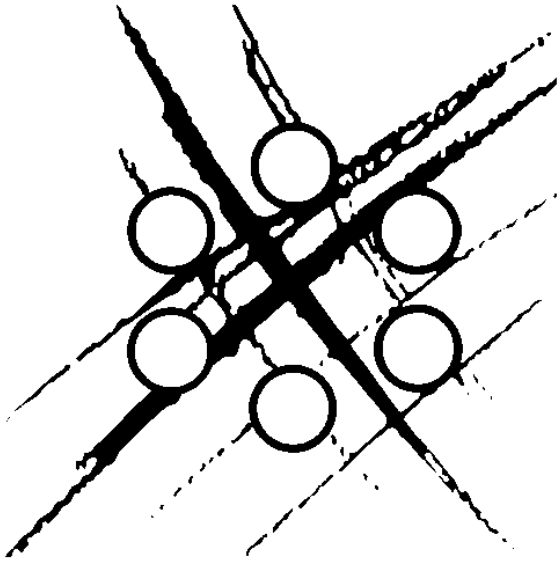
DIRECTION DE L'ÉNERGIE NUCLEAIRE

DIRECTION DELEGUEE AUX ACTIVITES NUCLEAIRES DE SACLAY

DEPARTEMENT DE MODELISATION DES SYSTEMES ET STRUCTURES

SERVICE FLUIDES NUMERIQUES, MODELISATION ET ETUDES



RAPPORT DM2S

SFME/LGLS/RT/09-017/A

Normalisation des champs et des maillages pour le couplage de codes

AUTEURS

A. GEAY, F.PERDU

COMMISSARIAT A L'ÉNERGIE ATOMIQUE

DIRECTION DE L'ENERGIE NUCLÉAIRE
DIRECTION DELEGUEE DES ACTIVITES NUCLEAIRES
DÉPARTEMENT DE MODELISATION DES SYSTEMES ET STRUCTURES
SERVICE FLUIDES NUMERIQUES, MODELISATION ET ETUDES

RAPPORT DM2S

RÉFÉRENCE : SFME/LGLS/RT/09-017/A

TITRE : Normalisation des champs et des maillages pour le couplage de codes.

AUTEURS	SIGNATURES	AUTEURS	SIGNATURES
A. GEAY		F. PERDU	

Ce rapport a pour objet de définir un modèle de champs et de maillage pour le couplage de codes. De ce modèle de maillages et de champs, une interface sera proposée. L'architecture de l'implémentation associée à cette interface sera explicitée.

MOTS CLES : Salome, MED, couplage, normalisation
AFFAIRE : DOB/Domaine DSOE/Simulation EOTP A-PICI2-01
Titre de l'affaire : développement

			Visa			
			Nom	N. CROUZET		V. BERGEAUD
A	11/12/09	22	Date			
Indice	**Date**	**Nb.Pages**		**Vérificateur**	**Autre visa**	**Approbateur**

diffusion : normale ☒ *restreinte* ☐ *confidentiel CEA* ☐ *CD* ☐ *SD* ☐

LISTE DE MODIFICATION

Indice	Date	Motif et description de la modification
A		Document initial

SOMMAIRE

1. Vocabulaire

Cellule : entité géométrique appartenant à un maillage, et de dimension égale à celle de son maillage.

Sous-cellule ou face : entité géométrique appartenant à un maillage, de dimension égale à celle de son maillage moins 1.

SPMD : Single program multiple data. Paradigme de programmation parallèle qui consiste à faire coopérer plusieurs même instances de programmes. MPI et PVM sont des interfaces offrant des services permettant une implémentation selon ce paradigme.

Tuple : Un tuple est une collection de n (fixé) éléments ayant un même type informatique (typiquement `int` ou `double`). Néanmoins, la sémantique de chacun des éléments constitutifs du tuple peut être différente de celles des autres éléments du tuple.

2. Introduction

Le module MED (modèle d'échange de données) de Salomé a été conçu, comme son nom le sous-entend, pour échanger des données entre codes de manière générale. Cet échange se fait selon deux canaux principaux : les fichiers et l'échange de process à process pour le couplage.

Historiquement la librairie MEDMEM incluse dans MED a implémenté des structures de données suivant au plus près le modèle de MED fichier existant. De fait, cette structure de donnée MEDMEM est très riche pour pouvoir gérer l'étendue des possibilités couvertes par MED fichier. Cette richesse découle de l'objectif de MED fichier de proposer une structure couvrant l'union des demandes venant de disciplines différentes.

Pour couvrir sa responsabilité de couplage, une couche CORBA cliente et serveur a été développée dans le module MED, basée sur la structure de donnée de MEDMEM tout juste implémentée. Il est apparu par la suite que pour le couplage, synchrone par CORBA ou asynchrone par MPI, une simplification de la structure de donnée sous-jacente était souhaitable. Cette structure simplifiée pour le couplage prenant la forme informatique d'une librairie a été nommée MEDCoupling. MEDMEM a vocation à conserver quant à elle sa mission première de manipulation de structures calquées aux fichiers MED.

Outre sa mission de couplage, MEDCoupling doit être développé en conformité avec l'API ICoco [2], les interpolations conservatives et la structure de visualisation VTK. De par sa vocation, MEDCoupling peut et doit présenter une liste de dépendances la plus maitrisée possible pour éviter les écueils rencontrés précédemment (conflits de versions de dépendances HDF5 par exemple). MEDCoupling doit bien sûr être compatible avec les contraintes HPC (structure compacte, limitation des copies et du déclenchement des algorithmes intensifs...). Enfin, ces structures de données doivent être également exploitables par Hxx2salome pour générer automatiquement des wrappers Salomé à partir de codes échangeant du MEDCoupling.

Nous présenterons ici une spécification et une conception de MEDCoupling afin qu'il puisse répondre aux différents items cités ci-dessus.

3. Spécifications fonctionnelle

3.1. MAILLAGES

Un maillage possède comme attributs non mutable :

- un nom
- une dimension, ce qui implique que toutes les cellules constituant le maillage ont la même dimension. Dans le cas général, elle est égale ou supérieure à 1. Pour les maillages non structurés cette valeur peut également prendre en plus les valeurs 0 ou -1 comme on l'explicitera au §3.2.
- une dimension d'espace
- une unité de longueur par dimension de l'espace

Un maillage, quelque soit sont type, peut retourner :

- son nombre de nœuds
- son nombre de cellules

De cette classe maillage on concrétise deux classes : les maillages non structurés et les maillages cartésiens.

3.2. MAILLAGES NON-STRUTURES

La classe de maillages non-structurés doit offrir 2 modes de remplissage. L'un direct, qui autorise à l'utilisateur avancé connaissant la convention des tableaux d'assigner directement ses tableaux. Un autre plus ergonomique, de description cellule par cellule. Les maillages non-structurés doivent offrir également deux modes de stockage :

- Connectivité nodale des cellules ou connectivité descendante des cellules et nodale des sous cellules
- Coordonnées des nœuds

Afin d'autoriser un maximum de types de code différents, il est nécessaire d'interpoler depuis ou vers des maillages dégénérés. Les maillages non-structurés, de part leur structure plus dynamique, auront la responsabilité de gérer les maillages plus spécifiques suivants :

- Maillage de nuages de points. Ces maillages un peu particuliers seront constitués de cellules de type POINT de dimension égale à 0.
- Maillage abstrait. Il s'agit d'un maillage contenant exactement une seule cellule et aucun nœud. Ce type de maillage est utilisé pour représenter un champ contenant une seule valeur intégrée sans aucun support spatial. Ce cas apparait typiquement lorsque l'on

couple un code système raisonnant sur une valeur intégrée sur la totalité de l'espace d'un champ avec un code d'éléments finis raisonnant sur un champ discrétisé spatialement.

Contrairement à la tendance de MED fichier les cellules constituant le maillage de MEDCoupling n'ont pas obligatoirement vocation à être groupées par type de cellules.

3.3. MAILLAGES CARTESIENS

La classe des maillages cartésiens représente un maillage structuré dont les nœuds et les cellules sont rangés selon les axes du trièdre. Ainsi pour les maillages de ce type la donnée de n tableaux à une composante de flottants (n égal à la dimension de l'espace et du maillage) suffit à définir le maillage. Ces maillages cartésiens auront une numérotation nodale et des cellules. La numérotation des cellules et des nœuds sont numérotés selon les x croissants puis les y croissants puis les z croissants. Ainsi chacun des n tableaux devra être une suite monotone.

3.4. CHAMPS

On décrit ici ce qui sera entendu par champ dans MEDCoupling. La notion de champ de MEDCoupling permet de décrire la notion théorique de champ définie dans [2]. La différence entre les deux définitions sera plus largement explicitée au §8.

Un champ au sens MEDCoupling possède d'abord un support spatial discrétisé qui sera un maillage. Le champ MEDCoupling portera sur ***toutes*** les entités, spécifique à sa discrétisation spatiale, de son maillage support, contrairement à ce qui existe dans MED fichier et MEDMEM. Il sera néanmoins toujours possible d'exploiter des supports (familles et groupes) MED fichier grâce à des méthodes d'extraction de maillage. Le champ ne modifiera jamais son maillage support. MEDCoupling veillera à ce que l'utilisateur ne modifie pas le contenu d'un maillage support d'une instance de champ.

Pour conclure le champ MEDCoupling possède comme propriété non mutable :

- Un nom
- Un maillage non mutable
- Une discrétisation spatiale, qui spécifie s'il s'agit d'un champ aux cellules ou aux éléments par exemple
- Un flag description de la nature intrinsèque des valeurs du champ : Conservatif et volumique, intégral ou intégral avec contraintes globales [2]
- Une discrétisation temporelle définissant l'intervalle de temps, s'il existe, sur lequel l'instance décrit le champ
- Nombre de composantes (égale pour tous les tuples)
- Nombre de tuples
- Autant de valeurs spatio-temporelles fonction de la discrétisation spatiale et temporelle du champ que nécessaire

3.4.1. Discrétisation spatiale

Cette information permet de connaître le lien qui existe, indépendamment de la discrétisation temporelle, entre les valeurs du champ et son maillage. Cette classe aura la responsabilité de calculer en tout point de l'espace du maillage la valeur du champ. Les discrétisations spatiales classiques sont les champs :

- P0 : champ constant par cellule de son maillage
- P1 : champ défini au nœud de son maillage et dont la valeur en un point P quelconque du maillage est la combinaison linéaire des nœuds constituants la cellule dans laquelle se trouve P
- P1NC : champ défini aux faces de son maillage et dont la valeur en un point P quelconque du maillage est la combinaison linéaire des valeurs aux faces constituant la cellule dans laquelle se trouve P

3.4.2. Discrétisation temporelle

Cette information permet, indépendamment de la discrétisation spatiale, d'associer un intervalle de temps sur lequel le champ qui l'agrège sera défini. Cet intervalle peut être, et cela sera souvent le cas, de longueur nulle. Cette classe aura la responsabilité de donner la valeur du champ en tout temps de son intervalle ainsi que la valeur en début et fin de cet intervalle.

3.5. ASPECTS INFORMATIQUES

A côté de ces spécifications fonctionnelles orientées physique, plusieurs aspects informatiques doivent être remplis par cette structure de donnée.

En vue de remplir les contraintes liées au HPC, la gestion de la mémoire allouée dynamiquement doit être la plus efficace possible. Ainsi, la gestion de la mémoire dynamique par la bibliothèque doit limiter au maximum les copies de données et gérer le partage des données pour les objets les plus volumineux. Ceci passe par un comptage de référence embarqué aux objets volumineux.

Enfin, toujours en vue d'intégrer les contraintes liées au HPC, les algorithmes consommateurs de CPU tel que les interpolateurs doivent pouvoir n'être déclenchés que lorsque cela est vraiment nécessaire. D'où la nécessité d'intégrer un système orienté demande. Ce système implique l'implémentation dans la bibliothèque d'une gestion de date universelle qui permet de détecter les éventuelles mise-à-jour ou non d'objets volumineux en mémoire tels que les objets maillage et pouvoir ainsi déclencher l'exécution d'un minimum d'algorithmes (lazy evaluation).

4. Spécifications de l'implémentation

Ici va être décrit l'ensemble des interfaces haut niveau attendues des utilisateurs de la bibliothèque.

4.1. MAILLAGE

Voici l'interface commune à tous les types de maillages :

- `void setName(const char *name)`
- `const char *getName() const`
- `bool isEqual(const MEDCouplingMesh *other, double prec) const`
- `void checkCoherency() const throw(INTERP_KERNEL::Exception)`
- `int getNumberOfCells() const`
- `int getNumberOfNodes() const`
- `int getSpaceDimension() const`
- `int getMeshDimension() const`
- `void getBoundingBox(double *bbox) const`
- `MEDCouplingFieldDouble *getMeasureField() const`

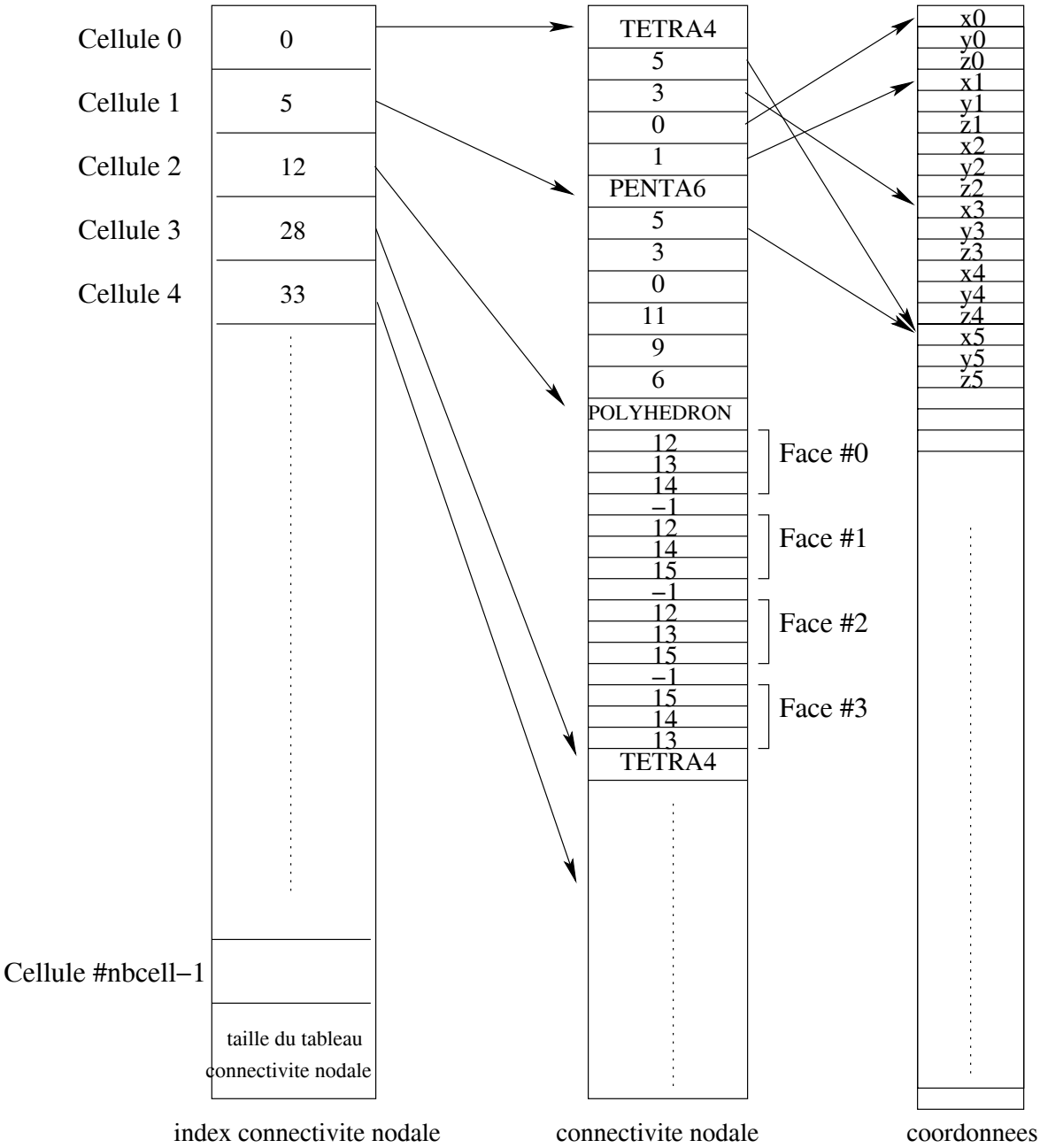
La méthode `checkCoherency` vérifie l'allocation des tableaux et la cohérence de leur longueur ainsi que les types des cellules soient cohérents avec la dimension du maillage.

La méthode `getMeasureField` retourne un champ P0 donnant la mesure de chacune des cellules du maillage sur lequel a été appelée cette méthode. On appelle mesure d'une cellule d'un maillage, son volume si le maillage est de dimension 3, son aire s'il est de dimension 2 ou sa longueur pour le maillage de dimension 1. La classe abstraite responsable du maillage générique aura le nom `MEDCouplingMesh`.

4.2. MAILLAGE NON-STRUCTURE EN CONNECTIVITE NODALE

4.2.1. Mode de stockage

On va décrire ici comment la connectivité nodale du maillage non structuré est stockée au sein de `MEDCouplingUMesh`. Le but de ce stockage est d'être compatible avec celui spécifié et implémenté dans MED fichier et donc dans MEDMEM. De plus, il est choisi de rendre ce format de stockage le plus proche possible de son contexte d'utilisation (langage C/C++, interaction avec la librairie de visualisation VTK, et la possibilité d'avoir une interface d'édition de maillage conviviale). Pour réussir à obtenir ce compromis, le stockage MEDCoupling conservera de la norme MED fichier l'orientation des cellules et les types. Le stockage en revanche sera à la convention C (0,n-1) et non à la convention FORTRAN (1,n). De plus le rangement par type n'existera plus pour faciliter l'insertion de cellules. Enfin toujours dans un souci de confort et d'homogénéisation de l'implémentation, les polyèdres et les polygones ne seront pas traités séparément. Voici le mode de stockage ainsi retenu :



Dans la figure précédente TETRA4, PENTA6 et POLYHEDRON sont des enum donc stockable comme des entiers.

A noter que le mode de stockage MED de la connectivité des polyèdres est constitué de 3 tableaux : un d'index des cellules, un d'index des faces constituant les cellules et enfin la connectivité nodale des faces. Comme on ne dispose que de deux tableaux pour décrire la connectivité et que l'on désire stocker les polyèdres de la même façon que les autres type on fait disparaitre le tableau d'index des faces par l'insertion de « balises » -1 dans le tableau de connectivités.

4.2.2. Interface

A l'interface de la classe abstraite des maillages se rajoutent les méthodes spécifiques suivantes :

- `static MEDCouplingUMesh *New()`
- `static MEDCouplingUMesh *New(const char *name, int meshDim)`
- `MEDCouplingUMesh *clone(bool recDeepCpy) const`
- `void allocateCells(int nbOfCells)`
- `void insertNextCell(INTERP_KERNEL::NormalizedCellType type, int size, const int *nodalConnOfCell)`
- `void finishInsertingCells()`
- `void setConnectivity(DataArrayInt *conn, DataArrayInt *connIndex, bool isComputingTypes)`
- `DataArrayInt *getNodalConnectivity() const`
- `DataArrayInt *getNodalConnectivityIndex() const`
- `INTERP_KERNEL::NormalizedCellType getTypeOfCell(int cellId) const`
- `int getNumberOfNodesInCell(int cellId) const`
- `void zipCoords() (voir description plus loin)`
- `DataArrayInt *zipCoordsTraducer() (voir description plus loin)`
- `void getReverseNodalConnectivity(DataArrayInt *revNodal, DataArrayInt *revNodalIndx) const`
- `MEDCouplingPointSet *buildPartOfMySelf(const int *start, const int *end, bool keepCoords) const`

Les methodes `zipCoords` et `zipCoordsTraducer` permettent de renuméroter les nœuds de l'instance afin qu'aucun nœud ne soit inutilisé dans la connectivité nodale de toute l'instance. La deuxième méthode retourne en plus un tableau de correspondance entre l'ancienne numérotation et la nouvelle.

A noter que la méthode `insertNextCell` est une méthode de confort qui permet à l'utilisateur de définir la connectivité de son maillage cellule par cellule et ce quel que soit le type.

4.3. MAILLAGE NON-STRUCTURE EN CONNECTIVITE DESCENDANTE

Il s'agit aussi d'une classe qui hérite de la classe `MEDCouplingMesh` mais qui possède un mode de stockage différent pour pouvoir définir des champs avec une discrétisation spatiale PNC. Voici les méthodes spécifiques dont on désire disposer pour cette classe :

- `static MEDCouplingUMeshDesc *New()`
- `void setMeshDimension(unsigned meshDim)`
- `int getNumberOfFaces() const`
- `void setConnectivity(DataArrayInt *descConn, DataArrayInt *descConnIndex, DataArrayInt *nodalFaceConn, DataArrayInt *nodalFaceConnIndx)`
- `MEDCouplingPointSet *buildPartOfMySelf(const int *start, const int *end, bool keepCoords) const`

4.4. MAILLAGES CARTÉSIENS

Voici l'API spécifique aux maillages cartésiens :

- `static MEDCouplingCMesh *New()`
- `DataArrayDouble *getCoordsAt(int i) const`
- `void setCoords(DataArrayDouble *coordsX,DataArrayDouble *coordsY=0, DataArrayDouble *coordsZ=0)`

4.5. CHAMPS

Ne sont gérés pour le moment dans `MEDCoupling` que les champs flottants double précision. Ces champs devront présenter l'API suivante :

- `static MEDCouplingFieldDouble *New(TypeOfField type, TypeOfTimeDiscretization td=NO_TIME)`
- `void checkCoherency() const throw(INTERP_KERNEL::Exception)`
- `void setMesh(const ParaMEDMEM::MEDCouplingMesh *mesh)`
- `const MEDCouplingMesh *getMesh() const`
- `void setName(const char *name)`
- `void setDescription(const char *desc)`
- `const char *getName() const`

- `TypeOfField getTypeOfField() const`
- `MEDCouplingFieldDouble *clone(bool recDeepCpy) const`
- `void setTime(double val, int dt, int it)`
- `double getTime(int& dt, int& it)`
- `double getIJ(int tupleId, int compoId) const`
- `void setArray(DataArrayDouble *array)`
- `DataArrayDouble *getArray() const`
- `double accumulate(int compId) const`
- `double measureAccumulate(int compId, bool isAbs) const`
- `void applyLin(double a, double b, int compoId)`
- `int getNumberOfComponents() const`
- `int getNumberOfTuples() const throw(INTERP_KERNEL::Exception)`
- `void getValueOn(const double *spaceLoc, double *res) const`
- `void getValueOn(const double *spaceLoc, double time, double *res) const`
- `void getValueOnD(const double *spaceLoc, int eltId, double *res) const`
- `void getValueOnD(const double *spaceLoc, double time, int eltId, double *res) const`

La méthode `accumulate` effectue une somme de la compId^{ieme} composante sur tout le champ. De la même façon `measureaccumulate` effectue la somme pondérée de compId^{ieme} composante du champ, la pondération dépendant du type de champ. Ces deux méthodes servent à vérifier le caractère conservatif d'une interpolation.

Comme on le voit le constructeur de champs par méthode statique nécessite deux paramètres, l'un de discrétisation spatiale et l'autre de discrétisation temporelle. Ces paramètres auront une influence directe sur :

- les interpolations
- les méthodes `getValueOn` et `getValueOnD` (équivalent de `getValueOn` pour le conforme P1 en ignorant le paramètre `eltId`, pour le discontinu P0 et P1NC permet de spécifier la cellule que l'on souhaite considérer)

- La méthode `checkConsistency` qui vérifie la cohérence entre le nombre de tuples du champ par rapport à la discrétisation et le maillage support

4.6. DATAARRAY

Cette classe ne sera pas instanciable mais ses classes dérivées `DataArrayDouble` et `DataArrayInt` gérant respectivement des tableaux de `int` et de `double` le seront. On utilisera par la suite le terme générique de `DataArray`, pour citer indifféremment `DataArrayDouble` ou `DataArrayInt`. La classe `DataArray` sera en charge de considérer un tableau C/C++ comme une série contigüe de tuples ayant chacun un nombre fixe d'éléments. Cette classe participe aussi aux contraintes citées au §3.5 au sens où l'on peut directement lui confier un tableau alloué et rempli éventuellement ailleurs dans une autre librairie que `MEDCoupling` et le lui assigner sans recopies. Voici les différents cas de figures possibles pour l'assignation d'un tableau C/C++ géré par `DataArray` :

- Pas d'ownership : Dans ce cas `DataArray` assigne le tableau sans recopie et considère le tableau comme const. Ce qui implique qu'aucune modification du tableau ne sera possible dans MEDCoupling ou alors une exception sera jetée. A la destruction de l'objet DataArray il ne sera fait aucune destruction du pointeur représentant ce tableau agrégé.

- Ownership : Dans ce cas `DataArray` assigne le tableau également sans recopie mais assigne aussi le mode d'allocation associé à ce tableau. Typiquement s'il s'agit d'une allocation via `malloc` (enum C_ALLOC) ou via un `new[]` (enum CPP_ALLOC)

- Copie : Dans ce cas on aura recours à une allocation et ensuite à une recopie du pointeur passé en paramètre

Charge alors à l'utilisateur de spécifier à l'instance de `DataArrayInt` ou `DataArrayDouble` si elle possède la propriété de ce tableau.

5. Conception de l'implémentation

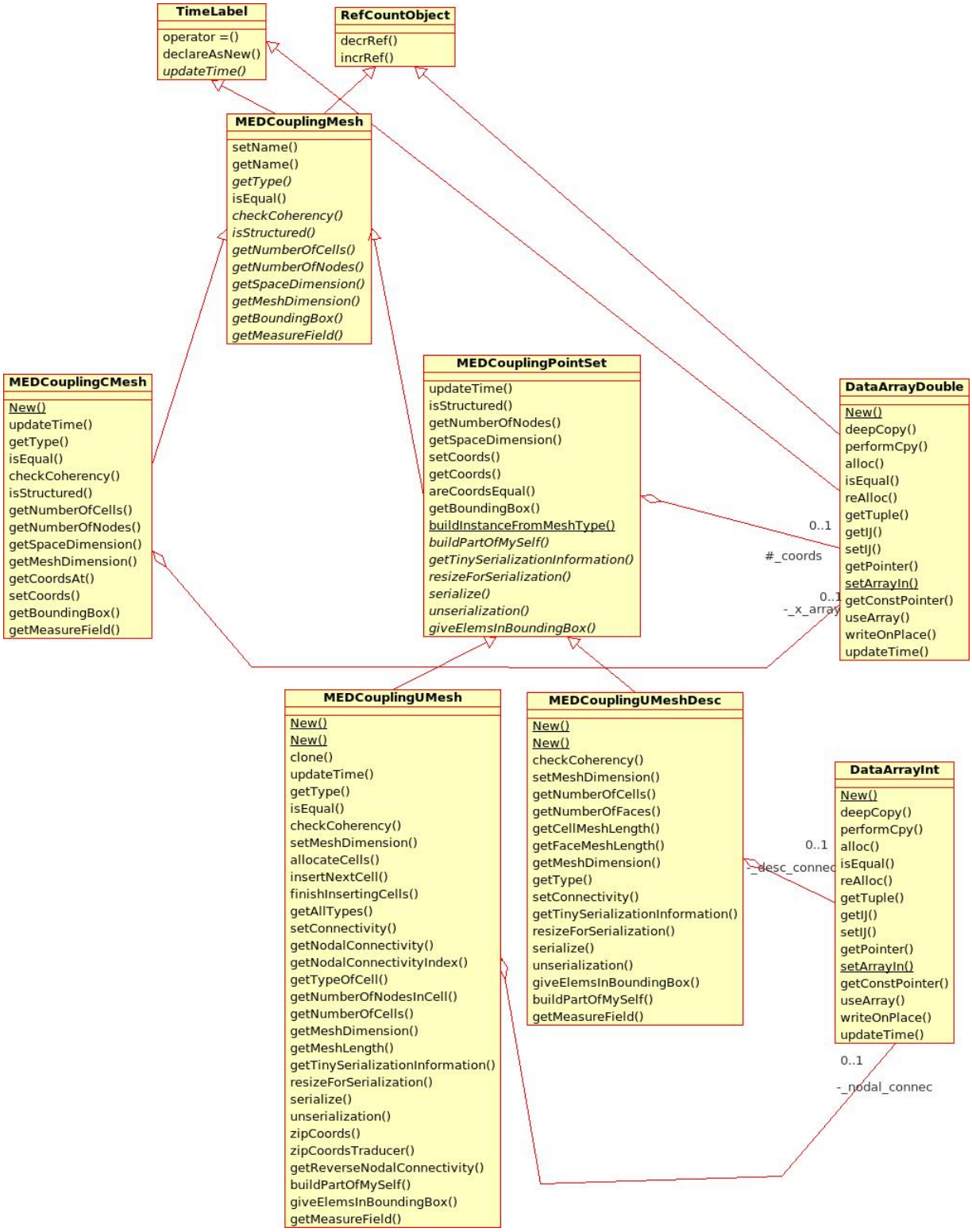
5.1. TIMELABEL

Comme on l'a vu plus haut, il est souhaitable de ne déclencher les algorithmes consommateurs en temps CPU, tels que les interpolateurs conservatifs, que lorsque ceci est nécessaire. La classe `TimeLabel` est en charge d'offrir un mécanisme permettant de connaitre « l'heure » de la dernière modification de l'état interne. « L'heure » universelle est matérialisée par une variable statique entière initialisée à 0 ne faisant que croitre. Chaque instanciation de classe héritant de `TimeLabel` stockera le temps courant et incrémentera le temps universel d'une unité. Toutes les classes instanciables héritant de `TimeLabel` doivent surcharger la méthode `updateTime` afin de mettre à jour le compteur local par rapport à toutes les instances de `TimeLabel` agrégées et non mutables. En revanche il est de la responsabilité de l'utilisateur de la librairie, et qui désirerait tirer parti des optimisations impliquées par ce mécanisme, d'invoquer la méthode `declareAsNew` pour notifier à l'objet qu'il a été modifié. C'est par exemple le cas d'un objet de type `DataArrayDouble` dont on aurait modifié le contenu du tableau C/C++ agrégé. Tous les objets agrégeant directement ou indirectement cette instance de `DataArrayDouble` modifieront automatiquement leur état interne en conséquence après invocation de la méthode `updateTime`.

5.2. RefCountObject

Comme vu au chapitre précédent `MEDCoupling` a de manière native à gérer des objets volumineux que l'on souhaite pouvoir partager entre différents objets agrégeant, pour limiter au maximum les copies. Les cas d'utilisation les plus courants étant le partage des coordonnés entre plusieurs maillages pour avoir une représentation en cellule et en face d'un même maillage. L'autre cas est celui qui consiste à fabriquer un maillage à partir d'un autre par la seule modification de ses coordonnées et en partageant la connectivité. Comme conséquence directe la classe `RefCountObject` possède des constructeurs protégés pour interdire l'instanciation en pile des instances de `RefCountObject` et malgré tout garder l'appel possible depuis ses classes filles. En effet laisser possible une allocation en pile entrainerait une violation mémoire à l'appel de `decrRef` avec le compteur à 1. De la même manière les classes filles de `RefCountObject` auront des constructeurs privés pour la même raison. L'instanciation de toutes les classes filles se fera obligatoirement grâce aux méthodes statiques `New`. Les sous classes les plus illustres que l'on verra au §5.3 et au §5.4 plus en détail sont : `MEDCouplingMesh`, `DataArrayDouble`, `MEDCouplingFieldDouble`.

5.3. DIAGRAMME DES CLASSES MAILLAGES



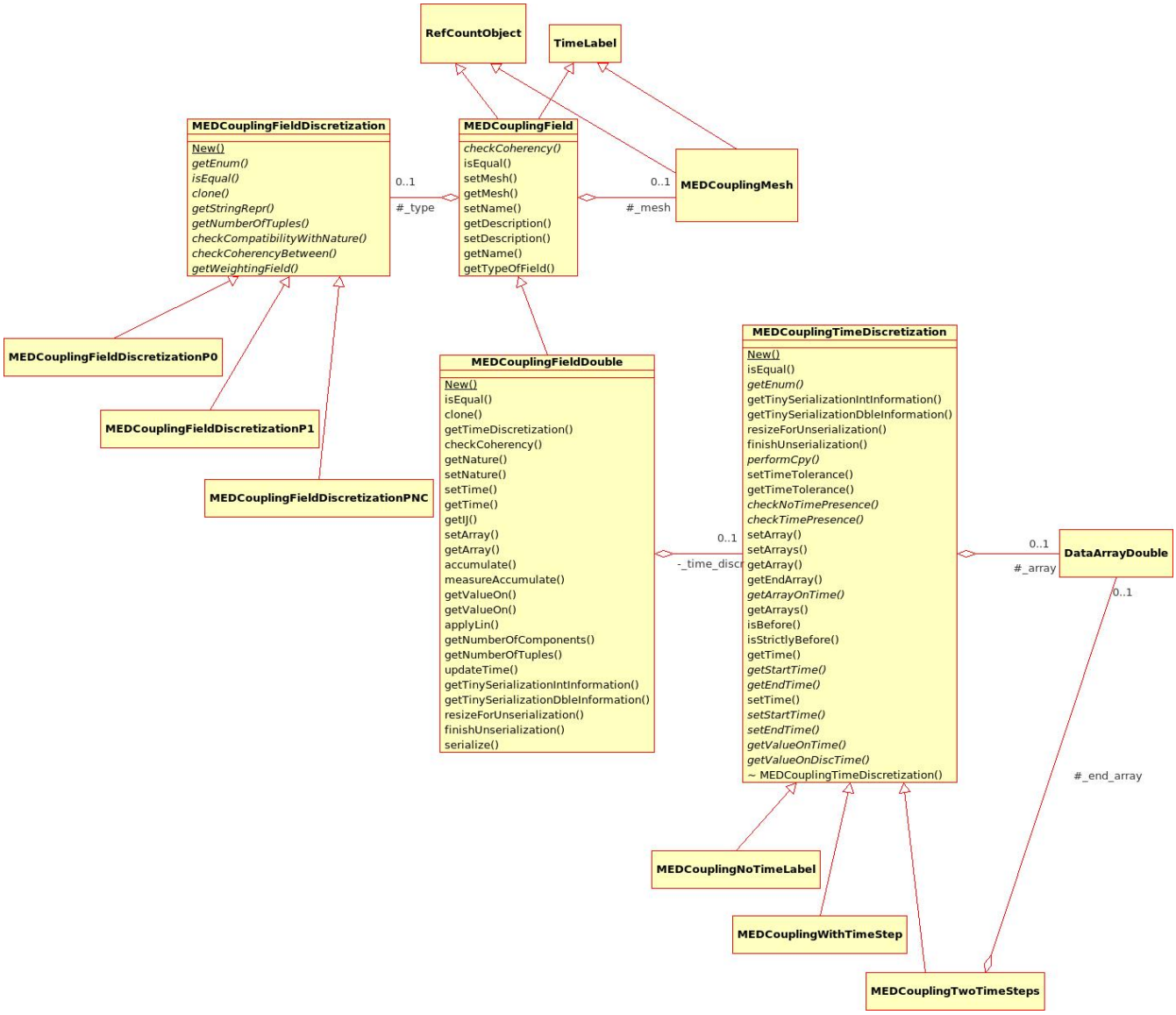
La classe des maillages non-structurés agrège soit :

- 3 tableaux, 1 de flottant et 2 d'entiers qui stockent respectivement les coordonnées, le tableau de connectivité nodale et un tableau d'index sur la connectivité nodale. Il s'agit de la class `MEDCouplingUMesh` ci-dessus.

- 5 tableaux, 1 de flottant et 4 d'entiers qui stockent respectivement les coordonnées, le tableau de connectivité descendante, un tableau d'index sur la connectivité descendante, un tableau de connectivité nodales des cellules de dimension n-1 et le tableau d'index correspondant. Il s'agit de la classe `MEDCouplingUMeshDesc` dans le diagramme de classe ci-dessus.

La classe `MEDCouplingPointSet` a été introduite pour inclure un maximum de comportements communs à tous les types de maillages non-structurés. On peut remarquer que cette classe définit des méthodes virtuelles pures de sérialisation, désérialisation qui seront utilisées pour transférer de process à process ces objets via MPI ou CORBA.

5.4. DIAGRAMME DES CLASSES CHAMPS



5.5. LES CLASSES DE DISCRETISATION

La classe `MEDCouplingFieldDiscretization` est l'interface qui définit ce qu'une discrétisation spatiale d'un champ doit prendre en charge. Une instance est construite par pattern factory au constructeur d'un champ grâce au premier paramètre enum. Ainsi cette classe, suivant la classe concrète instanciée aura la responsabilité de :

- Contrôler le nombre de tuples dans le champ agrégeant
- Construire une chaîne de caractères pour aiguiller vers le bon interpolateur

- Construire un champ de pondération permettant de contrôler suivant la discrétisation choisie de la conservation du champ après interpolation

- Participer à trouver les valeurs retournées par les méthodes `MEDCouplingFieldDouble::getValueOn` et `getValueOnNC`

La classe `MEDCouplingTimeDiscretization` est la classe qui gère la définition temporelle d'un champ. Cette classe de part sa nature agrège donc les tableaux de données au nom du champ sur lequel elle porte. La discrétisation en temps est soit sans label, pour les champs stationnaires, soit sur un temps ou alors linéaire en temps. De la même façon que pour la discrétisation spatiale on a recours à un pattern factory pour l'instanciation à la construction du champ grâce au deuxième paramètre. Cette classe a comme responsabilité :

- Contrôler la cohérence des requêtes en temps faite sur le champ agrégeant

- Donner la valeur du champ à un temps donné

6. Exemples d'utilisation

6.1. CREATION D'UN MAILLAGE

Voici l'exemple de création d'un maillage non structuré en nodal :

```
double   targetCoords[27]={-0.3,-0.3,0.5,   0.2,-0.3,1.,   0.7,-0.3,1.5,   -0.3,0.2,0.5, 0.2,0.2,1., 0.7,0.2,1.5, -0.3,0.7,0.5, 0.2,0.7,1., 0.7,0.7,1.5};

int targetConn[18]={0,3,4,1, 1,4,2, 4,5,2, 6,7,4,3, 7,8,5,4};

MEDCouplingUMesh *targetMesh=MEDCouplingUMesh::New("toto",2);

targetMesh->allocateCells(5);

targetMesh->insertNextCell(INTERP_KERNEL::NORM_QUAD4,4,targetConn);

targetMesh->insertNextCell(INTERP_KERNEL::NORM_TRI3,3,targetConn+4);

targetMesh->insertNextCell(INTERP_KERNEL::NORM_TRI3,3,targetConn+7);

targetMesh->insertNextCell(INTERP_KERNEL::NORM_QUAD4,4,targetConn+10);

targetMesh->insertNextCell(INTERP_KERNEL::NORM_QUAD4,4,targetConn+14);

targetMesh->finishInsertingCells();

DataArrayDouble *myCoords=DataArrayDouble::New();

myCoords->alloc(9,3);//On alloue 9 tuples constitués chacuns de 3 éléments

//1er mode copie des elements avec std ::copy
```

```
std::copy(targetCoords,targetCoords+27,myCoords->getPointer());
// 2eme mode : pas de copie et pas d'ownership
myCoords->useArray(targetCoords,false,CPP_DEALLOC,9,3);
//
targetMesh->setCoords(myCoords);//myCoords a un CR égal à 2
myCoords->decrRef();//myCoords n'est plus utilise dans ce scope CR=1
```

A ce stade, le maillage `targetMesh` est utilisable avec un compteur de référence à 1.

6.2. CREATION D'UN CHAMP STATIONNAIRE SANS TEMPS

```
MEDCouplingFieldDouble
*fieldOnCells=MEDCouplingFieldDouble::New(ON_CELLS,NO_TIME);//CR=1 au New
fieldOnCells->setMesh(targetMesh);//CR=2 car fieldOnCells a une ref dessus
DataArrayDouble *array=DataArrayDouble::New();
// field with 9 elements per tuple
array->alloc(nbOfCells,9);
fieldOnCells->setArray(array);
tmp=array->getPointer();
std::fill(tmp,tmp+9*nbOfCells,7.);
array->decrRef();
fieldOnCells->checkCoherency();
```

6.3. CREATION D'UN CHAMP STATIONNAIRE AVEC TEMPS

```
MEDCouplingFieldDouble
*fieldOnCells=MEDCouplingFieldDouble::New(ON_CELLS,ONE_TIME);
fieldOnCells->setMesh(targetMesh);
DataArrayDouble *array=DataArrayDouble::New();
// field with 9 elements per tuple
array->alloc(nbOfCells,9);
fieldOnCells->setArray(array);
```

```
tmp=array->getPointer();

array->decrRef();

fill(tmp,tmp+9*nbOfCells,7.);

fieldOnCells->setTime(2.3,4,1);

fieldOnCells->checkCoherency();
```

7. Liens avec Salome

A la suite de l'implémentation de structures MEDCoupling normalisant les maillages et les champs pour le couplage le module MED doit offrir un moyen de transfert entre process. Le module MED devra proposer le transfert de ces structures :

- Par CORBA pour le couplage DataFlow avec supervision en fournissant des servants CORBA. Des classes de chargement local, MEDCouplingClient, devront aussi être développées pour permettre un interfaçage avec HXX2SALOME.

- Par MPI pour le couplage direct avec lien à MEDCoupling

Le module MED doit aussi fournir des méthodes de lecture/écriture de fichiers MED vers les structures normalisées de MEDCoupling. Il est à noter que les notions de FAMILLE et de GROUPE dans MED fichier n'existent pas dans MEDCoupling. Cela entraîne une approche différente du chargement à partir d'un fichier MED par rapport à MEDMEM. Les contraintes de minimisation de dépendances de pré-requis vues au §2, font que ces méthodes de lecture/écritures seront dans un module séparé.

8. Liens avec ICoCo

L'objet d'ICoCo est de définir une interface de couplage de codes [2]. Celle-ci repose en particulier sur une classe `ICoco::Field` permettant de décrire un champ. La classe `ICoco::Field` n'a pas été spécifiée dans ICoCo. Elle est censée permettre de décrire les valeurs d'un champ en fonction de l'espace et du temps, et de contenir toutes les informations nécessaires dans ce but (maillage(s), discrétisation(s),...). La classe MEDCouplingField, une fois testée et validée sur des cas d'utilisation courants, a vocation à être adoptée comme classe des champs échangés via ICoCo. Elle introduit cependant certaines restrictions que l'on va expliciter ci-dessous.

8.1. Comparaison des champs ICoco et MEDCoupling

Comme sité au §2.2 de [2] un champ ICoco au sens strict peut être considéré comme une fonction qui au tuple (x,y,z,t) associe un tuple qui est la valeur du champ et ce sur un domaine limité de l'espace. Le champ n'étant défini que sur un nombre fini de valeurs, ce champ sera décrit par des valeurs discrètes sur un support spatio-temporel et une discrétisation spatiale et éventuellement un autre temporelle pour pouvoir calculer les valeurs de ce champ ailleurs qu'aux entités de son support.

Pour pouvoir couvrir au maximum tout le spectre des utilisations de la notion de champ que cette définition implique, MEDCoupling à introduit la notion de discrétisation spatiale (`MEDCouplingFieldDiscretization`) et temporelle (`MEDCouplingTimeDiscretization`). Ces

discrétisations permettent d'associer à des valeurs reposant sur un support discret de valeurs en tous points de l'enveloppe de ce support. Ceci constitue un ajout par rapport à ce qui existe dans MED et MEDMEM.

Néanmoins une différence subsiste entre la notion de champ MEDCoupling et celle d'ICoco : ICoco considère la variable temps à égalité avec les variables d'espace, ce qui implique que dans ce modèle le support en temps du champ est considéré sur le même plan que celui d'espace. Or informatiquement parlant ces supports en espace ont de manière générale des occupations mémoires très supérieur à celui en temps. Ceci a entrainé à considérer le champ MEDCoupling comme reposant sur un maillage invariant sur l'enveloppe temporelle de définition de ce champ. Cela est déjà le cas de la norme des fichiers MED et de MEDMEM. De cela découle que la structure `MEDCouplingField` n'est pas suffisante, à elle seule, pour décrire un champ reposant sur un maillage mobile au cours de son temps de définition. En revanche une collection d'instances de `MEDCouplingField` suffit à couvrir la notion d'ICoco.

9. Références

[1] : Notes on conservative remappers and their parallel implementation. Michael Ndjinga.

[2] : Proposition d'une interface de couplage en vue des couplages de codes dans Neptune DER/SSTH/LMDL/2006-010