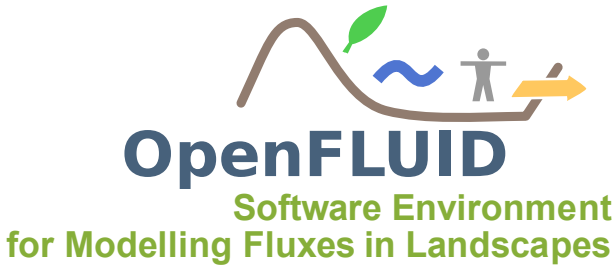

TP4 : Utilisation de paramètres de simulateurs

Objectifs:	Utiliser des paramètres de simulateurs depuis la définition du modèle
Pré-requis:	TP3

Fonctionnalités:	`DECLARE_USED_PARAMETER()`
	`openfluid::ware::WareParams_t`
	`OPENFLUID_GetSimulatorParameter()`

Nous allons améliorer le simulateur `training.su.prod` créé lors du TP3, en ajoutant le coefficient de rétention S comme paramètre global du simulateur, afin qu'il ne soit pas fixé dans le code source.

1 Création d'un nouveau projet OpenFLUID

Afin de repartir du TP précédent, créer un projet OpenFLUID nommé TP4 et y importer le jeu de données d'entrée du TP3.

2 Code source

Uen fois dans le projet *TP4*, pour modifier le code source du simulateur `training.su.prod`, aller dans *Développement*, *Ouvrir un simulateur* et sélectionner le simulateur `training.su.prod`. Les modifications du code source vont porter sur la signature, sur les attributs privés de la classe, sur le constructeur de la classe, ainsi que sur les méthodes `initParams()` et `runStep()`.

2.1 Signature

Nous allons déclarer la prise en compte d'un paramètre de simulateur pour le coefficient de rétention (nommé `S`). Cette déclaration se fait au travers de l'instruction `DECLARE_USED_PARAMETER`.

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

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


// Scheduling
  DECLARE_SCHEDULING_DEFAULT;


END_SIMULATOR_SIGNATURE
```

2.2 Attribut privé

Nous allons tout d'abord déclarer un attribut privé afin que la valeur de S récupérée depuis la méthode `initParams()` puisse être utilisée depuis `runStep()`. Cet attribut sera nommé `m_S`.

Note: Les attributs privés sont accessibles depuis toutes les méthodes de la classe

Dans la classe `SUSimulator`, une fois complétés, les attributs privés devraient être similaires à:

```
private:
  double m_S;
```

2.3 Constructeur

Les paramètres de simulateur étant facultatifs, il est préférable de prévoir une valeur par défaut en cas d'absence de ce paramètre. Cette valeur par défaut est initialisée dans le constructeur du simulateur, appelé `SUSimulator()`.
Une fois complété, le constructeur devrait être similaire à:

```
SUSimulator(): PluggableSimulator(), m_S(0.00005) // initialisation de S a 0.00005
{

}
```

2.4 initParams()

Dans la méthode `initParams()`, nous allons récupérer la valeur du paramètre `S` en utilisant l'instruction `OPENFLUID_GetSimulatorParameter`. Cette instruction ne fournit une valeur que

si le paramètre est présent. Si celui-ci est absent, la valeur par défaut est conservée.

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

```
void initParams(const openfluid::ware::WareParams_t& Params)
{
  // recuperation du parametre S
  OPENFLUID_GetSimulatorParameter(Params,"S",m_S);
}
```

Note: Veiller à bien décommenter le paramètre nommé `Params` pour la méthode initParams(), le cas échéant.

2.5 runStep()

La seule modification à apporter dans le `runStep()` concerne la valeur de la variable S. Nous allons lui donner la valeur du paramètre de simulateur lu dans `initParams()`.

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

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

  OPENFLUID_UNITS_ORDERED_LOOP("SU",pSU)
  {
    S = m_S;  // Coeff. de retention recoit la valeur lue en parametre

    // 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();
}
```

3 Simulation

3.1 ... avec l'interface OpenFLUID-Builder

Donner une valeur au paramètre S du simulateur `training.su.prod`, comprise entre 0.000002 et 0.00005. Plus la valeur est grande, plus le sol est infiltrant.

Lancer plusieurs simulations en modifiant ce paramètre.

3.2 ... en ligne de commande

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

```
<?xml version="1.0" standalone="yes"?>
<openfluid>
 <model>
  <simulator ID="training.signal.prod" enabled="1">
  </simulator>
  <simulator ID="training.su.prod" enabled="1">
   <param name="S" value="0.00005"/>
  </simulator>
 </model>
</openfluid>
```

La commande à exécuter est donc :

```
openfluid run <Bureau>/formation/projects/TP4 -c
```

(à taper sur une seule ligne)

Si tout s'est bien passé, les résultats de la simulation sont accessibles dans `<Bureau>/formation/projects/TP4/OUT`.