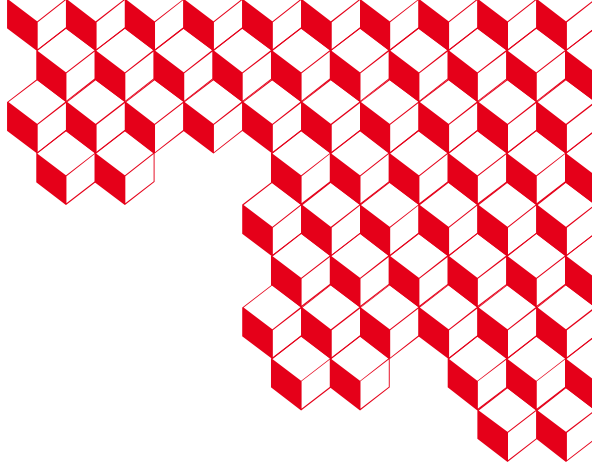


Rencontres Arcane

4ème édition

Introduction à Arcane

Lundi 24 mars 2025

1. Présentation

Présentation

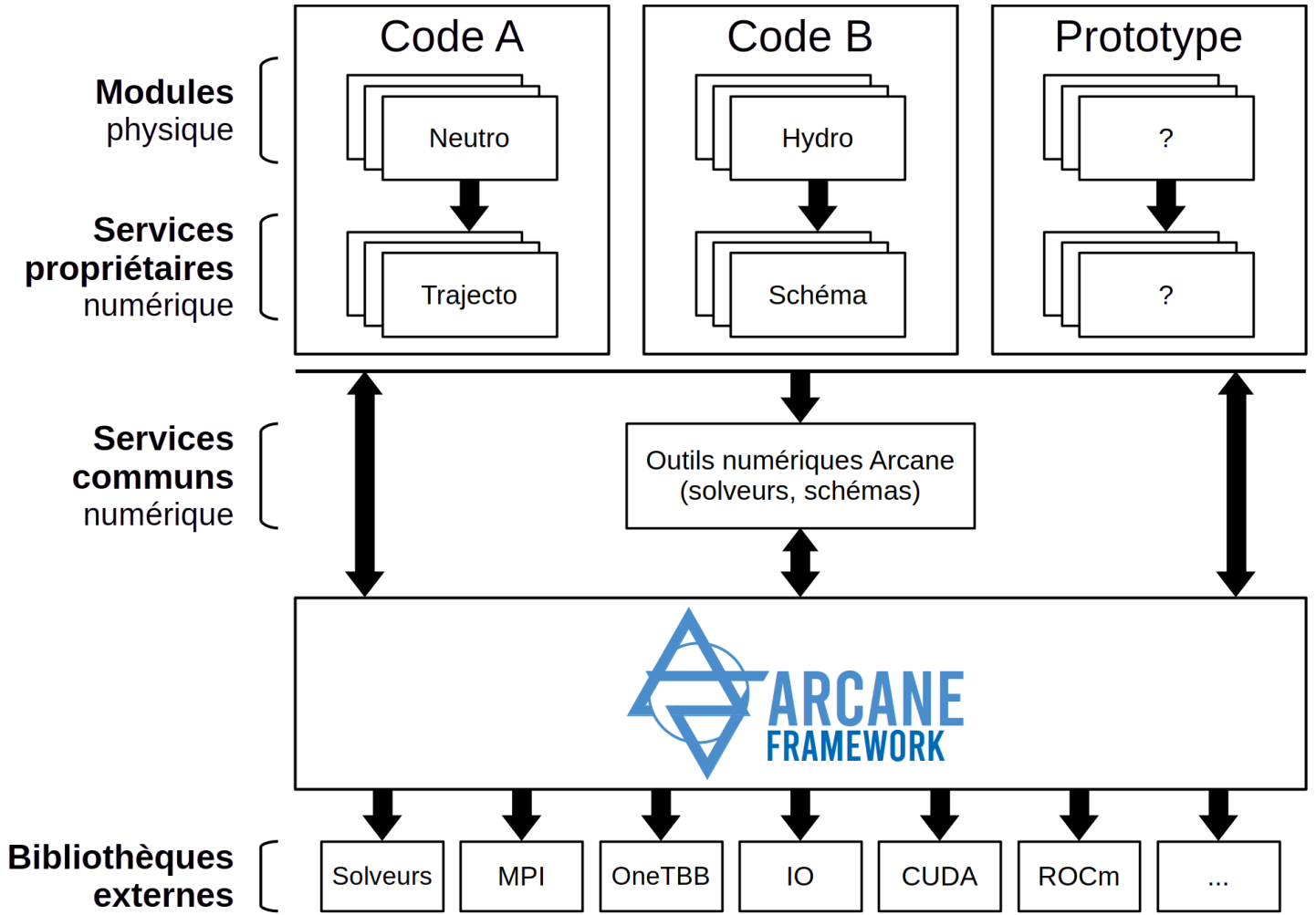
- Arcane est une plate-forme de développement pour les codes de simulations volumes/éléments finis 2D/3D parallèles
- Principes de base
  - simplifier au maximum l'écriture des modules numériques et d'environnement en prenant en charge les aspects informatiques
  - obtenir de bonnes performances sur machines parallèles (grappes de SMP)
  - développer rapidement: outils de mise au point, temps de compilation réduits, simplicité de mise en œuvre.
- La plateforme Arcane est développée au CEA depuis 2000
  - Bibliothèque C++
  - Environ 460 000 lignes de code C++ (et 43 000 lignes de code C#)*
  - Support Linux et Windows
- Collaboration avec IFPEN (www.ifpenergiesnouvelles.com) depuis 2007
  - Utilisation pour des codes de recherche en géophysique
  - Utilisation commerciale
- Disponible en open source sur github.com sous licence Apache2.

*Utilisation de l'outil `cloc` (github.com/AlDanial/cloc)

Historique

- 2007 : Début collaboration avec IFPEN sur Arcane
- 2012 : Début collaboration avec IFPEN sur une bibliothèque pour gérer l'algèbre linéaire.
  - Ces développements se basent sur une bibliothèque développée à l'IFPEN : Alien
  - Cette bibliothèque peut s'utiliser avec Arcane (pour les codes basés sur Arcane) ou sans Arcane (pour les codes historiques de l'IFPEN n'utilisant pas Arcane)
- 2016 : Décision de mettre en Open Source Alien
  - Cela nécessite de mettre en Open Source la partie de Arcane utilisée par Alien. Cette partie extraite de Arcane a trois composantes
    - arccon : gestion de la configuration
    - axlstar : gestion des descripteurs de modules/services et du jeu de données utilisateurs.
    - arccore: classes de Arcane utilisées par Alien (classes de base, tableaux, traces, échange de message, ...)
- 2019: Décision de mettre en Open Source Arcane
- 2021: Première version ouverte disponible sur 'gitlab.com' puis passage sur 'github.com'.

Schéma général

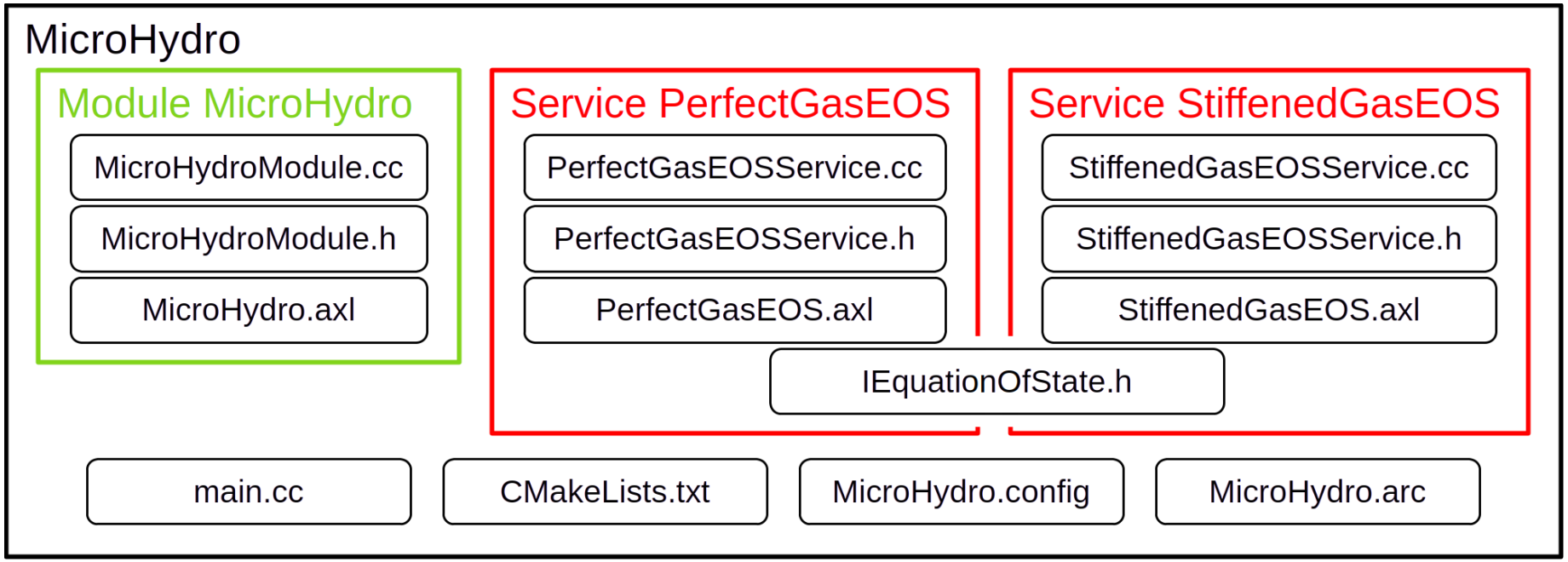


Fonctionnalités disponibles dans Arcane

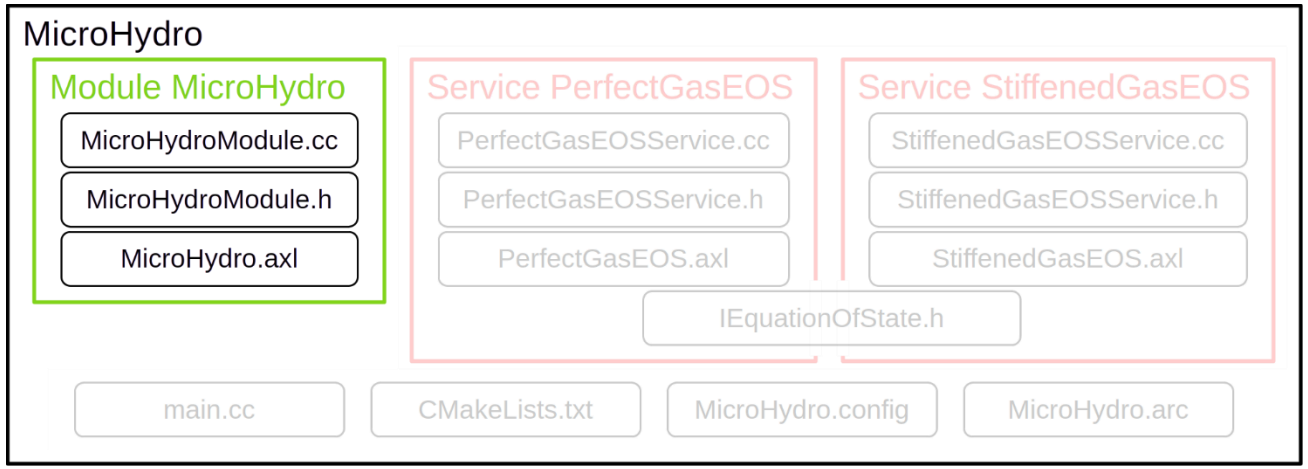
Arcane prend en charge les services suivants :

- Gestion des structures liées au maillage
- Gestion des variables (Température, Pression, …) à travers une base de données
- Parallélisme
- Gestion des modules et de leurs interactions
- Options de configuration des modules (jeu de données)
- Protections / reprises
- Fourniture de fonctions utilitaires (mathématiques, listing, temps d'exécution, …)
- Outils d'analyse et d'aide à la mise au point
- Retour-arrière
- Sorties spécifiques de dépouillement
  - courbes
  - historiques

Exemple de code

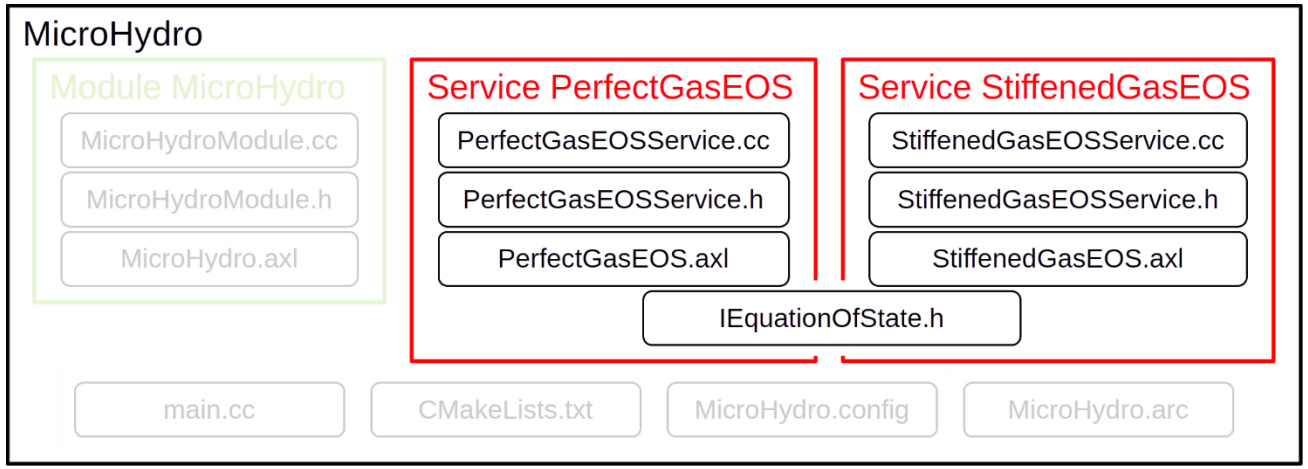


Notion de module



- Le code est constitué d'un ensemble de **modules**
  - hydrodynamique
  - photonique
  - post-traitement
  - …
- Chaque module est indépendant et possède
  - ses **variables**, qui représentent les données traitées par le module (ex: température, vitesse, Δt)
  - ses **options de configuration** (jeu de données)
  - ses **points d'entrée**, qui représentent les opérations effectuées par le module (ex: calcul des forces, déplacement des nœuds)
- Les interfaces entre les modules sont gérées par Arcane
- **Les modules communiquent via les variables:**
  - deux modules utilisant une variable publique de même nom partagent sa valeur

Notion de service



- Un service est un plug-in implémentant une interface
  - Service Technique
    - lecture et structure du maillage
    - sorties pour le dépouillement
    - algorithme de repartitionnement
  - Service Fonctionnel
    - vecteur, matrice, solveur
    - trajectographie
    - schéma numérique

- Caractéristiques
  - Utilise le modèle de conception « Fabrique »
  - Configurable de la même manière qu'un module, mais sans point d'entrée

Exemple de services

- La plupart des fonctionnalités d'Arcane sont fournies via cette notion de service, par exemple :
  - Maillage (IMesh,IItemFamily)
    - Lima, VTK, XDMF, GMSH
  - Parallélisme (IParallelMng)
    - MPI, Threads
  - Post-traitement (IPostProcessorWriter)
    - Ensight, Hercule, VtkHdf
  - Partitionnement dynamique (IMeshPartitionner)
    - Parmetis, Zoltan, SplitSD, Scotch
  - Opérateur de discrétisation (IDiscreteOperator)
    - Schéma O, 2 points
  - Solveurs algébriques
    - Arcane, Hypre, Sloop
- Il est possible de changer une fonctionnalité en réimplémentant certains services.

Jeu de données

Informations à propos du jeu de données

Configuration du post-processing

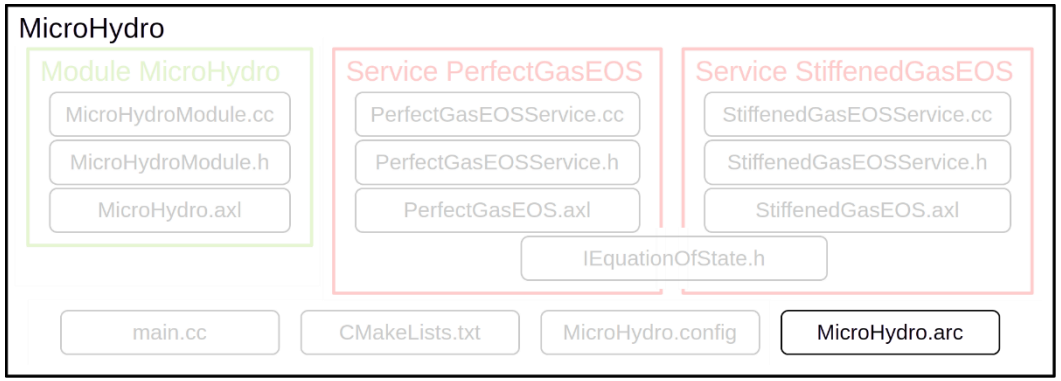
Maillage(s) à lire ou à générer

Définition des valeurs des options du module « MicroHydro »

```
<?xml version='1.0'?>
<case codeversion="1.0" codename="MicroHydro">
  <arcane>
    <title>Exemple</title>
    <description>Exemple Arcane d'un module Hydro très, très simplifié</description>
    <timeloop>MicroHydroLoop</timeloop>
  </arcane>

  <arcane-post-processing>
    <output-period>10</output-period>
    <output>
      <variable>CellMass</variable>
      ...
    </output>
  </arcane-post-processing>

  <meshes>
    <mesh>
      <filename internal-partition='true'>sod.vtk</filename>
    </mesh>
  </meshes>

  <micro-hydro>
    <deltat-init>0.001</deltat-init>
    <deltat-min>0.00001</deltat-min>
    <deltat-max>0.0001</deltat-max>
    <final-time>0.05</final-time>

    <eos-model name="StiffenedGas">
      <limit-tension>0.01</limit-tension>
    </eos-model>

    <boundary-condition>
      <surface>XMIN</surface>
      <type>Vx</type>
      <value>0.</value>
    </boundary-condition>
    ...
  </micro-hydro>
</case>
```

Choix du service EOS à utiliser et définition des options de ce service

Option complexe avec plusieurs options simples



Fichier de configuration

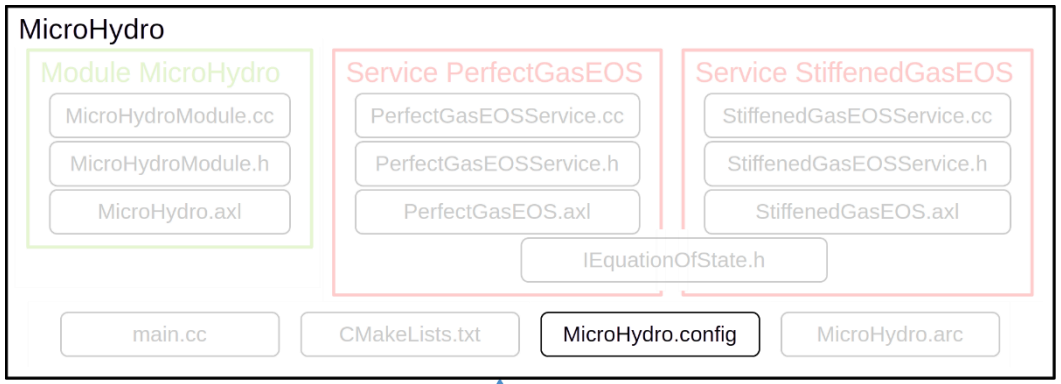
Informations à propos de la boucle en temps

Modules qui seront utilisés dans cette boucle

Points d'entrée à lancer à l'initialisation

Points d'entrée à lancer à chaque pas de temps

```
<?xml version="1.0"?>
<arcane-config code-name="MicroHydro">
  <time-loops>
    <time-loop name="MicroHydroLoop">
      <title>MicroHydro</title>
      <description>Boucle en temps de l'exemple Arcane MicroHydro</description>

      <modules>
        <module name="MicroHydro" need="required"/>
        <module name="ArcanePostProcessing" need="required"/>
      </modules>

      <entry-points where="init">
        <entry-point name="MicroHydro.HydroStartInit"/>
      </entry-points>

      <entry-points where="compute-loop">
        <entry-point name="MicroHydro.ComputePressureForce"/>
        <entry-point name="MicroHydro.ComputeVelocity"/>
        ...
      </entry-points>
    </time-loop>
  </time-loops>
</arcane-config>
```



Lancement du code

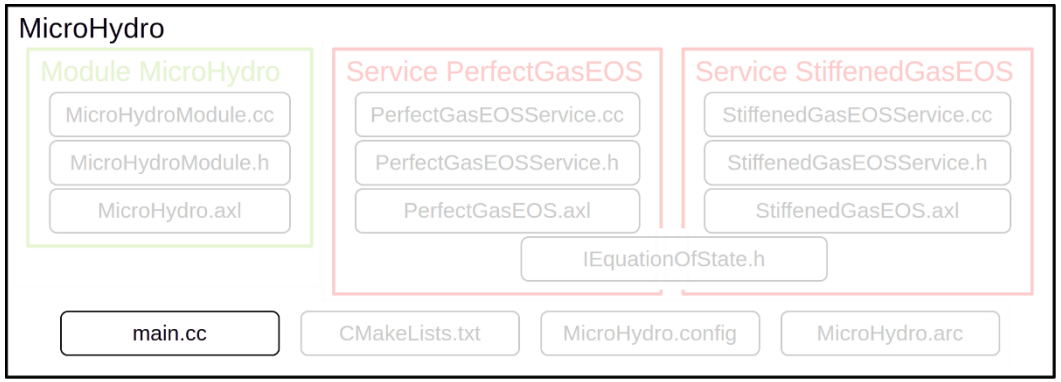
Initialisation d'Arcane (lecture des arguments, Init)

Ajout d'informations à propos du code

Affichage de l'aide générique Arcane

Lancement de la boucle en temps définie dans le jeu de données

```
#include <arcane/launcher/ArcaneLauncher.h>

using namespace Arcane;

int main(int argc,char* argv[])
{
  ArcaneLauncher::init(CommandLineArguments(&argc,&argv));

  auto& app_build_info = ArcaneLauncher::applicationBuildInfo();
  app_build_info.setCodeName("MicroHydro");
  app_build_info.setCodeVersion(VersionInfo(1,0,0));

  if(ArcaneLauncher::printHelp()){
    return 0;
  }

  return ArcaneLauncher::run();
}
```



Mode standalone

```cpp
#include <arcane/launcher/ArcaneLauncher.h>

#include <arcane/utils/ITraceMng.h>
#include <arcane/utils/FatalErrorException.h>
#include <arcane/utils/Real3.h>

#include <arcane/core/MeshReaderMng.h>
#include <arcane/core/IMesh.h>
#include <arcane/core/ISubDomain.h>
#include <arcane/core/IParallelMng.h>
#include <arcane/core/ItemGroup.h>
#include <arcane/core/VariableTypes.h>

#include <arcane/utils/Exception.h>

#include <iostream>


using namespace Arcane;

void executeSample(const String& case_file);

int main(int argc, char* argv[])
{
  String case_file;

  auto func = [&] {
    std::cout << "Sample: StandaloneSubDomain\n";

    CommandLineArguments cmd_line_args(&argc, &argv);
    ArcaneLauncher::init(cmd_line_args);
    if (argc > 1)
      case_file = argv[argc - 1];
    executeSample(case_file);
  };

  return arcaneCallFunctionAndCatchException(func);
}
```

```cpp
void executeSample(const String& case_file)
{
  StandaloneSubDomain launcher{ ArcaneLauncher::createStandaloneSubDomain(case_file) };
  ISubDomain* sd = launcher.subDomain();

  ITraceMng* tm = launcher.traceMng();

  MeshReaderMng mrm(sd);

  IMesh* mesh = mrm.readMesh("AdditionalMesh", "plancher.msh", sd->parallelMng());

  Int32 nb_cell = mesh->nbCell();
  tm->info() << "NB_CELL=" << nb_cell;

  VariableNodeReal3& nodes_coordinates = mesh->nodesCoordinates();
  ENUMERATE_ (Cell, icell, mesh->allCells()) {
    Cell cell = *icell;
    Real3 cell_center;
    for (Node node : cell.nodes()) {
      cell_center += nodes_coordinates[node];
    }
    cell_center /= cell.nbNode();
    tm->info() << "Cell=" << cell.uniqueId() << " center=" << cell_center;
  }
}
```

2. Maillage

Maillage

- La classe interface `IMesh` permet d'accéder aux informations du maillage :
  - `#include <arcane/core/IMesh.h>`
- Un maillage par défaut est créé lors de l'initialisation du code.
- Le choix de ce maillage est fait dans le jeu de données.
- Il est possible de créer d'autres maillages, de dimension quelconque.
- Le maillage est dynamique (i.e: il est possible d'ajouter/supprimer des entités).
- Les modules et services accèdent automatiquement au maillage par défaut via leur classe de base (`BasicModule` ou `BasicService`).
- Un maillage est composé de familles d'entités (classe interface `IItemFamily`).
- Une famille d'entité gère des entités de même genre (nœud, face, maille, ...).

Entités du maillage

- La classe de base des entités du maillage est `Item` :
  - `#include <arcane/core/Item.h>`
- Les autres classes dérivent de `Item`.
- La classe `Item` (et ses dérivées) est juste un ‘handle’ sur les informations du maillage et peut donc s’utiliser par valeur.
- IMPORTANT : les évolutions du maillage invalident ces ‘handle’: il ne faut donc pas conserver d’instances de `Item` (et ses dérivés) entre deux évolutions de maillage.

Entité	Dimension
Item	0D, 1D, 2D ou 3D
Cell	Dimension du maillage (N)
Node	0D
Edge	1D (n’existe que en 3D)
Face	N-1
Particle	-
DoF	-

Entités du maillage

- Les méthodes suivantes sont disponibles pour toutes les entités du maillage :

Méthode	Type	Description
localId()	Int32	Numéro local au sous-domaine (peut évoluer)
uniqueId()	ItemUniqueId (Int64)	Numéro unique sur tout le maillage
owner()	Int32	
kind()	eItemKind	Genre de l'entité
isOwn()	bool	Indique si l'entité appartient à ce sous-domaine
isShared()	bool	Indique si l'entité est partagée avec d'autres sous-domaines

- Les uniqueId() restent constants durant toute la durée de vie de l'entité: ils peuvent donc servir pour conserver des informations entre deux évolutions du maillage.

Itération sur les entités du maillage

L'itération sur un groupe d'entités se fait via les macros préfixés par ENUMERATE_

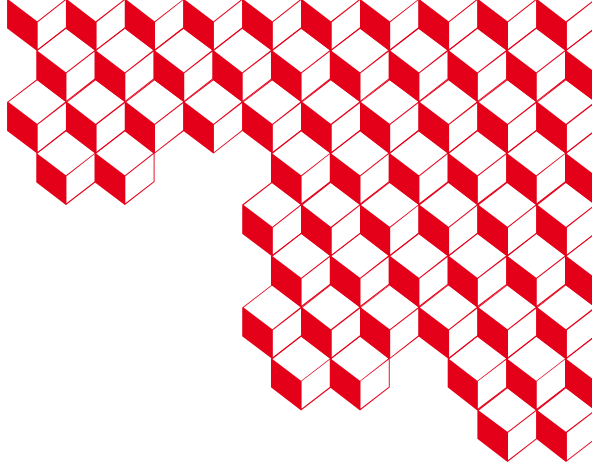
- ENUMERATE_(Node, « itérateur », « `NodeGroup` »)
- ENUMERATE_(Cell, « itérateur », « `CellGroup` »)
- ENUMERATE_(Face, « itérateur », « `FaceGroup` »)
- …

Ces macros prennent en argument le nom de l'itérateur et un `ItemGroup`.

L'itération sur une liste d'entités se fait via une boucle « for each » (C++11) :

- for(Node node : « `NodeVector` »)
- for(Cell cell : « `CellVector` »)
- for(Face face : « `FaceVector` »)
- …

```
// Calcul de la masse aux nœuds en 3D sur des hexaèdres
VariableNodeReal node_mass = ...;
VariableCellReal cell_mass = ...;
VariableCellReal cell_density = ...;
VariableCellReal cell_volume = ...;
CellGroup cells = ...
ENUMERATE_(Cell, icell, cells){
  cell_mass[icell] = cell_density[icell] * cell_volume[icell];
  Cell cell = *icell;
  Real contrib_node_mass = 0.125 * cell_mass[icell];
  for( Node node : cell.nodes()){
    m_node_mass[node] += contrib_node_mass;
  }
}
```



Merci

https://github.com/arcaneframework/framework
https://discord.gg/tqdu7U2UwH