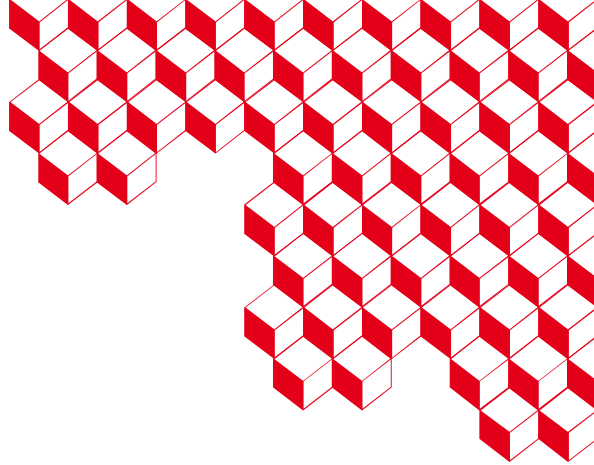


Modèles utilisateurs dans Phénix

Mise en place et exemples d'utilisations

Nicolas Peton CEA/DAM

Marie Spraul CEA/DAM

Rencontres Arcane 2023

17 avril 2023

Plan de la présentation

1. **Présentation générale des modèles utilisateurs**
2. **Point de vue développeur**
3. **Point de vue utilisateur**
4. **Conclusions**

1. Présentation générale des modèles utilisateurs

Qu’est-ce qu’un modèle utilisateur ?

- Définition générale :
  - Fonctionnalité permettant aux utilisateurs de mettre en œuvre leurs propres modèles et lois (non implémentés dans le code d’origine) au cours d’un calcul

- Objectifs :
  - Tester de nouvelles fonctionnalités
  - Explorer des configurations différentes

- Exemples de modèles utilisateurs :
  - Initialisation de grandeurs scalaires ou vectorielles
  - Utilisation de nouvelles équations d’états et lois de rhéologie
  - Prise en compte de conditions aux limites spécifiques
  - Définition de lois personnalisées (pression, vitesse…) dépendant du temps et/ou de l’espace
  - …

Principe de fonctionnement

- Un modèle utilisateur est assimilé à un service Arcane codé par les utilisateurs
- Base C# compatible avec Arcane
- Compilation à la volée des modèles au moment du lancement du calcul
- Interaction forte entre développeur et utilisateur

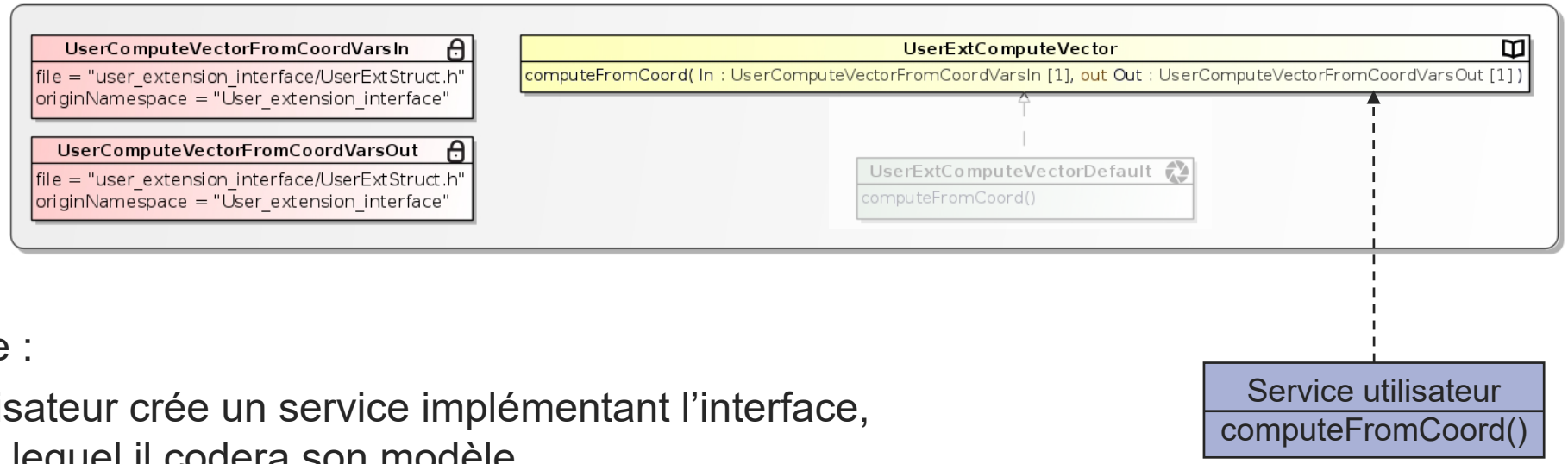
- Actions côté développeur :
  - Modélisation informatique du modèle utilisateur (interface, variables en entrée/sortie…)
  - Branchement dans le code existant
  - Mise en place d'un générateur de code pour les utilisateurs

- Actions côté utilisateur :
  - Complétion des fichiers générés automatiquement
  - Codage du modèle (langage C#) et définition des paramètres éventuels (fichier .axl)

2. Point de vue développeur

Modélisation

- Composition d'un modèle utilisateur :
  - Une interface
  - Un service « Default » associé (inutilisé en pratique)
  - Deux structures de données par méthode pour les variables en entrée et en sortie
- Conventions :
  - Le nom de l'interface débute par « UserExt »
  - L'ensemble des structures de données pour les variables est défini dans un unique fichier



- Principe :
  - L'utilisateur crée un service implémentant l'interface, dans lequel il codera son modèle
  - Ce service sera référencé dans le jeu de données

Modélisation

- Exemples de structures utilisées pour les variables en entrée et en sortie :

```
// ----------------------------------------------------------------------------
//! Struct de variables en entrée pour UserExtComputeVector::computeFromCoord
// ----------------------------------------------------------------------------
struct UserComputeVectorFromCoordVarsIn
{
  //! Constructeur par défaut
  UserComputeVectorFromCoordVarsIn(): coord(nullptr), time(0.0)
  {}

  //! Constructeur avec coordonnées
  UserComputeVectorFromCoordVarsIn(const StdReal3Vector* loc_coord): coord(loc_coord), time(0.0)
  {}

  //! Constructeur avec coordonnées et temps
  UserComputeVectorFromCoordVarsIn(const StdReal3Vector* loc_coord, const Real loc_time)
      : coord(loc_coord), time(loc_time)
  {}

  //! Coordonnées
  const StdReal3Vector* coord;

  //! Temps physique de la simulation
  const Real time;
};
```

Variables en entrée

```
// ----------------------------------------------------------------------------
//! Struct de variables en sortie pour UserExtComputeVector::computeFromCoord
// ----------------------------------------------------------------------------
struct UserComputeVectorFromCoordVarsOut
{
  //! Constructeur par défaut
  UserComputeVectorFromCoordVarsOut() = default;

  //! Constructeur
  UserComputeVectorFromCoordVarsOut(StdReal3Vector* loc_vitesse)
      : vitesse(loc_vitesse)
  {}

  //! Vitesse à initialiser
  StdReal3Vector* vitesse;
};
```

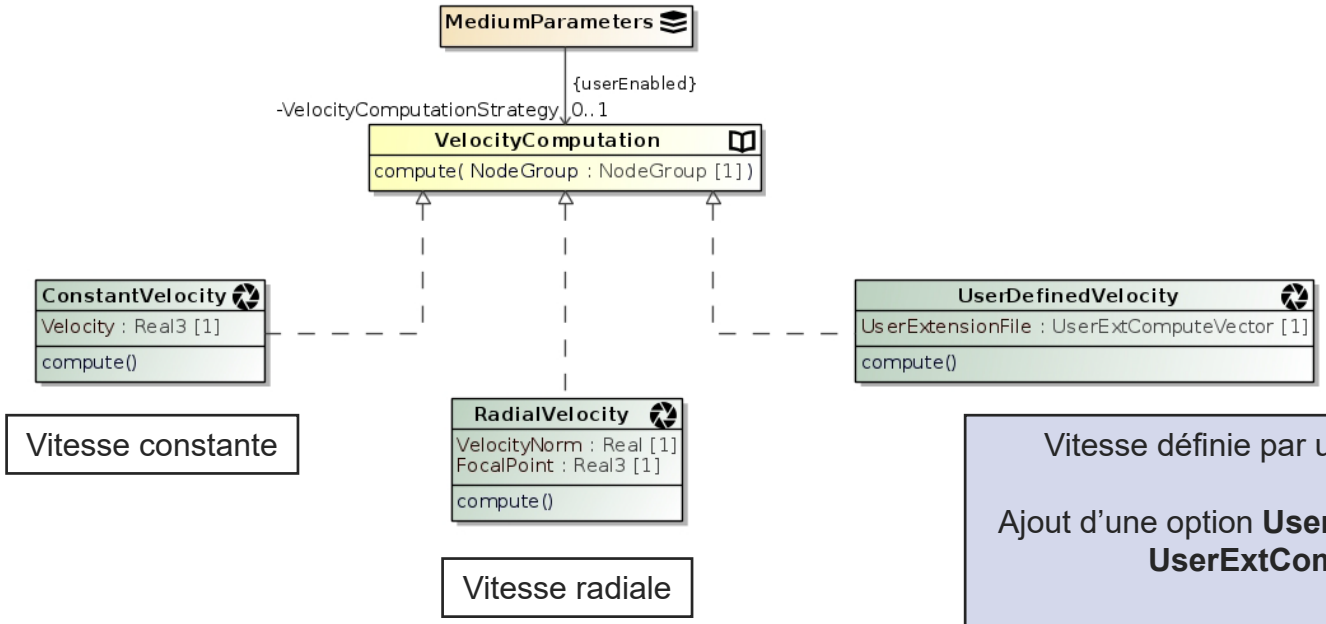
Variable en sortie

- Utilisation de structures de données simplifiées pour les utilisateurs (StdRealVector, StdReal3Vector…)

Branchement au reste du code

- Branchement des modèles utilisateurs dans Arcane comme une option classique
- Convention : le nom de l'option associée est **UserExtensionFile**

- Exemple : initialisation de la vitesse sur un milieu



Vitesse constante

Vitesse radiale

Vitesse définie par un modèle utilisateur

Ajout d'une option **UserExtensionFile** de type **UserExtComputeVector**

L'interface **UserExtComputeVector** sera implémentée par un service créé par l'utilisateur

Appel au modèle utilisateur dans le code

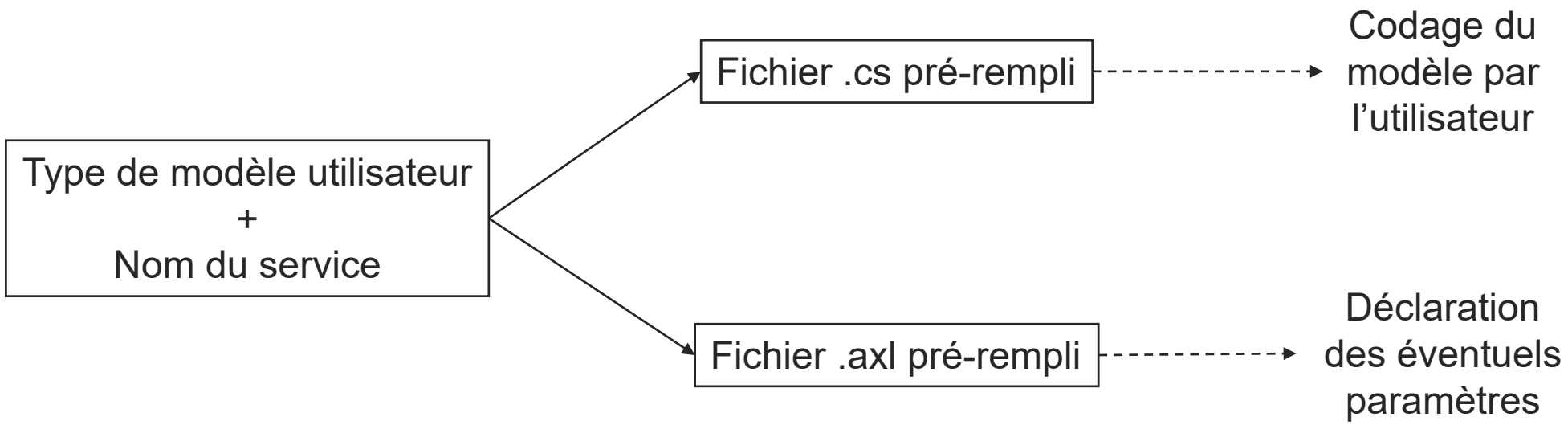
- Nécessité d'effectuer des conversions entre les structures de données simplifiées (StdRealVector, StdReal3Vector…) et les variables Arcane

```
void
UserDefinedVelocityService::_compute(const VariableNodeReal3& node_coord,
                                     const Real time,
                                     VariableNodeReal3& node_velocity,
                                     const NodeGroup node_group)
{
  // Construction de la structure de variables en entrée
  Commontypes::StdReal3Vector coord_vec;
  coord_vec.resize(node_group.size());
  ENUMERATE_NODE (node_i, node_group) {
    coord_vec[node_i.index()] = node_coord[node_i];
  }
  User_extension_interface::UserComputeVectorFromCoordVarsIn user_vars_in(&coord_vec, time);

  // Construction de la structure de variables en sortie
  Commontypes::StdReal3Vector output_value;
  output_value.resize(node_group.size());
  User_extension_interface::UserComputeVectorFromCoordVarsOut user_vars_out(&output_value);

  // Appel du modèle utilisateur
  getUserExtensionFile()->computeFromCoord(&user_vars_in, &user_vars_out);

  // Recopie des valeurs issues du modèle utilisateur dans la variable Arcane
  ENUMERATE_NODE (node_i, node_group) {
    node_velocity[node_i] = output_value[node_i.index()];
  }
}
```

- Idée générale :
  - Construction des structures de données pour les variables en entrée et en sortie
  - Appel de la méthode du modèle utilisateur spécifié dans le jeu de données (via l'option **UserExtensionFile**)
  - Recopie des valeurs retournées par le modèle utilisateur dans la variable Arcane

Aide aux utilisateurs

- Mise en place d'un script Python interactif pour la génération de code « modèles utilisateurs »
- A chaque modèle est associé un template .jinja



- Actions côté développeur :
  - Mettre à jour le script Python pour la bonne génération des fichiers .cs et .axl
  - En cas de nouveau modèle utilisateur, créer un nouveau template

3. Point de vue utilisateur

Structure des fichiers .cs

Première partie

```
using System;
using Arcane;
using user_extension_interface;
using Real = System.Double;
using Real3 = Arcane.Real3;

#if true
// TRES IMPORTANT: la classe doit être publique
[Arcane.Service("LoiVitesse",typeof(IUserExtComputeVector))]
public class LoiVitesse : ArcaneLoiVitesseObject
{
  #region Generated Stuff
  static  LoiVitesse ()
  {
    Console.WriteLine("STATIC INIT");
  }

  public LoiVitesse (ServiceBuildInfo bi) : base(bi)
  {
  }

  /// <summary>
  /// Return the name of the service
  /// </summary>
  public override string GetImplName()
  {
    return "LoiVitesse";
  }
  #endregion
```

En-tête du fichier (à ne pas modifier)

Seconde partie

En-tête de la fonction avec description des variables en entrée et en sortie

```
/// ------------------------------------------------------------------------------------------
/// <summary>
/// Computation of velocity from coordinates
/// </summary>
/// <param name="vars_in">
///   Input variables :
///   - coord (array Real3) : array of coordinates
/// </param>
/// <param name="vars_out">
///   Output variables :
///   - vitesse (array Real3) : array of computed velocity values
/// </param>
public override void ComputeFromCoord(UserComputeVectorFromCoordVarsIn vars_in,
                                       UserComputeVectorFromCoordVarsOut vars_out)
{
  /// Affichage :
  Console.WriteLine("Inside {0}", this);

  /// Commenter la ligne ci-dessous pour utiliser cette méthode
  throw new NotImplementedException("Modèle utilisateur : méthode ComputeFromCoord non implémentée");

  /// Insérer le code ici...
  /// Vous pouvez coder votre méthode à partir de la boucle ci-dessous en la décommentant
  /// int pb_size = vars_out.vitesse.Count;
  /// for (int i=0; i<pb_size; ++i) {
  ///   Real v_x = 0.0;
  ///   Real v_y = 0.0;
  ///   Real v_z = 0.0;
  ///   vars_out.vitesse[i] = new Real3(v_x, v_y, v_z);
  /// }
}

//-----------------------------------------------------------
}
```

Corps de la fonction dans laquelle l'utilisateur doit coder son modèle

Structure des fichiers .axl

```
<?xml version="1.0" encoding="utf-8" ?>
<service name="LoiVitesse" version="1.0" type="caseoption" parent-name="user_extension_interface.IUserExtComputeVector_WrapperService">
<description>Jeu de donnees du service LoiVitesse</description>
<interface name="user_extension_interface.IUserExtComputeVector" inherited='false'/>
<variables/>
<!-- OPTIONS DU SERVICE -->
<options>
  <!--<simple name="nom-option-1" type="real" optional="false">
    <description>Entrer la description du paramètre</description>
  </simple>-->
</options>
<!-- FIN OPTIONS DU SERVICE -->
```

Déclaration des éventuels paramètres utilisés dans le fichier .cs associé

- Les noms des fichiers .cs et .axl doivent être identiques
- Attention au format des noms des paramètres :
  - Fichier .axl : **nom-parametre**
  - Fichier .cs : **Options.NomParametre**
- Les valeurs des paramètres sont à renseigner dans le fichier .arc

Exemple de modèle utilisateur avec paramètres

```
<?xml version="1.0" encoding="utf-8" ?>
<service name="LoiVitesse" version="1.0" type="caseoption" parent-name="user_extension_interface.IUserExtComputeVector_WrapperService">
<description>Jeu de donnees du service LoiVitesse</description>
<interface name="user_extension_interface.IUserExtComputeVector" inherited='false'/>
<variables/>
<options>
  <simple name="x-focalisation" type="real" optional="false"/>
  <simple name="y-focalisation" type="real" optional="false"/>
  <simple name="norme-vitesse" type="real" optional="false"/>
</options>
</service>
```

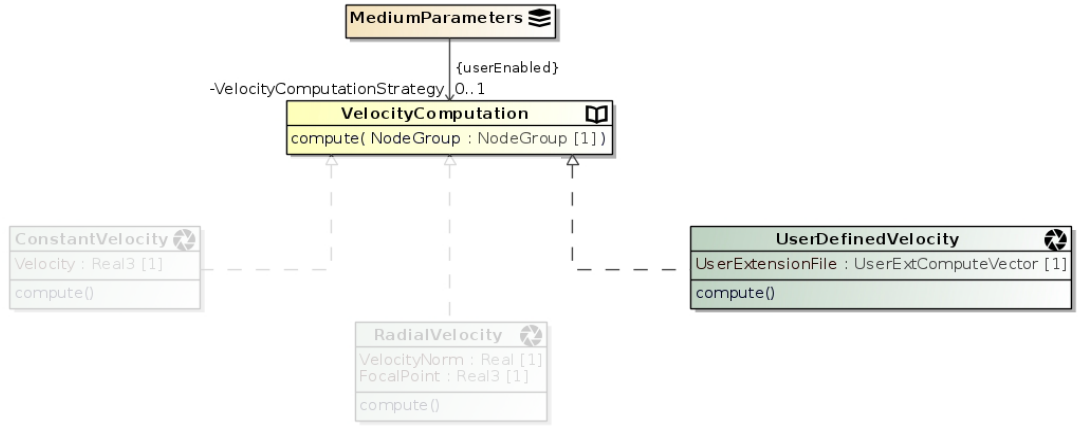
LoiVitesse.axl

```
<medium-parameters>
  <velocity-computation-strategy name="UserDefinedVelocity">
    <user-extension-file name="LoiVitesse">
      <x-focalisation>0.5</x-focalisation>
      <y-focalisation>0.5</y-focalisation>
      <norme-vitesse>2000.</norme-vitesse>
    </user-extension-file>
  </velocity-computation-strategy>
</medium-parameters>
```

Test.arc (extrait)

```
public override void ComputeFromCoord(UserComputeVectorFromCoordVarsIn vars_in,
                                       UserComputeVectorFromCoordVarsOut vars_out)
  {
     int pb_size = vars_out.vitesse.Count;
     for (int i=0; i<pb_size; ++i) {
         Real3 coord = vars_in.coord[i];
         Real3 focal_point = new Real3(Options.XFocalisation, Options.YFocalisation, 0.0);
         Real3 direction = focal_point - coord;
         Real norm = Math.Sqrt(direction.x * direction.x + direction.y * direction.y);

         if (norm > 1.0e-16) {
            Real3 velocity = (direction / norm) * Options.NormeVitesse;
            vars_out.vitesse[i] = velocity;
         } else {
            vars_out.vitesse[i] = new Real3(0.0, 0.0, 0.0);
         }
     }
  }
```

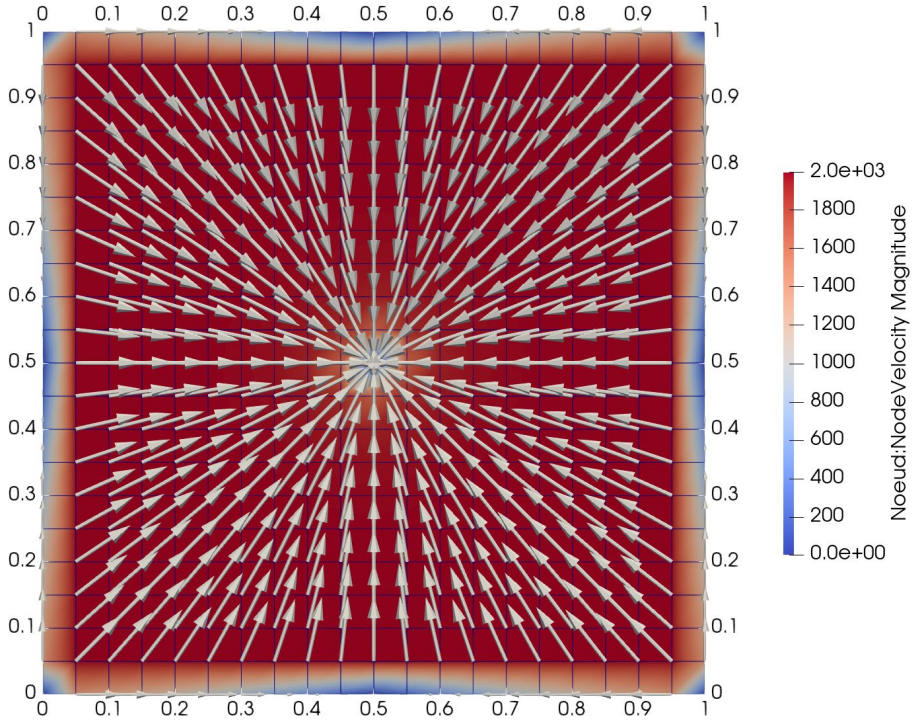
LoiVitesse.cs (extrait)



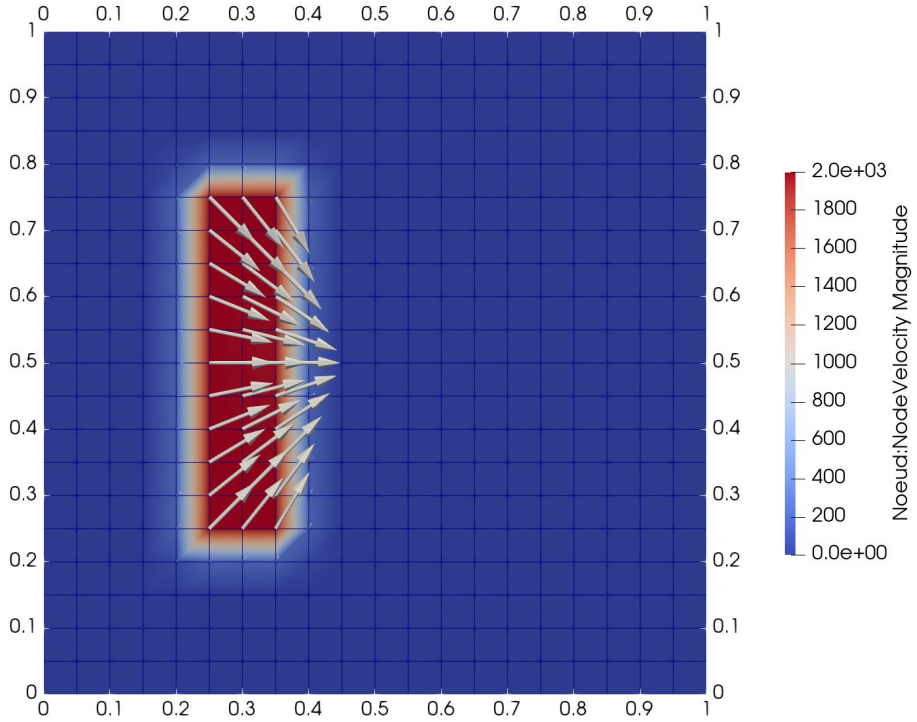
Compilation du modèle utilisateur à la volée au moment de l'exécution du code

Application : initialisation d'une vitesse

- Vitesse radiale imposée :
  - Centre de focalisation : (0.5, 0.5)
  - Norme de la vitesse : 2000 m/s



Vitesse radiale imposée
sur l'ensemble du domaine



Vitesse radiale imposée
sur un unique milieu

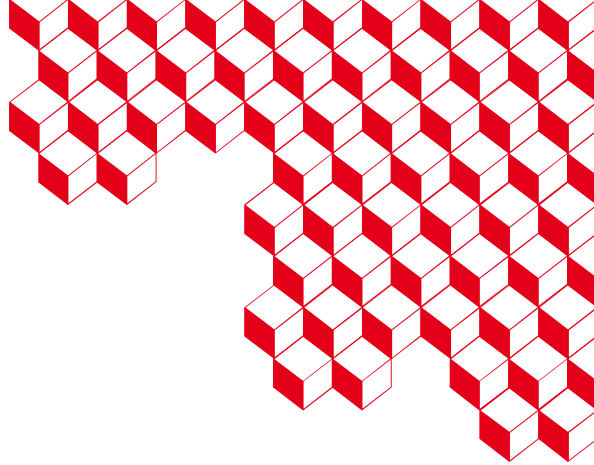
4. Conclusions

Conclusions

- Les modèles utilisateurs peuvent être employés dans de nombreuses configurations :
  - Initialisation de grandeurs
  - Utilisation d'équations d'états et de lois de comportements
  - Prise en compte de conditions aux limites
  - Définition de lois (pression, vitesse…) dépendant du temps et de l'espace
  - …

- Actions côté développeur :
  - Construction du modèle utilisateur (interface, variables en entrée/sortie…)
  - Branchement au reste du code comme une option classique
  - Mise en place de générateur de code pour les utilisateurs

- Utilisation simplifiée côté utilisateurs :
  - Emploi de scripts permettant de générer les squelettes des fichiers à remplir
  - Codage en C# avec des structures de données simplifiées



Merci de votre attention