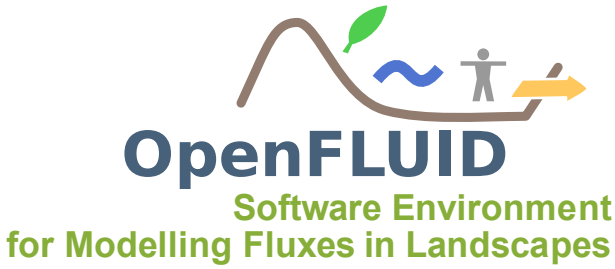

TP4 : Utilisation de paramètres de fonctions de simulation

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

Nous allons améliorer la fonction de simulation (`training.su.prod`), créée lors du TP3, en ajoutant le coefficient de rétention S comme paramètre global de la fonction et non pas comme fixé dans le code source

1 Code source

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()`.

1.1 Signature

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

Une fois complétée, la signature devrait être similaire à :

```
BEGIN_SIGNATURE_HOOK
  DECLARE_SIGNATURE_ID("training.su.prod");
  DECLARE_SIGNATURE_NAME("");
  DECLARE_SIGNATURE_DESCRIPTION("");

  DECLARE_SIGNATURE_VERSION("12.03");
  DECLARE_SIGNATURE_SDKVERSION;
  DECLARE_SIGNATURE_STATUS(openfluid::base::EXPERIMENTAL);

  DECLARE_SIGNATURE_DOMAIN("");
  DECLARE_SIGNATURE_PROCESS("");
  DECLARE_SIGNATURE_METHOD("");
  DECLARE_SIGNATURE_AUTHORNAME("Doe␣J.");
  DECLARE_SIGNATURE_AUTHOREMAIL("doe@foobar.org");

  DECLARE_FUNCTION_PARAM("s","","-");
```

```
  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");
END_SIGNATURE_HOOK
```

1.2 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 `class SUFunction`, une fois complétés, les attributs privés devraient être similaires à :

```
private:
  double m_S;
```

1.3 Constructeur

Les paramètres de fonction 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 de la fonction, appelé `SUFunction()`
Une fois complété, le constructeur devrait être similaire à :

```
SUFunction(): PluggableFunction()
{
  m_S = 0.0035;  // initialisation de S à 0.0035;
}
```

1.4 initParams()

Dans la méthode `initParams()`, nous allons récupérer la valeur du paramètre `s` en utilisant l'instruction `OPENFLUID_GetFunctionParameter`. Cette instruction ne fournit une valeur que si le paramètre est présent. Si il n'est pas présent, la valeur par défaut est conservée.

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

```
bool initParams(openfluid::core::FuncParamsMap_t Params)
{
  // recuperation du parametre S
  OPENFLUID_GetFunctionParameter(Params,"s",&m_S);

  return true;
}
```

1.5 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 fonction lu dans `initParams()`.

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

```
bool runStep(const openfluid::base::SimulationStatus* SimStatus)
{
  openfluid::core::Unit* pSU;
  openfluid::core::DoubleValue RainValue;
  openfluid::core::DoubleValue RunoffValue;
  openfluid::core::DoubleValue InfiltrationValue;
  double S;
  DECLARE_UNITS_ORDERED_LOOP(5);

  BEGIN_UNITS_ORDERED_LOOP(5,"SU",pSU);
    S = m_S;  // Coeff. de retention recoit la valeur lue en parametre

    OPENFLUID_GetVariable(pSU,"water.atm-surf.H.rain",SimStatus->getCurrentStep(),&RainValue);

    RunoffValue = 0.0;
    if (RainValue > (0.2*S))
    {
      RunoffValue = std::pow(RainValue-(0.2*S),2)/(RainValue+(0.8*S));
    }
    InfiltrationValue = RainValue-RunoffValue;

    OPENFLUID_AppendVariable(pSU,"water.surf.H.infiltration",InfiltrationValue);
    OPENFLUID_AppendVariable(pSU,"water.surf.H.runoff",RunoffValue);
  END_LOOP;

  return true;
}
```

2 Simulation

Dans le jeu de données d'entrée, nous allons ajouter le paramètre `s` à la fonction `training.su.prod`. Pour cela, il faut modifier le fichier `model.fluidx` et rajouter le paramètre dans une sous-balise `<param>` de la balise `<function>` concernée.

2.1 ... avec l'interface OpenFLUID-Builder

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

Ensuite, donner une valeur au paramètre S, comprise entre 0.0001 et 0.008. Plus la valeur est grande, plus le sol est infiltrant.

Lancer plusieurs simulations en modifiant ce paramètre.

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>
     <function fileID="training.signal.prod"/>
     <function fileID="training.su.prod">
       <param name="s" value="0.0035" />
```

```
    </function>
  </model>
</openfluid>
```

La commande à exécuter est donc :

```
openfluid-engine -i /home/openfluid/formation/datasets/TP1-TP7
  -o /home/openfluid/formation/outputs/TP4
```

(à taper sur une seule ligne)

Si tout s'est bien passé, les résultats de la simulation sont accessibles dans `/home/openfluid/formation/outputs/TP4`.