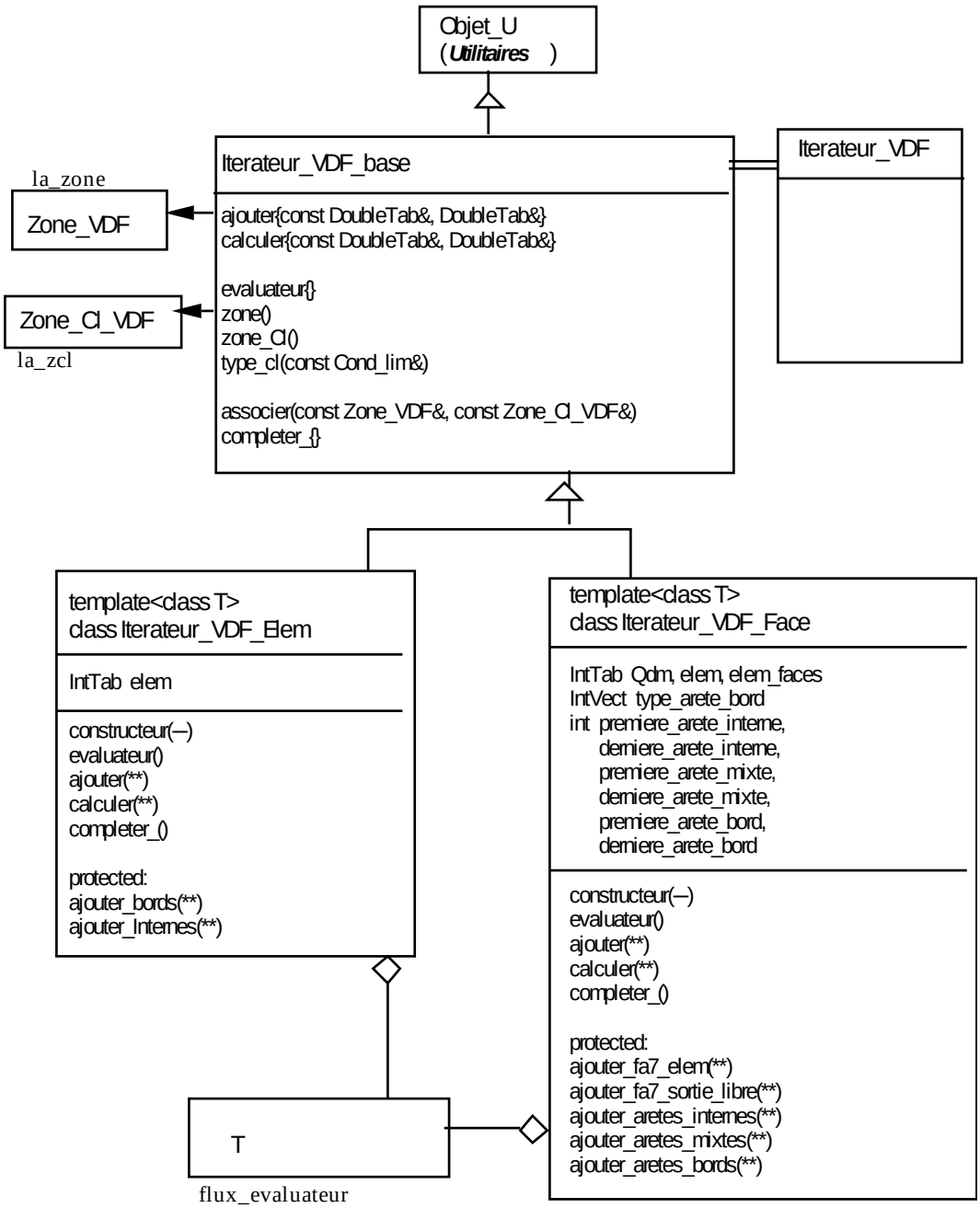
`ajouter_aretes_internes(---)` : boucle sur les arêtes internes
Pour chaque arête,
appel de la fonction `flux_arete_interne()` de l'évaluateur pour les 2 facettes fy (resp. fx).
répartition du flux sur les faces voisines f2 et f3 (resp. f0 et f1).

`ajouter_fa7_elem(---)` : boucle sur les facettes(2) à l'intérieur d'un élément.
Pour chaque facette, appel de la fonction `flux_fa7_elem()` de l'évaluateur, puis répartition sur les faces f1 et f2.

`ajouter_aretes_bord(---)` : boucle sur les arêtes bord. Suivant le type de condition limite appliquée aux deux faces de bord entourant l'arête, appel des fonctions:
`flux_arete_paroi()`
`flux_arete_fluide()`
`flux_arete_paroi_fluide()`
`flux_arete_symetrie()`
de l'évaluateur, et envoi du flux sur la face f1 .

`ajouter_aretes_mixtes(---)` : boucle sur les arêtes mixtes. Appel des fonctions:
`flux_arete_mixte()` de l'évaluateur, et envoi du flux sur les faces entourant l'arête.

`ajouter_fa7_sortie_libre(---)` : boucle sur les frontières pour trouver les frontières qui portent une conditon limite de type sortie_libre.
calcul du flux à ajouter sur la face qui porte cette condition limite. La fonction `flux_fa7_sortie_libre()` de l'évaluateur est appelée.



Itérateur VDF

arguments des opérations (**) : const DoubleTab&, DoubleTab&

5.2 Classes particulières des opérateurs VDF pour la diffusion :

1.) Opérateurs associés à une équation de transport

Le champ diffusé est **scalaire** de type `Champ_P0_VDF` (calculé au centre des éléments). L'itérateur associé à l'opérateur est du type **`Iterateur_VDF_Elem`**.

Le type des opérateurs de diffusion dépend:
- du type de l'écoulement: laminaire ou turbulent
- du type du champ de diffusivité : uniforme (isotrope, constant en espace)
  ou isotrope, variable en espace
- du nombre de composantes du champ de diffusivité. (l'équation `Convection_Diffusion_Concentration` peut calculer le transport de plusieurs constituants, le champ de diffusivité a alors plusieurs composantes.)

1.1) l'écoulement est *laminaire*

Classe instanciable	Evaluateur correspondant	Champ de diffusivité	nb Inc
`Op_Diff_VDF_Elem`	`Eval_Diff_VDF_const_Elem`	uniforme	1
`Op_Diff_VDF_var_Elem`	`Eval_Diff_VDF_var_Elem`	variable	1
`Op_Diff_VDF_Multi_inco_Elem`	`Eval_Diff_VDF_Multi_inco_const_Elem`	uniforme	>1
`Op_Diff_VDF_Multi_inco_var_Elem`	`Eval_Diff_VDF_Multi_inco_var_Elem`	variable	>1

1.2) l'écoulement est *turbulent*

Classe instanciable	Evaluateur correspondant	Champ de diffusivité	nb Inc
`Op_Dift_VDF_Elem`	`Eval_Dift_VDF_const_Elem`	uniforme	1
`Op_Dift_VDF_var_Elem`	`Eval_Dift_VDF_var_Elem`	variable	1
`Op_Dift_VDF_Multi_inco_Elem`	`Eval_Dift_VDF_Multi_inco_const_Elem`	uniforme	>1
`Op_Dift_VDF_Multi_inco_var_Elem`	`Eval_Dift_VDF_Multi_inco_var_Elem`	variable	>1

1.3) Opérateur de diffusion spécialisé pour l'équation de **transport** des champs **K** et **ε**

`Op_Diff_K_Eps_VDF_Elem:`
L'évaluateur associé a le type `Eval_Diff_K_Eps_VDF_Elem`

2.) Opérateurs associés à une équation de Navier_Stokes

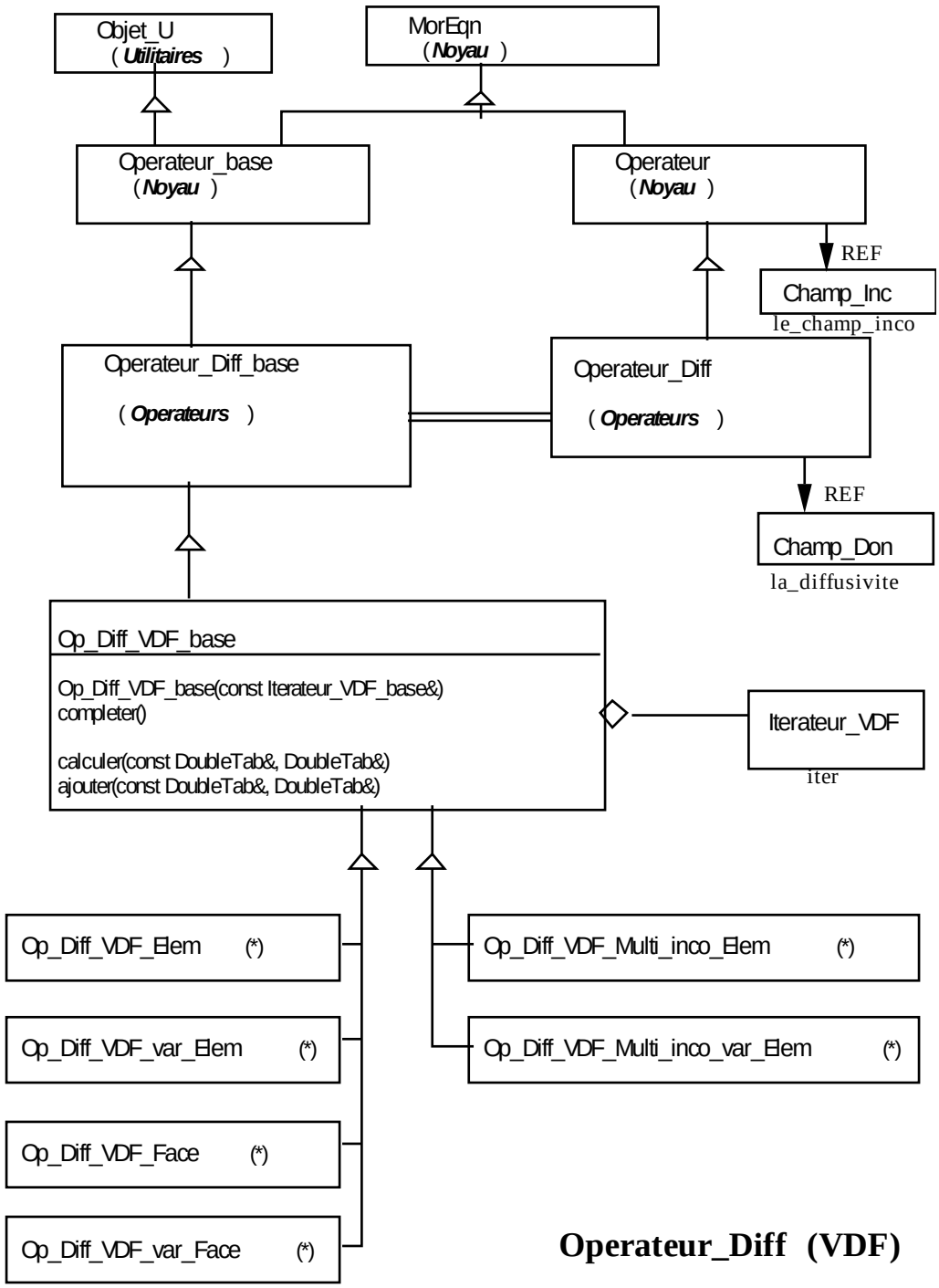
Le champ diffusé est **vectoriel**, calculé au centre des faces (de type Champ_Face). L'itérateur associé à l'opérateur est du type **`Iterateur_VDF_Face`**.

2.1) l'écoulement est *laminaire*

Classe instanciable	Evaluateur correspondant	Champ de viscosité
`Op_Diff_VDF_Face`	`Eval_Diff_VDF_const_Face`	uniforme
`Op_Diff_VDF_var_Face`	`Eval_Diff_VDF_var_Face`	variable

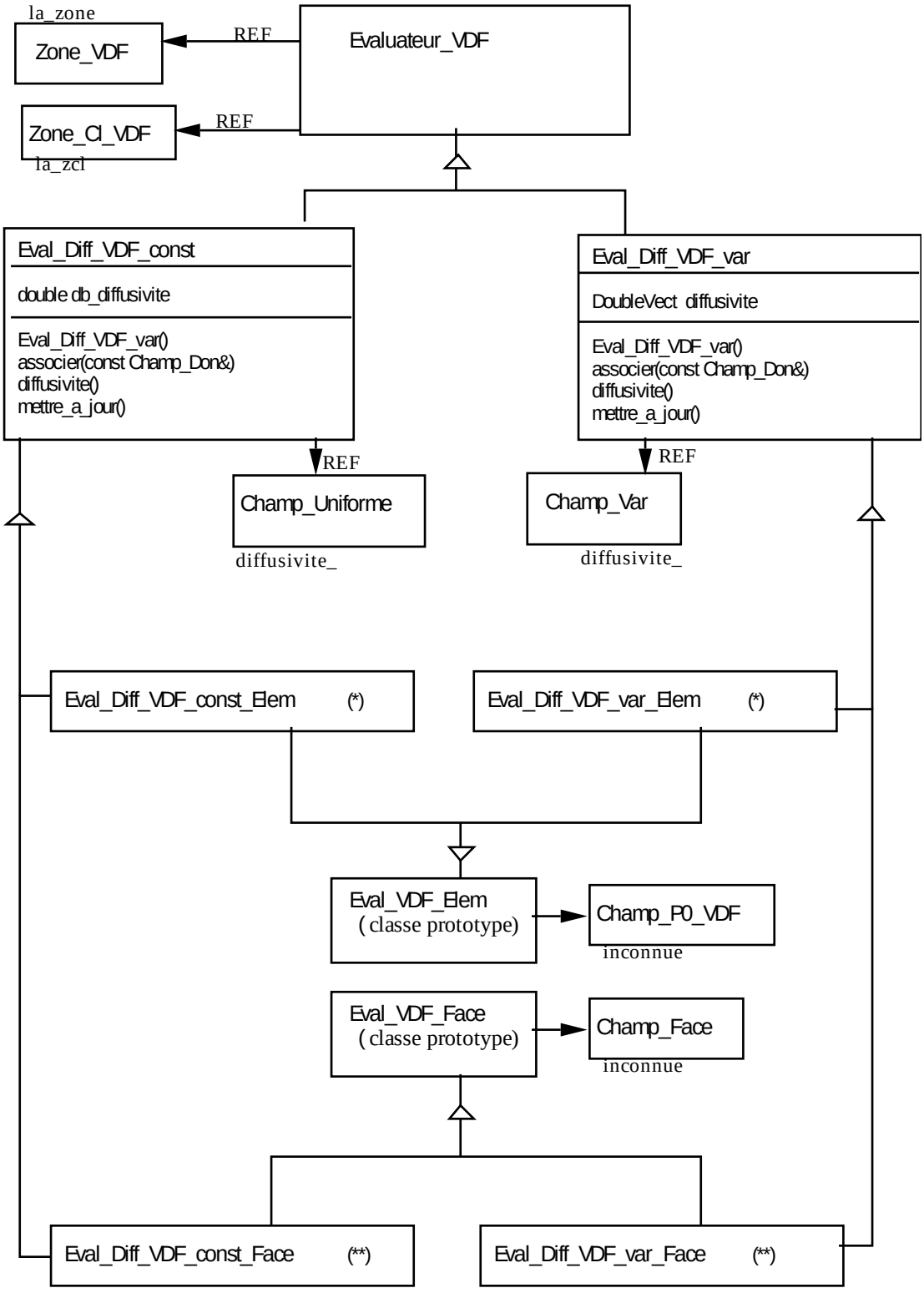
2.2) l'écoulement est *turbulent*

Classe instanciable	Evaluateur correspondant	Champ de viscosité
`Op_Dift_VDF_Face`	`Eval_Dift_VDF_const_Face`	uniforme
`Op_Dift_VDF_var_Face`	`Eval_Dift_VDF_var_Face`	variable



Operateur_Diff (VDF)

(*) méthodes implémentées par les classes instanciables d' **Operateur_Diff** :
constructeur()
associer(const Zone_dis&, const Zone_Cl_dis&, const Champ_Inc&)
associer_diffusivite(const Champ_Don&)
diffusivite()
calculer_dt_stab()



Evaluateurs VDF Diffusion (champ à une inconnue)

5.3 Classes particulières des opérateurs VDF pour la convection:

Les schémas de convection Amont et Quick sont décrits dans le document [PRES1]

1.) Opérateurs associés à une équation de transport :

Le champ convecté est **scalaire**, de type `Champ_P0_VDF` (calculé au centre des éléments). L'itérateur associé à l'opérateur est du type `Iterateur_VDF_Elem`.

`Op_Conv_Amont_VDF_Elem:` schéma de convection amont

`Op_Conv_Quick_VDF_Elem:` schéma Quick

2) Opérateurs associés à une équation de quantité de mouvement :

Le champ convecté est **vectoriel**, calculé au centre des faces (de type Champ_Face). L'itérateur associé à l'opérateur est du type `Iterateur_VDF_face`.

`Op_Conv_Amont_VDF_Face:` schéma de convection amont

`Op_Conv_Quick_VDF_Face:` schéma Quick

5.4 Classes particulières des opérateurs VDF pour la divergence:

`Op_Div_VDF_Elem`: Divergence d'un champ localisé aux faces dans une discrétisation VDF. Cette classe porte un itérateur de type `Iterateur_VDF_Elem`.
Cette classe va servir à calculer div(U).
Remarque : Le champ référencé par l'évaluateur devrait être l'inconnue de l'équation Navier_Stokes, soit le champ de vitesse. Or ce champ est de type `Champ_Face`. C'est pourquoi on ne remplit pas la référence `inconnue` dans `Eval_VDF_Elem`.

5.5 Classes particulières des opérateurs VDF pour le gradient:

`Op_Grad_VDF_Face`: Gradient d'un champ scalaire P0 localisé au centre des mailles dans une discrétisation VDF. Cette classe porte un itérateur de type `Iterateur_VDF_Face`.
Cette classe va servir à calculer Grad(P).
Remarque : Le champ référencé par l'évaluateur devrait être l'inconnue de l'équation Navier_Stokes, soit le champ de vitesse. Or on a besoin de la pression qui est de type `Champ_P0_VDF`. C'est pourquoi on ne remplit pas la référence `inconnue` dans `Eval_VDF_Face`.

6- VDF / Sources

Hiérarchie:		
Classe de base:	`Source_base`	(Noyau)
Classe générique:	`Source`	(Noyau)

Pour la discrétisation VDF, on a développé:
- des termes sources dans une hiérarchie d'itérateurs et d'évaluateurs,
- et des termes source directs.

6.1 Termes sources traités dans la hiérarchie des itérateurs et des évaluateurs

La hiérarchie est comparable à celle des opérateurs VDF. Ces termes sources portent un objet "`Iterateur_Source_VDF`" qui porte un objet "`Evaluateur_Source_VDF`".

Les **`Iterateurs_Source_VDF`** :

`Iterateur_Source_VDF_Elem` : gestion des boucles sur les faces et répartition des flux sur les éléments voisins

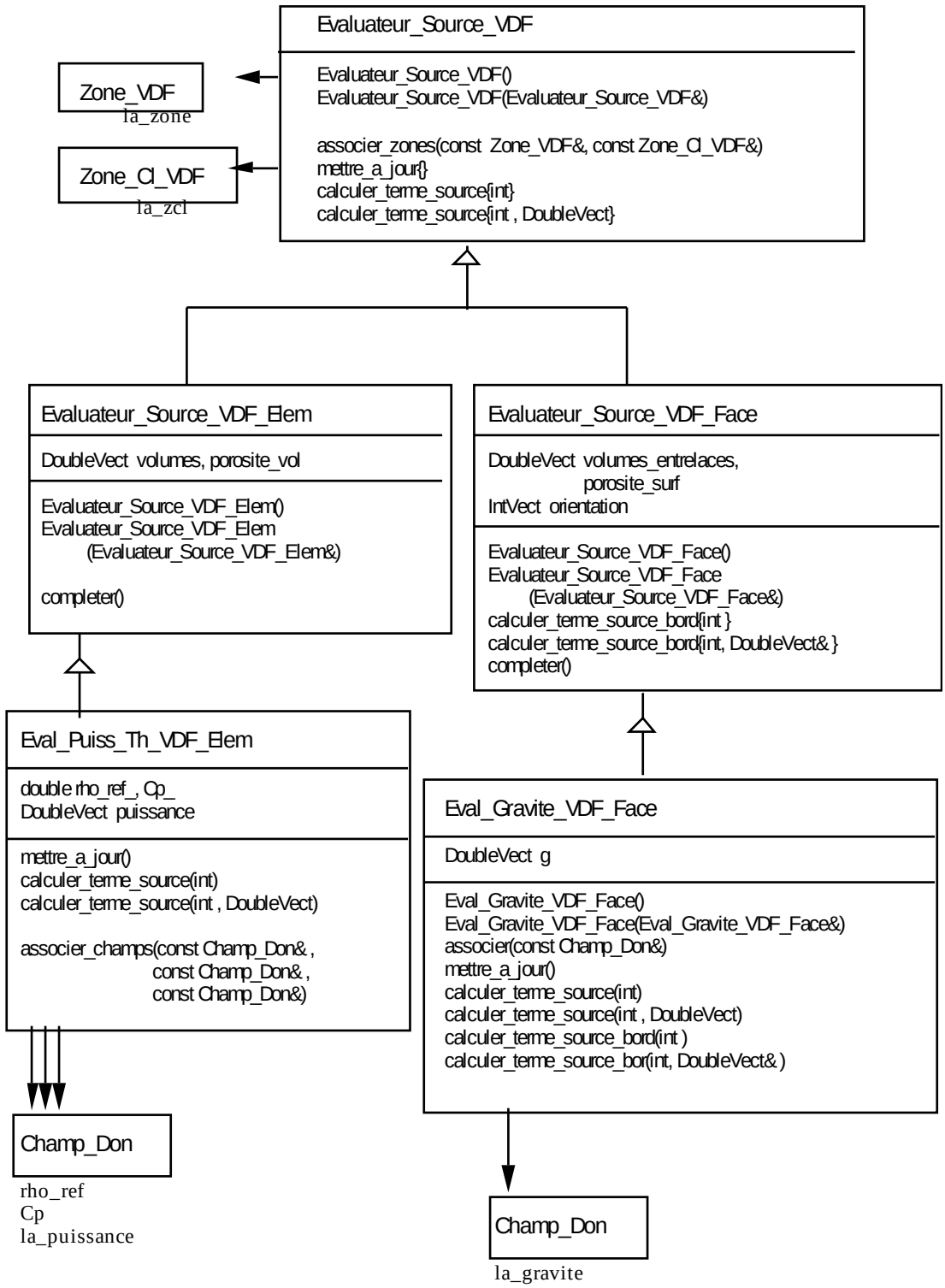
`Iterateur_Source_VDF_Face` : gestion des boucles sur les arêtes et répartition des flux sur les faces voisines

Classe de **base**: `Source_base`
`Terme_Source_VDF_base`

Classe **instanciables**:

`Terme_Gravite_VDF_Face`: terme gravité qui figure dans l'équation de Navier_Stokes divisée par la masse volumique. La masse volumique est supposée constante. Ce terme source porte un itérateur du type `Iterateur_Source_VDF_Face`. L'évaluateur porté est de type `Eval_Gravite_VDF_Face`.

`Terme_Puissance_Thermique_VDF_Elem`: terme source dans l'équation de la transport de la température qui représente le dégagement volumique de puissance thermique. Ce terme source porte un itérateur du type `Iterateur_Source_VDF_Elem`. L'évaluateur porté est de type `Eval_Puiss_Th_VDF_Elem`.



Evaluateur_Source_VDF : Gravité
Puissance thermique

6.2 Termes sources VDF traités directement :

On n'a pas intégré les classes qui suivent dans la hiérarchie `Terme_Source_VDF_base` parce que les termes sources que ces classes représentent demandent des traitements spécifiques pour certaines conditions aux limites.

6.2-1) Termes sources de Boussinesq:

Classes instanciables:

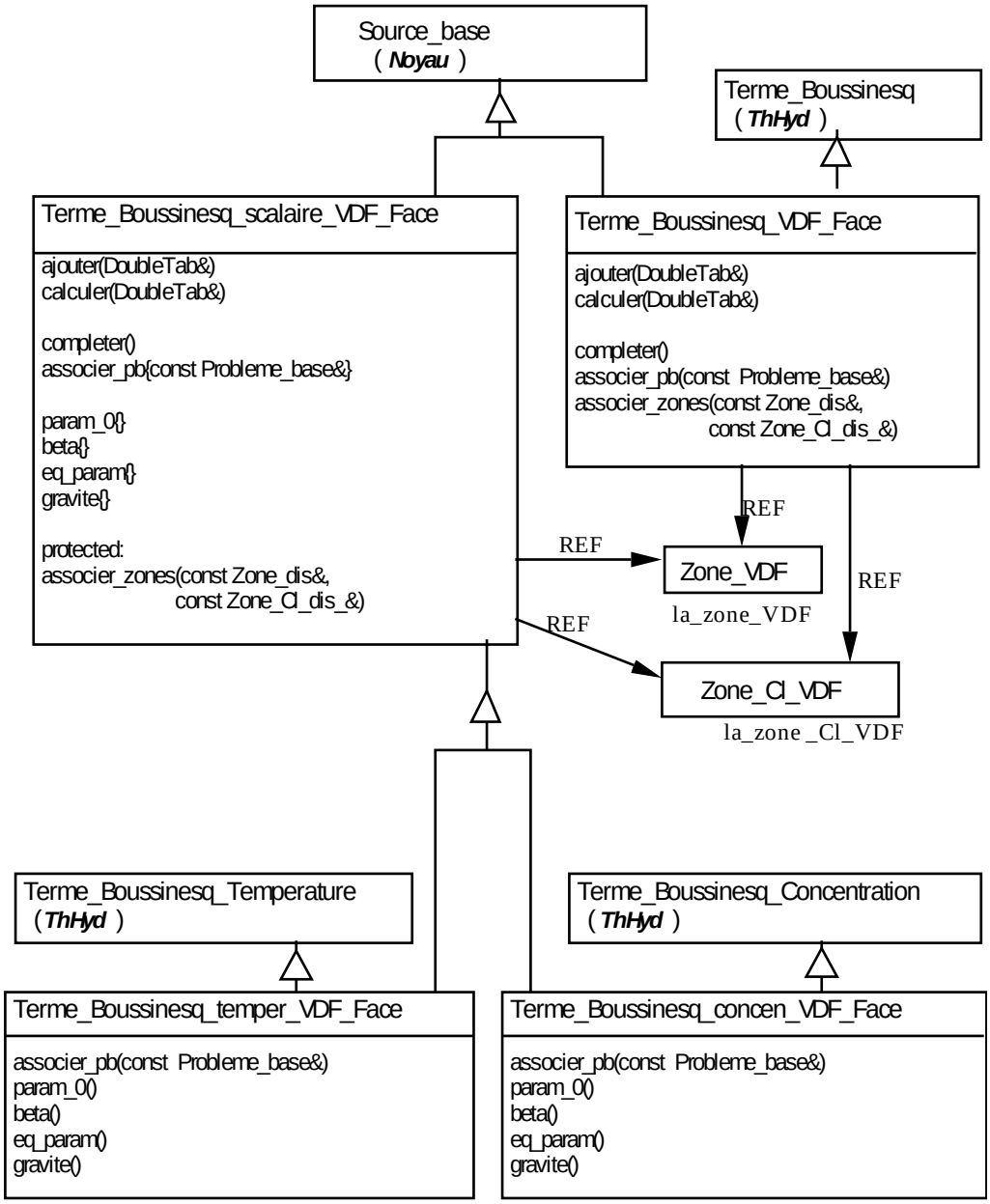
2-1-1) L'équation de Convection_Diffusion transporte un scalaire. Le champ `beta_` de variation de la masse volumique en fonction du scalaire est uniforme en espace:

`Terme_Boussinesq_temper_VDF_Face` : l'équation transporte la température

`Terme_Boussinesq_concen_VDF_Face` : le scalaire transporté est la concentration

2-1-4) L'équation de Navier-Stokes est couplée avec une équation thermique et une équation de transport de la concentration

`Terme_Boussinesq_VDF_Face` : les champs `beta` sont uniformes.



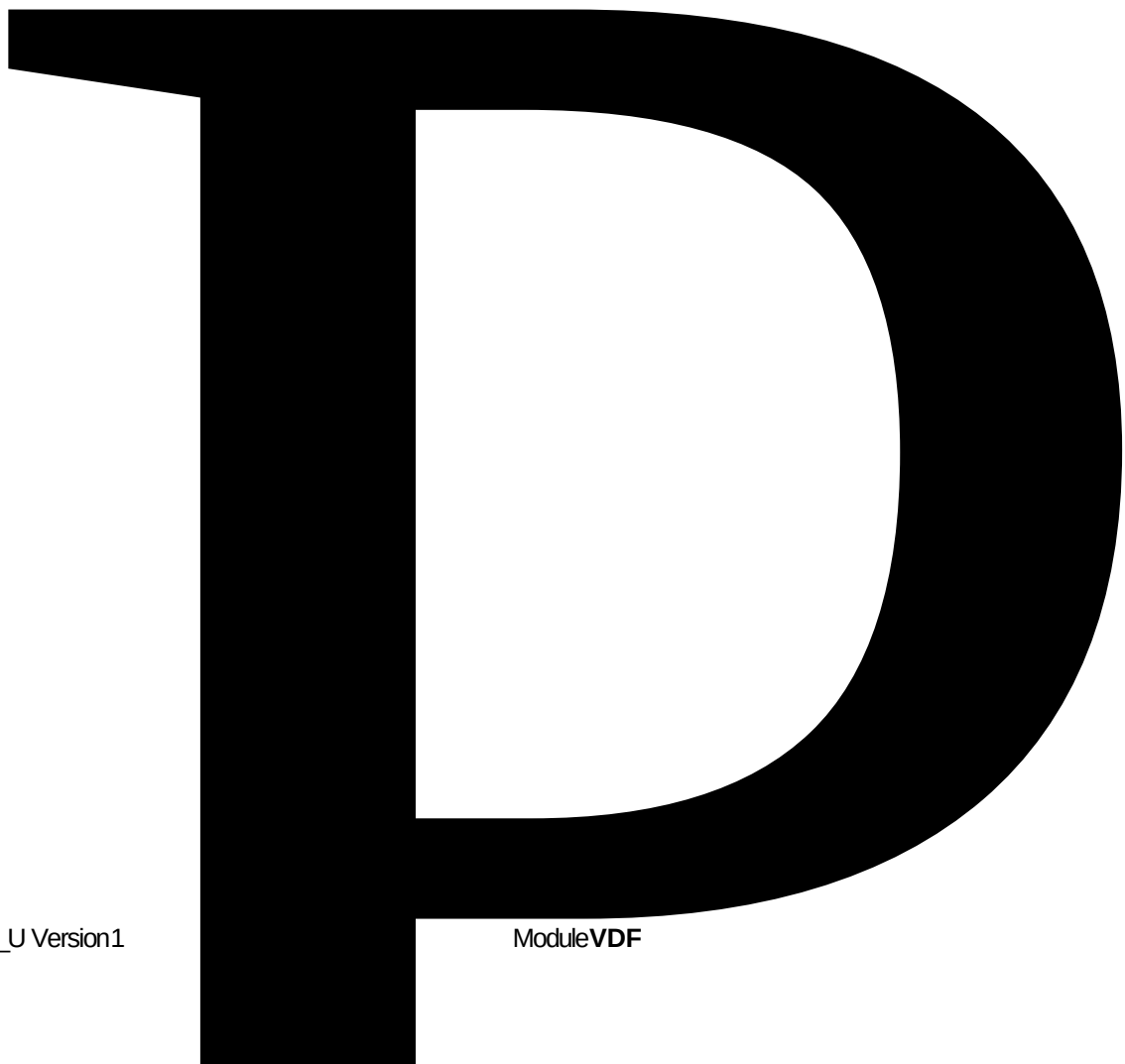
Terme Source de Boussinesq (VDF/Sources)

6.2-2) Termes sources qui figurent dans l'équation de transport du couple (K,ε):

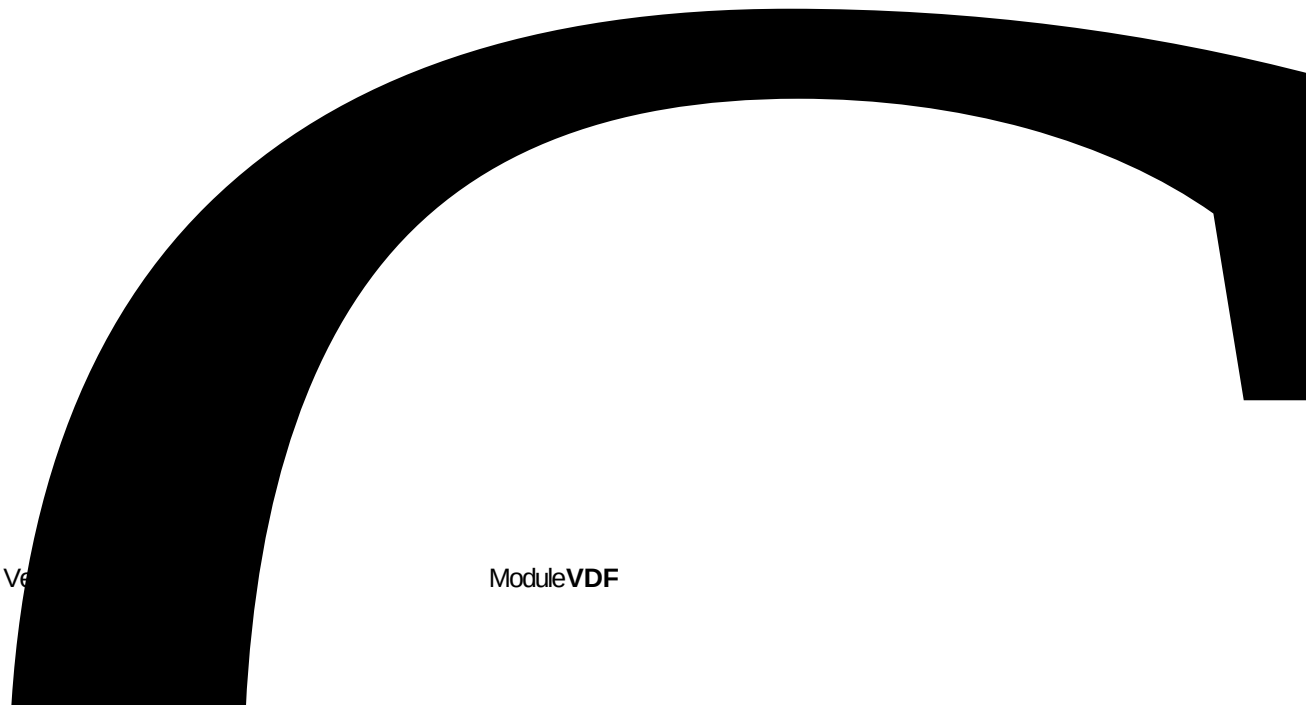
Rappel : l'équation `Transport_K_Eps` est contenue par le modèle de turbulence `Modele_turbulence_hyd_K_Eps`. Cette équation résout conjointement les deux équations de transport de l'énergie cinétique turbulente K et du taux de dissipation turbulent ε.
Les termes sources de ces équations sont eux aussi calculés ensemble, et ils sont ajoutés automatiquement.

Dans le calcul de ces termes source, interviennent:

P la production de l'énergie cinétique turbulente par l'écoulement moyen



et G la production ou la destruction de cette énergie par les forces de flottabilité.



qui sont calculés par la classe `Calcul_Production_K_VDF` .

Les classes représentant ces termes sources dérivent de la classe `Terme_Source_base` et de la classe `Calcul_Production_K_VDF` .

`Source_Transport_K_Eps_VDF_Elem` : L'équation de Navier_Stokes n'est pas couplée à une équation de transport d'un scalaire.

`Source_Transport_K_Eps_anisotherme_VDF_Elem` : L'équation de Navier_Stokes est couplée à l'équation de Convection-Diffusion de la température. On suppose que le coefficient de variation de la masse volumique en fonction de la température est uniforme.

`Source_Transport_K_Eps_aniso_therm_concen_VDF_Elem` : L'équation de Navier_Stokes est couplée à l'équation de Convection-Diffusion de la concentration et à l'équation de Convection-Diffusion de la température. Les champs `beta_t` et `beta_c` sont uniformes.

7- VDF / Solveur_Masse

Hiérarchie:		
Classe de base:	`Solveur_Masse_base`	(Noyau)
Classe générique:	`Solveur_Masse`	(Noyau)

Dans la discrétisation VDF, les matrices masse sont diagonales elles ne sont donc pas assemblées.

`Masse_VDF_Elem`: Ce solveur est utilisé pour l'équation de **transport d'un scalaire** dans une discrétisation VDF. Le champ scalaire est de type `Champ_P0_VDF` L'équation est intégrée sur les 6 faces du volume de contrôle centré sur la maille.
L'écoulement est incompressible, la densité est supposée constante.

Le terme correspondant à une maille vaut :
vol(elem)= volume géométrique(elem)* porosité volumique(elem)

La méthode `appliquer()` a en argument le vecteur `sm` défini au centre des mailles.
Elle renvoie ce même vecteur `sm` après lui avoir appliqué le solveur masse:

sm(elem)= sm(elem)/ vol(elem)

`Masse_VDF_Face`: Ce solveur est utilisé pour l'équation de **Navier-Stokes** dans une discrétisation VDF. La vitesse est de type `Champ_face` définie au centre des faces. L'équation est intégrée sur les 6 faces du volume de contrôle entrelacé centré sur la face.
L'écoulement est incompressible, la densité est supposée constante.

Le terme correspondant à une face vaut :
vol(face)= volume entrelacé(face)* porosité surfacique(face).

La méthode `appliquer()` a en argument le vecteur `sm` défini au centre des faces.
Elle renvoie ce même vecteur `sm` après lui avoir appliqué le solveur masse:

sm(face)= sm(face)/ vol(face)

Traitement des bords:
Pour les conditions aux limites de type Dirichlet ou de type Dirichlet_homogène ou de type symétrie la vitesse normale doit rester égale à sa valeur initiale.
`Solveur_Masse` écrit sm(face)= 0.

Pour les autres bords: sm(face)= sm(face)/ vol(face)

8- VDF / Solveur_Pression

Hiérarchie:		
Classe de base:	`Solveur_Pression_base`	(ThHyd)
Classe générique:	`Solveur_Pression`	(ThHyd)

`Solveur_Pression_GCP_ssor_VDF` **:** Solveur pression par l'algorithme du gradient conjugué avec pré-conditionnement SSOR pour la discrétisation VDF.

9- VDF / Turbulence

9.1 Modèles de turbulence :

Hiérarchie:		
Classe de base:	`Mod_turb_hyd_base`	(ThHyd/Turbulence)
Classe générique:	`Mod_turb_hyd`	(ThHyd/Turbulence)

`Turbulence_hyd_sous_maille_VDF` : classe instanciable repésentant le modèle "simulation des grandes échelles" pour la discrétisation VDF.
La classe générique correspondante est `Mod_turb_hyd` (ThHyd).

Hiérarchie:		
Classe de base:	`Modele_turbulence_scal_base`	(ThHyd/Turbulence)
Classe générique:	`Modele_turbulence_scal`	(ThHyd/Turbulence)

`Modele_turb_scal_Prandtl_sm_VDF` : classe instanciable repésentant le modèle de Prandtl pour la turbulence d'une équation de convection-diffusion quand l'hydraulique est calculée par le modèle de turbulence sous-maille.

9.2 Lois de paroi :

Hiérarchie:		
Classe de base:	`Turbulence_paroi_base`	(ThHyd/Turbulence)
Classe générique:	`Turbulence_paroi`	(ThHyd/Turbulence)

`Paroi_std_hyd_VDF` : détermine le cisaillement turbulent par la théorie de la couche limite isotherme.
Calcule également les grandeurs K , ε , ν_t au voisinage de la paroi.
Cette classe est instanciée quand le problème est de type `Pb_Hydraulique_Turbulent`.

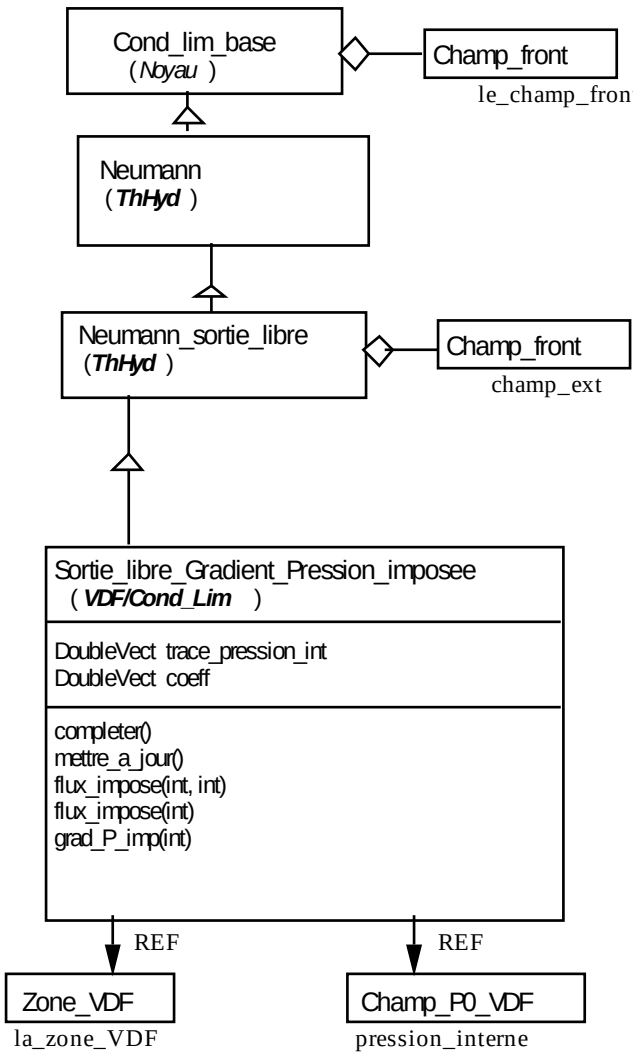
`Paroi_std_th_hyd_VDF` : Cette classe est instanciée quand les problèmes sont de type `Pb_Thermohydraulique_Turbulent` ou `Pb_Hydraulique_Concentration_Turbulent` ou `Pb_Thermohydraulique_Concentration_Turbulent` .
Elle hérite de la classe précédente et calcule les distances équivalentes à utiliser pour le calcul des coefficients d'échange à la paroi.
Elle calcule également les grandeurs α_t de la diffusivité turbulente au voisinage de la paroi.

10- VDF / Cond_Lim

Hiérarchie:		
Classe de base:	`Cond_lim_base`	(Noyau)
Classe générique:	`Cond_lim`	(Noyau)

`Sortie_libre_Gradient_Pression_impose` : Frontière ouverte avec condition de gradient de pression imposé dans une équation de Navier_Stokes.

`le_champ_front` (porté par la classe de base `Cond_lim_base`) contient la valeur du gradient imposé. La fonction `flux_impose()` renvoie une valeur de la pression à l'extérieur calculée à partir du gradient imposé et de la pression à l'intérieur du domaine.
`champ_ext` contient une valeur de la vitesse du fluide à l'extérieur accessible par la méthode `val_ext()`.



VDF / Cond_Lim

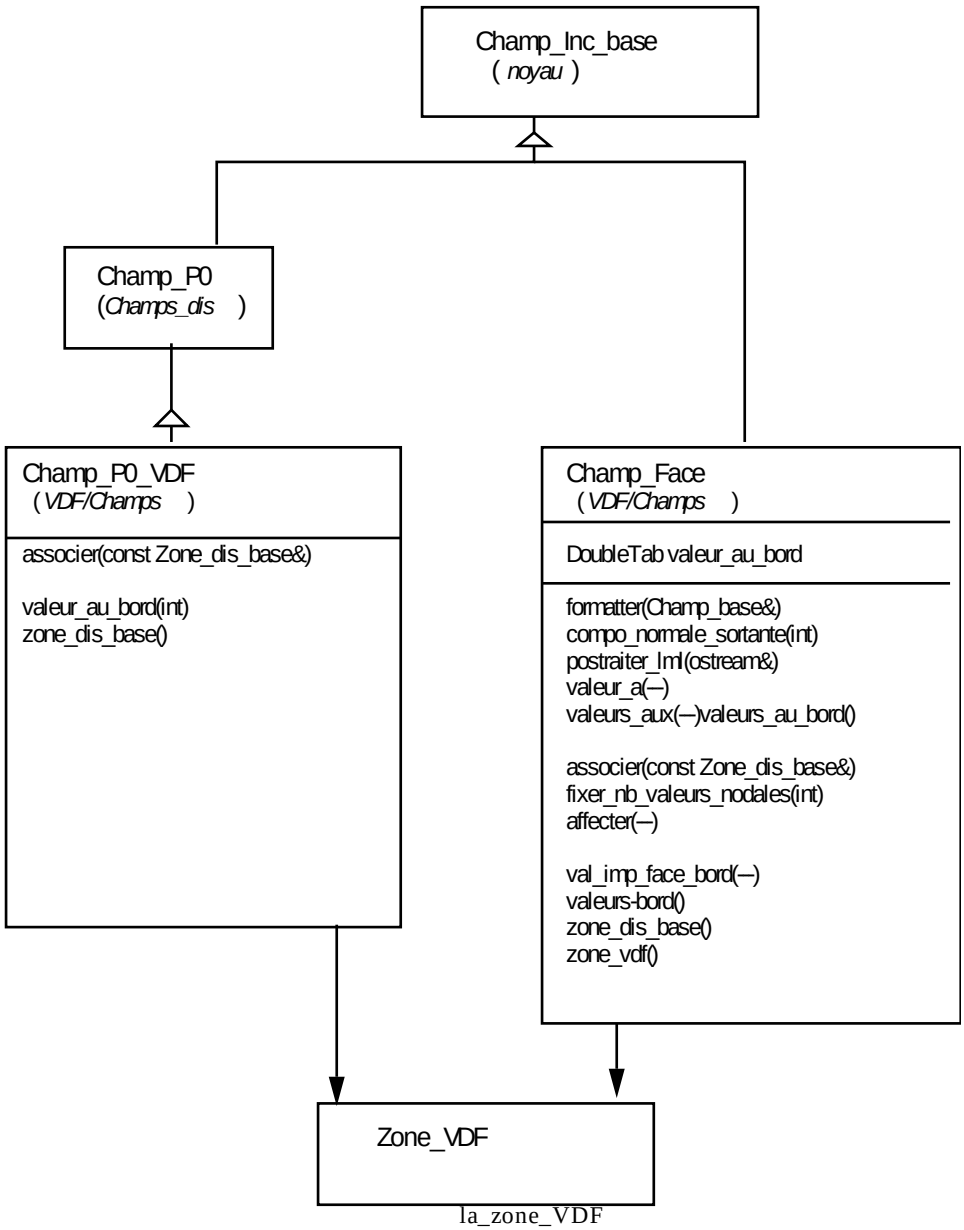
11- VDF / Champs

11.1 Classes Champ_Inc implémentées pour les VDF:

Hiérarchie:		
Classe de base:	`Champ_Inc_base`	(Noyau)
Classe générique:	`Champ_Inc`	(Noyau)

`Champ_Face` : Cette classe sert à calculer un champ vectoriel dont on ne calcule que les composantes normales aux faces. Il n' a donc qu'une composante par face, et l'attribut `nb_comp` vaut 1. On peut néanmoins imposer toutes les composantes du champ sur le bord d'un domaine. Si n est le nombre total de faces de la zone et nb_faces_bord le nombre de faces de bord, le tableau de valeurs associé au champ est construit comme suit:
- n valeurs pour représenter les composantes normales aux faces,
- nb_faces_bord* dimension pour stocker les valeurs imposées au bord.

`Champ_P0_VDF` : Cette classe représente un champ discret P0 par élément associé à une zone discrétisée de type VDF


Champ_Inc_base
(noyau)
Champ_P0
(Champs_dis)
Champ_P0_VDF
(VDF/Champs)
associer(const Zone_dis_base&)
valeur_au_bord(int)
zone_dis_base()
Champ_Face
(VDF/Champs)
DoubleTab valeur_au_bord
formatter(Champ_base&)
compo_normale_sortante(int)
postraiter_lml(ostream&)
valeur_a(–)
valeurs_aux(–)valeurs_au_bord()
associer(const Zone_dis_base&)
fixer_nb_valeurs_nodales(int)
affecter(–)
val_imp_face_bord(–)
valeurs-bord()
zone_dis_base()
zone_vdf()
Zone_VDF
la_zone_VDF

VDF / Champs

11.2 Classes Champ_Fonc implémentées pour les VDF:

Hiérarchie:		
Classe de base:	`Champ_Fonc_base`	(Noyau)
Classe générique:	`Champ_Fonc`	(Noyau)

`Champ_Fonc_Face` : Cette classe contient un champ vectoriel dont on ne calcule que les composantes normales aux faces. Il n' a donc qu'une composante par face, et l'attribut `nb_comp` vaut 1.

`Champ_Fonc_P0_VDF` :Cette classe représente un champ discret P0 par élément associé à une zone discrétisée de type VDF

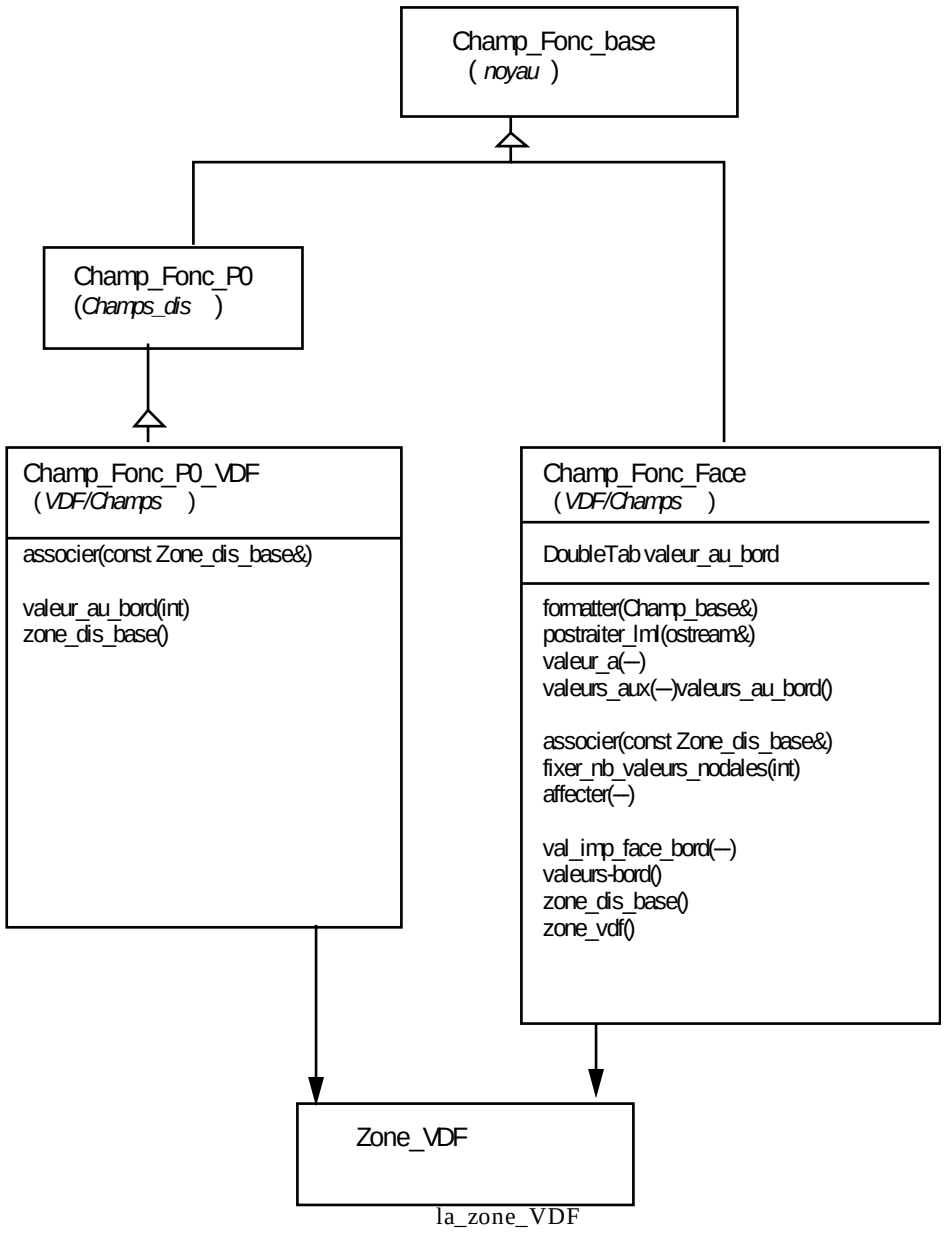
Classes instanciées par le modèle de turbulence `Mod_turb_hyd_ss_maille` quand la discrétisation est de type VDF:

Classes servant au calcul de l'intégrale en temps de la pression:

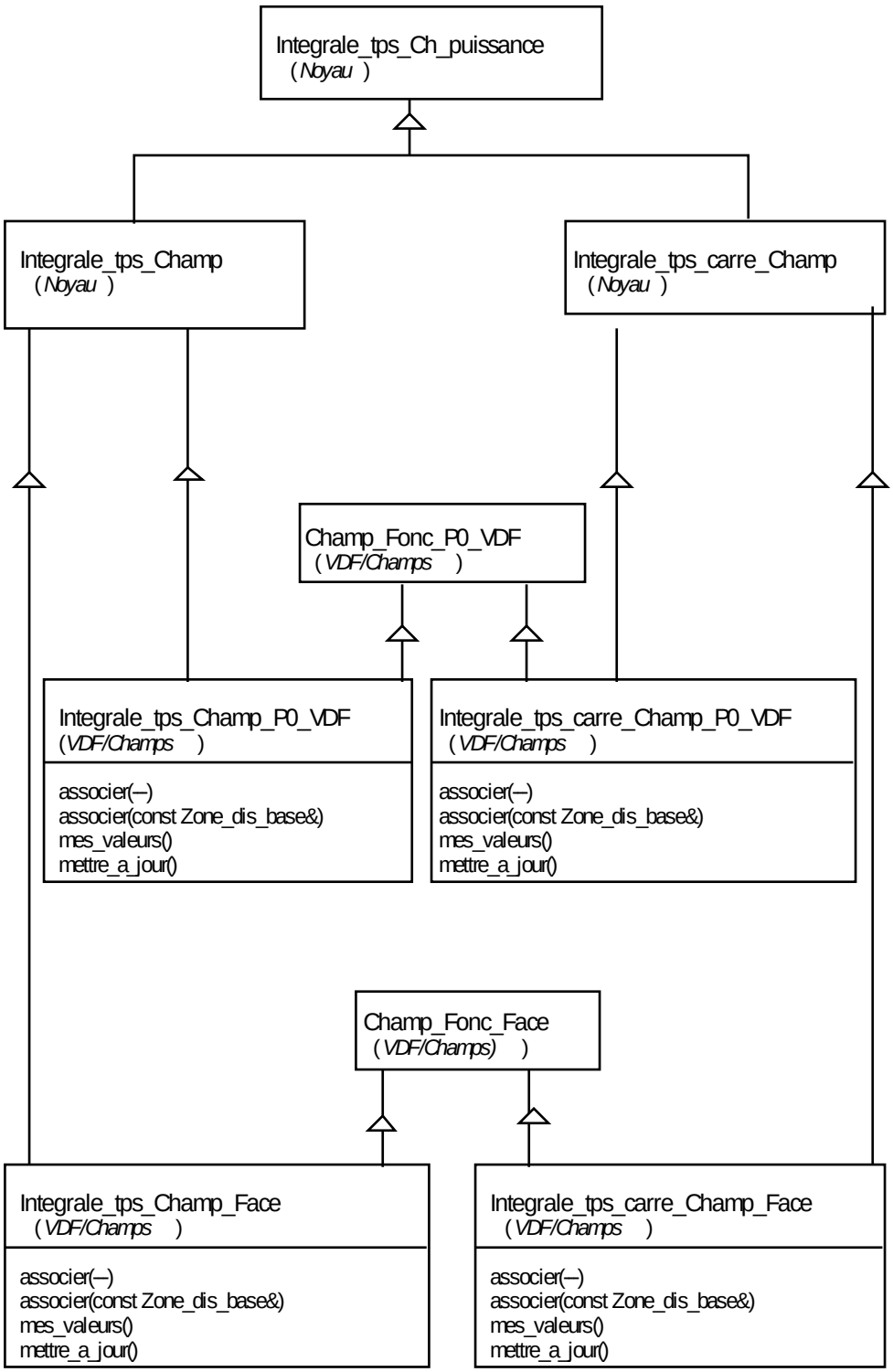
```
Integrale_tps_Champ_P0_VDF
Integrale_tps_carre_Champ_P0_VDF
```

Classes servant au calcul de l'intégrale en temps de la vitesse:

```
Integrale_tps_Champ_Face
Integrale_tps_carre_Champ_Face
```



VDF / Champ_Fonc



VDF / Champ_Fonc

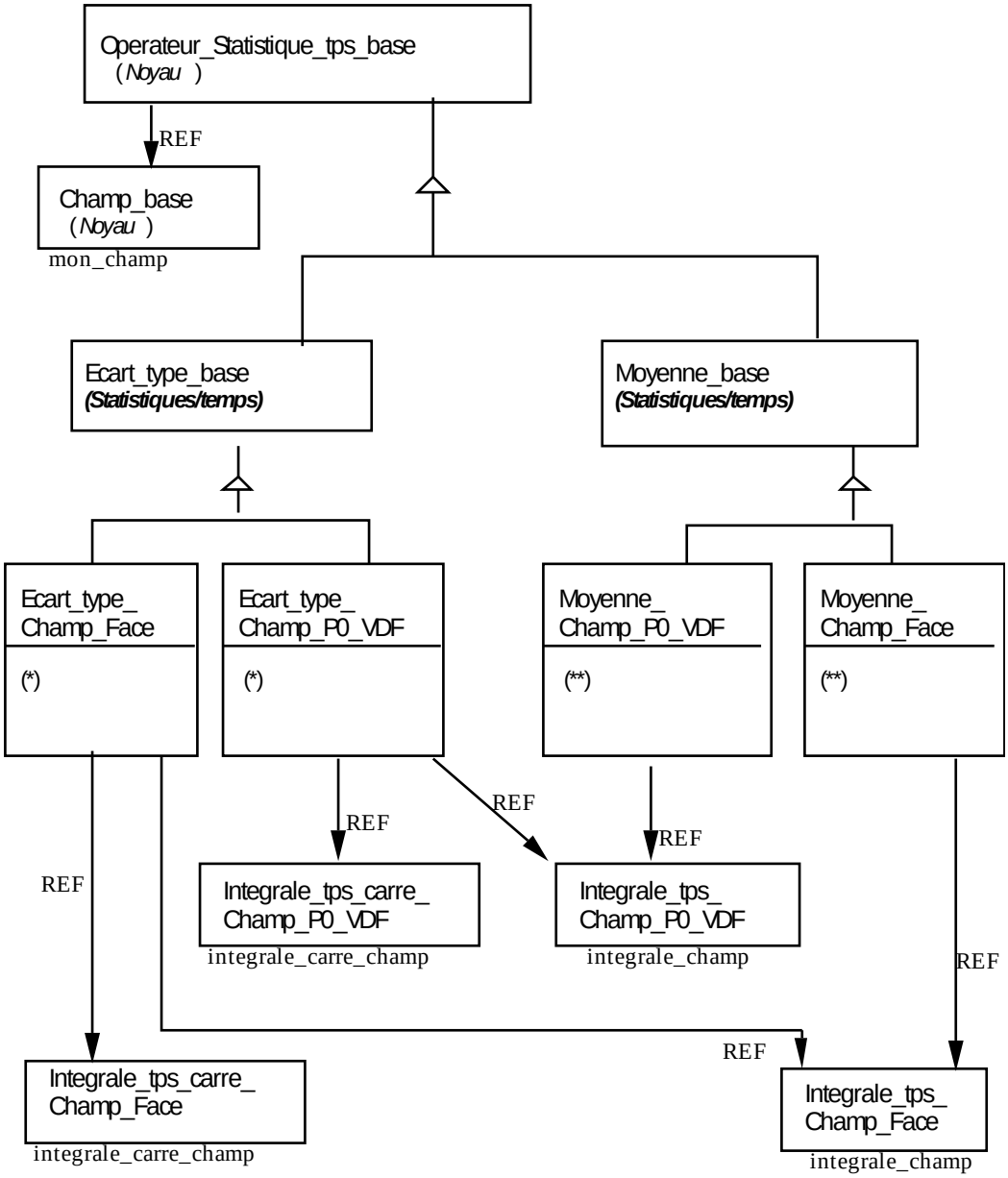
VDF / Statistiques_temps

Hiérarchie:		
Classe de base:	`Operateur_Statistique_base`	(Noyau)
Classe générique:	`Operateur_Statistique`	(Noyau)

Classes réalisant les calculs statistiques des champs de type `Champ_P0_VDF` et `Champ_Face` par rapport au temps. Ces classes sont instanciées par la classe `Postraitement`.

`Moyenne_Champ_P0_VDF` : calcul de la moyenne d'un `Champ_P0_VDF`
`Moyenne_Champ_Face` : calcul de la moyenne d'un `Champ_Face`

`Ecart_Type_Champ_P0_VDF` : calcul de l'écart type d'un `Champ_P0_VDF`
`Ecart_Type_Champ_Face` : calcul de l'écart type d'un `Champ_Face`



VDF / Statistiques_temps

Méthodes des classes:

(*) :	(**)
completer(Probleme_base&) valeurs() valeurs_carre() dt_integration() dt_integration_carre() postraiter_lm(Sortie&)	completer(Probleme_base&) valeurs() dt_integration() postraiter_lm(Sortie&)

REFERENCES

Documentation TRIO_U Version 1

STR/LTML/96-20	Document de conception TRIO_U Version 1	
STR/LTML/96-21	Manuel d'utilisation TRIO_U Version 1	
STR/LTML/96-22	Manuel de validation TRIO_U Version 1	
STR/LTML/96-23	Note de présentation TRIO_U Version 1	[PRES1]

Remarque 1 :

Dans l'attente de la parution de la "Note de présentation", on trouvera la description des schémas de discrétisation VDF (Volumes différences finies) dans la note de présentation de TRIO_VF:

STR/LML/95-349 TRIO-VF Note de Présentation
Version complète 8.6 de décembre 95
M. Villand

Le document à paraître comprendra les descriptions des schémas VDF et VEF.

Remarque 2 :

Un additif à ce document de conception paraîtra en septembre 96 : il comprendra le module VEF (Volumes Elements Finis) et un additif au module VDF pour l'introduction des coordonnées cylindriques.