OpenFLUID
Software Environment
for Spatial Modelling in Landscapes

TP6 : Utilisation de la connectivité entre unités spatiales

Objectifs:	Utiliser les relations de connectivité entre unités spatiales pour obtenir des informations -variables, attributs, ...- sur les unités connexes
Pré-requis:	TP3,TP4,TP5

Fonctionnalités:	Déclaration d'attribut requis
	`openfluid::core::IDIntMap`
	`openfluid::core::SpatialUnit::fromSpatialUnits()`
	`OPENFLUID_UNITSLIST_LOOP()`
	`openfluid::core::UnitsPtrList_t`
	`openfluid::core::Duration_t`

Nous allons créer un nouveau simulateur (`training.all.transfer`), qui calcule le débit en sortie de chaque unité spatiale des classes SU et RS. Il utilisera les variables produites par le simulateur `training.su.prod`, produira le débit d'eau pour chaque exutoire d'unité spatiale, et se basera sur les connexions entre ces unités spatiales pour assurer le transfert de cette eau à travers la zone étudiée.

1 Informations préliminaires

Le débit en sortie prend en compte le temps de transfert au travers de chaque unité spatiale de classe SU ou RS.

Soit d la distance de transfert, t le temps de transfert, v la vitesse de transfert

$$v = \frac{d}{t}$$

$$t = \frac{d}{v}$$

Soit Δt la durée d'un pas de temps, n le nombre de pas de temps pour le transfert sur une unité donnée

$$n.\Delta t = \frac{d}{v}$$

$$n = \frac{d}{v.\Delta t}$$

Avec v équivalent à la racine carrée de la pente p de chaque unité, multipliée par un coefficient global c, on obtient

$$n = \frac{d}{c.\sqrt{p}.\Delta t}$$

Le calcul du débit en sortie Q d'une unité spatiale sera égale à la somme des apports des unités amont, auquel on ajoute le débit local. Dans le cadre de cet exercice, seules les unités de types SU génèrent un débit local.

Soit Q le débit local en sortie d'une unité donnée de type SU, A l'aire de cette unité, H la hauteur de ruissellement, Δt la durée d'un pas de temps

$$Q = \frac{H.A}{\Delta t}$$

Note: Les hypothèses simplificatrices suivantes sont utilisées au cours de ce TP:

- les fossés (RS) ne débordent pas
- le transfert sur les parcelles (SU) est prioritaire sur les transferts sur les fossés (RS)
- les apports amonts ne sont pas pris en compte dans le calcul du partage ruissellement-infiltration sur les parcelles (SU)

2 Nouveau projet OpenFLUID

Afin de repartir du TP précédent, nous allons tout d'abord créer un projet OpenFLUID nommé TP6 (dans `<Bureau>/formation/projects/TP6`) et y importer le jeu de données d'entrée du TP5.

3 Code source

3.1 Génération du simulateur

Nous allons créer un nouveau simulateur à l'aide de l'interface de développement intégrée à OpenFLUID-Builder. Ce simulateur devra avoir comme caractéristiques, à renseigner dans la fenêtre de signature :

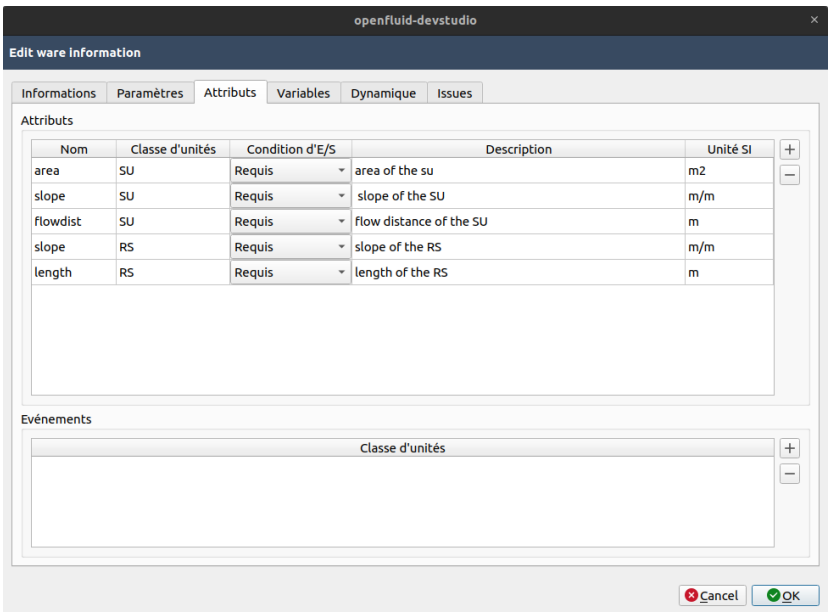
- ID : `training.all.transfer`
- Classe du simulateur : `TransferSimulator`
- Planification : `La planification utilise le DeltaT par défaut` (Onglet *Dynamique*)

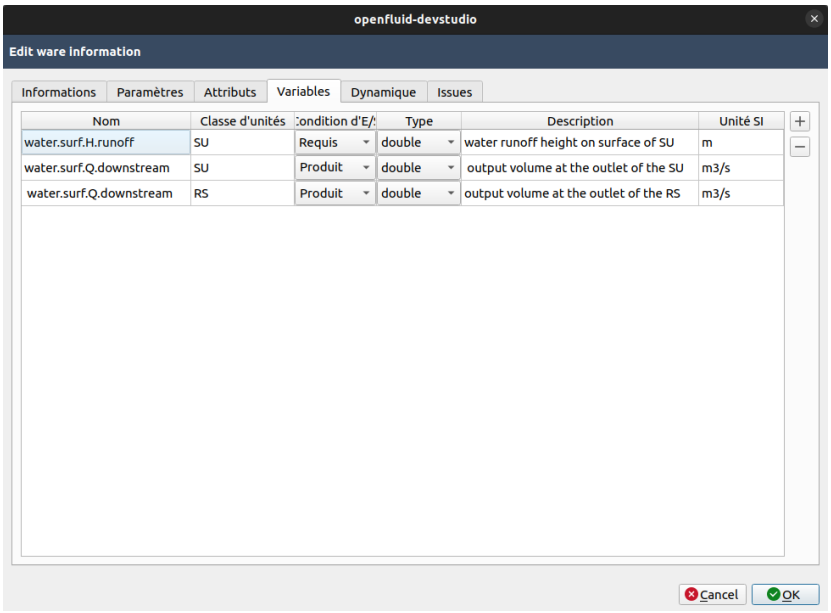
3.2 Signature

Ce simulateur génèrera des valeurs de débit à partir des débits en amont, ainsi que du ruissellement produit sur les unités de type SU. Nous allons donc déclarer la prise en compte de ce ruissellement (variable `water.surf.H.runoff`).
Ce calcul du débit et son transfert nécessitent la prise en compte de propriétés physiques (surface, pente et distance d'écoulement des SU, longueur et pente des RS) qui seront fournies sous forme d'attributs:

- `area` pour l'aire des SU
- `slope` pour la pente des SU ou des RS
- `flowdist` pour la distance d'écoulement des SU
- `length` pour la longueur des RS

Une fois complétée, la signature devrait être similaire à :





3.3 Attributs privés et constructeur

Le nombre de pas de temps de transfert étant lié à la pente, au coefficient global c et à la distance d'écoulement (invariants tout au long de la simulation), nous allons donc les stocker sous la forme d'un attribut privé, et les calculer une fois pour toutes dans la partie `prepareData()`.

Pour les stocker, nous utiliserons une structure de données de type `openfluid::core::IDIntMap` permettant de stocker une valeur de type entier pour chacune des unités spatiales concernées.

Les attributs privés comprennent également le coefficient global c pour les unités de classe SU et RS.

Une fois complétés, les attributs privés devraient être similaires à:

```
private:
  openfluid::core::IDIntMap m_SUTransferSteps;
  openfluid::core::IDIntMap m_RSTransferSteps;
  double m_SUMeanCelerity;
  double m_RSMeanCelerity;
```

Les attributs privés `m_SUMeanCelerity` et `m_RSMeanCelerity` seront initialisés dans le constructeur du simulateur, respectivement avec les valeurs 0.18 et 0.7

Une fois complété, le constructeur devrait être similaire à:

```
TransferSimulator(): PluggableSimulator(),
  m_SUMeanCelerity(0.18), m_RSMeanCelerity(0.7)
{

}
```

3.4 prepareData()

Pour calculer le nombre de pas de temps de transfert, nous allons utiliser deux boucles spatiales, une sur les SU et une autre sur les RS. Afin d'obtenir des nombres de pas de temps entiers, nous arrondirons les calculs au nombre entier supérieur (instruction `std::ceil()`).

Une fois complétée, la méthode `prepareData()` devrait être similaire à:

```
void prepareData()
{
  openfluid::core::DoubleValue Slope, Flowdist, Length;
  openfluid::core::SpatialUnit* pUnit;

  // Boucle spatiale sur les SU
  OPENFLUID_UNITS_ORDERED_LOOP("SU",pUnit)
  {
    OPENFLUID_GetAttribute(pUnit,"slope",Slope);
    OPENFLUID_GetAttribute(pUnit,"flowdist",Flowdist);

    m_SUTransferSteps[pUnit->getID()] =
      std::ceil(Flowdist / (std::sqrt(Slope)*m_SUMeanCelerity
                            *double(OPENFLUID_GetDefaultDeltaT())));
  }

  // Boucle spatiale sur les RS
  OPENFLUID_UNITS_ORDERED_LOOP("RS",pUnit)
  {
    OPENFLUID_GetAttribute(pUnit,"slope",Slope);
    OPENFLUID_GetAttribute(pUnit,"length",Length);

    m_RSTransferSteps[pUnit->getID()] =
      std::ceil(Length / (std::sqrt(Slope)*m_RSMeanCelerity
                           *double(OPENFLUID_GetDefaultDeltaT())));
  }
}
```

3.5 initializeRun()

Dans la méthode `initializeRun()`, nous allons initialiser les variables produites à la valeur 0 pour les RS et SU.
Une fois complétée, la méthode `initializeRun()` devrait être similaire à:

```
openfluid::base::SchedulingRequest initializeRun()
{
  openfluid::core::SpatialUnit* pUnit;


  // Boucle spatiale sur les SU
  OPENFLUID_UNITS_ORDERED_LOOP("SU",pUnit)
  {
    OPENFLUID_InitializeVariable(pUnit,"water.surf.Q.downstream",0.0);
```

```
  }

  // Boucle spatiale sur les RS
  OPENFLUID_UNITS_ORDERED_LOOP("RS",pUnit)
  {
    OPENFLUID_InitializeVariable(pUnit,"water.surf.Q.downstream",0.0);
  }

  return DefaultDeltaT();
}
```

3.6 runStep()

Dans la méthode `runStep()`, nous allons i) déterminer les apports amont à transférer sur chaque SU, ii) calculer et ajouter le débit local à chaque SU, iii) déterminer les apports amont à transférer sur chaque RS. Les apports amont à transférer, ainsi que le calcul du débit local doivent tenir compte du temps de transfert propre à chaque unité et qui est calculé dans la partie `initializeRun()`.

Pour déterminer l'ensemble des unités contributives amont, nous allons utiliser la méthode fromSpatialUnits() pour chaque unité, et parcourir cette liste d'unités contributives avec l'instruction `OPENFLUID_UNITSLIST_LOOP`.

Une fois complétée, la méthode `runStep()` devrait être similaire à:

```
openfluid::base::SchedulingRequest runStep()
{
  int StepToTransfer;
  openfluid::core::DoubleValue QValue, UpQValue;
  openfluid::core::DoubleValue Area;
  openfluid::core::DoubleValue Runoff;
  openfluid::core::SpatialUnit* pUnit;
  openfluid::core::SpatialUnit* pUpUnit;
  openfluid::core::UnitsPtrList_t* UpUnitsList;
  unsigned int CurrentStep;
  openfluid::core::Duration_t DeltaT;

  DeltaT=OPENFLUID_GetDefaultDeltaT();
  CurrentStep = OPENFLUID_GetCurrentTimeIndex() /
     OPENFLUID_GetDefaultDeltaT();

  // Boucle spatiale sur les SU
  OPENFLUID_UNITS_ORDERED_LOOP("SU",pUnit)
  {
    StepToTransfer =  CurrentStep - m_SUTransferSteps[pUnit->getID()];

    QValue = 0.0;

    if (StepToTransfer >= 0)
```

```
  {
    UpUnitsList = pUnit->fromSpatialUnits("SU");

    // Boucle spatiale sur les SU en amont
    OPENFLUID_UNITSLIST_LOOP(UpUnitsList,pUpUnit)
    {
      OPENFLUID_GetVariable(pUpUnit,"water.surf.Q.downstream",
                            StepToTransfer*DeltaT,UpQValue);

      // Mise a jour des apports amont des SU
      QValue = QValue + UpQValue.get();
    }

    OPENFLUID_GetAttribute(pUnit,"area",Area);
    OPENFLUID_GetVariable(pUnit,"water.surf.H.runoff",
                          StepToTransfer*DeltaT,Runoff);

    // Calcul du debit local de la SU
    QValue = QValue + (Runoff * Area / double(DeltaT));
  }

  OPENFLUID_AppendVariable(pUnit,"water.surf.Q.downstream",QValue);
}

// Boucle spatiale sur les RS
OPENFLUID_UNITS_ORDERED_LOOP("RS",pUnit)
{
  StepToTransfer = CurrentStep - m_RSTransferSteps[pUnit->getID()];
  QValue = 0.0;

  if (StepToTransfer >= 0)
  {
    UpUnitsList = pUnit->fromSpatialUnits("SU");

    OPENFLUID_UNITSLIST_LOOP(UpUnitsList,pUpUnit)
    {
      OPENFLUID_GetVariable(pUpUnit,"water.surf.Q.downstream",
                            StepToTransfer*DeltaT,UpQValue);

      // Mise a jour des apports amont des SU
      QValue = QValue + UpQValue.get();
    }

    UpUnitsList = pUnit->fromSpatialUnits("RS");

    OPENFLUID_UNITSLIST_LOOP(UpUnitsList,pUpUnit)
    {
      OPENFLUID_GetVariable(pUpUnit,"water.surf.Q.downstream",
                            StepToTransfer*DeltaT,UpQValue);

      // Mise a jour des apports amont des RS
```

```
        QValue = QValue + UpQValue.get();
      }
    }

    OPENFLUID_AppendVariable(pUnit,"water.surf.Q.downstream",QValue);
  }

  return DefaultDeltaT();
}
```

4 Simulation

Pour la simulation, nous allons compléter le jeu de données du projet TP6 en ajoutant le simulateur `training.all.transfer` dans le modèle après les simulateur déjà présents .

4.1 ... avec l'interface OpenFLUID-Builder

Avant de procéder à la simulation, régler la configuration des sorties (*Monitoring*) afin de prendre en compte les variables créées par le simulateur nouvellement ajouté.

4.2 ... en ligne de commande

Une fois complété, le fichier `model.fluidx` devrait être structuré comme suit:

```
<?xml version="1.0" encoding="UTF-8"?>
<openfluid format="fluidx 4">
  <model>
    <simulator ID="training.signal.prod" enabled="1">
    </simulator>
    <simulator ID="training.su.prod" enabled="1">
      <param name="S" value="0.00005"/>
    </simulator>
    <simulator ID="training.all.transfer" enabled="1">
    </simulator>
  </model>
</openfluid>
```

La commande à exécuter est donc :

```
openfluid run <Bureau>/formation/projects/TP6/ -c
```

(à taper sur une seule ligne)

Si tout s'est bien passé, les résultats de la simulation sont accessibles dans `<Bureau>/formation/projects/TP6/OUT`.