OpenFLUID
Software Environment
for Modelling Fluxes in Landscapes

TP3 : Utilisation de variables en entrée

Objectifs:	Utiliser des variables produites par d'autres simulateurs
Pré-requis:	TP1, TP2
Fonctionnalités:	`DECLARE_REQUIRED_VAR()` `OPENFLUID_GetVariable()`

Nous allons créer un simulateur (`training.su.prod`) qui produit des variables de ruissellement et d'infiltration sur les SU à partir du signal de pluie, et en appliquant la méthode SCS.

1 Code source

1.1 Création du simulateur

Nous allons créer un projet Eclipse et générer le simulateur au travers du plugin OpenFLUID pour Eclipse, en appliquant la démarche proposée dans le TP1. Ce simulateur devra avoir comme caractéristiques :

- ID : `training.su.prod`
- Fichier .cpp : `SUSim.cpp`
- Classe du simulateur : `SUSimulator`
- Scheduling : `Scheduling uses the default DeltaT value`

1.2 Signature

Ce simulateur génèrera des valeurs de ruissellement et d'infiltration à partir d'un signal de pluie produite par un autre simulateur. Nous allons donc déclarer la prise en compte de ce signal de pluie (variable `water.atm-surf.H.rain`) et la production des deux variables (`water.surf.H.runoff` pour le ruissellement et `water.surf.H.infiltration` pour l'infiltration).
La prise en compte d'une variable produite par un autre simulateur sera déclarée au travers de l'instruction `DECLARE_REQUIRED_VAR`.

Une fois complétée, la signature devrait être similaire à :

```
BEGIN_SIMULATOR_SIGNATURE("training.su.prod")

  DECLARE_NAME("");
  DECLARE_DESCRIPTION("");

  DECLARE_VERSION("13.05");
  DECLARE_STATUS(openfluid::ware::EXPERIMENTAL);

  DECLARE_DOMAIN("");
  DECLARE_PROCESS("");
  DECLARE_METHOD("");
  DECLARE_AUTHOR("","");

  DECLARE_REQUIRED_VAR("water.atm-surf.H.rain","SU","rainfall␣height␣on␣the␣SU","m");

  DECLARE_PRODUCED_VAR("water.surf.H.runoff","SU","water␣runoff␣height␣on␣surface␣of␣SU","m");
  DECLARE_PRODUCED_VAR("water.surf.H.infiltration","SU",
                       "water␣infiltration␣height␣through␣the␣surface␣of␣SU","m");


// Scheduling
  DECLARE_SCHEDULING_DEFAULT;


END_SIMULATOR_SIGNATURE
```

1.3 initializeRun()

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

```
openfluid::base::SchedulingRequest initializeRun()
{
  openfluid::core::Unit* SU;
  OPENFLUID_UNITS_ORDERED_LOOP("SU",SU)
  {
    OPENFLUID_InitializeVariable(SU,"water.surf.H.runoff",0.0);
    OPENFLUID_InitializeVariable(SU,"water.surf.H.infiltration",0.0);
  }

  return DefaultDeltaT();
}
```

1.4 runStep()

Dans la méthode runStep(), nous allons calculer le partage ruissellement-infiltration à partir du signal de pluie en utilisant la méthode SCS(USDA, TR-55 [1]).

Soit R le ruissellement, P la pluie, S le coefficient de rétention
$0.0001 \leq S \leq 0.008$

Si $P \leq (0.2 \cdot S)$ alors $R = 0.0$

Si $P > (0.2 \cdot S)$ alors $R = \frac{(P-0.2 \cdot S)^2}{(P+0.8 \cdot S)}$

1. Technical Release 55 : Urban Hydrology for Small Watersheds. USDA (U.S. Department of Agriculture). 1986. http ://www.nrcs.usda.gov/Internet/FSE_DOCUMENTS/stelprdb1044171.pdf

Nous allons utiliser une boucle spatiale sur les SU, récupérer le signal de pluie au travers de l'instruction `OPENFLUID_GetVariable`, et produire le ruissellement et l'infiltration calculés à l'aide de deux instructions `OPENFLUID_AppendVariable`. Pour cet exercice nous fixerons la valeur de S à une valeur arbitraire choisie dans sa plage de validité.

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

```
openfluid::base::SchedulingRequest runStep()
{
  openfluid::core::Unit* pSU;
  openfluid::core::DoubleValue RainValue;
  openfluid::core::DoubleValue RunoffValue;
  openfluid::core::DoubleValue InfiltrationValue;
  double S;

  OPENFLUID_UNITS_ORDERED_LOOP("SU",pSU)
  {
    S = 0.0035;  // Valeur du coeff. de retention fixe à 0,0035

    // recuperation de la valeur du signal de pluie
    OPENFLUID_GetVariable(pSU,"water.atm-surf.H.rain",RainValue);

    // calcul du ruissellement selon la methode SCS
    RunoffValue = 0.0;
    if (RainValue > (0.2*S))
    {
      RunoffValue = std::pow(RainValue-(0.2*S),2)/(RainValue+(0.8*S));
    }

    // calcul de l'infiltration par deduction
    InfiltrationValue = RainValue-RunoffValue;

    // production de l'infiltration et du ruissellement
    OPENFLUID_AppendVariable(pSU,"water.surf.H.infiltration",InfiltrationValue);
    OPENFLUID_AppendVariable(pSU,"water.surf.H.runoff",RunoffValue);
  }

  return DefaultDeltaT();
}
```

2 Simulation

Pour la simulation, nous allons compléter le jeu de données "Bassin versant TP" en ajoutant le simulateur `training.su.prod` dans le modèle.

Attention: L'ordre des simulateurs dans le modèle est important !

2.1 ... avec l'interface OpenFLUID-Builder

Nous allons tout d'abord créer un nouveau projet OpenFLUID nommé TP3 (dans `<Bureau>/formation/projects/TP3`) dont les données seront basées sur le projet TP2, cocher l'option *Importer des données/ Importer des données d'un projet existant* et choisir le projet *TP2*.

Ensuite, rajouter le simulateur (`training.su.prod`) dans le modèle du projet TP3, après le simulateur déjà présent (training.signal.prod).

2.2 ... en ligne de commande

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

```
<?xml version="1.0" encoding="UTF-8"?>
<openfluid>
  <model>
     <simulator ID="training.signal.prod"/>
     <simulator ID="training.su.prod" />
  </model>
</openfluid>
```

La commande à exécuter est donc :

```
openfluid -i <Bureau>/formation/projects/TP3/IN
  -o <Bureau>/formation/projects/TP3/OUT
```

(à taper sur une seule ligne)

Si tout s'est bien passé, les résultats de la simulation sont accessibles dans `<Bureau>/formation/projects/TP3/OUT`.