

TP7 : Utilisation d'évènements

Objectifs:	Prendre en compte des évènements discrets, distribués dans l'espace, et pouvant influencer sur le comportement du modèle
Pré-requis:	TP5
Fonctionnalités:	<code>openfluid::core::Event</code> <code>openfluid::core::EventsCollection</code> <code>OPENFLUID_GetEvents()</code>

Nous allons créer un nouveau simulateur (`training.su.S-evol`) qui va faire évoluer la valeur du coefficient de rétention S en fonction d'évènements discrets qui peuvent arriver tout au long de la période de simulation. Cette évolution sera produite sous la forme d'une variable de simulation. Nous modifierons ensuite le simulateur `training.su.prod` développé dans le TP3 pour qu'il prenne en compte cette variable dans son calcul du partage ruissellement-infiltration.

Note: Chaque évènement discret survient à un instant donné sur une unité spatiale donnée, et porte des informations qui le caractérise.

Note: Les évènements discrets peuvent être soit pré-positionnés dans un calendrier dans le jeu de données, soit générés par un ou plusieurs simulateurs au cours de la simulation.

1 Calendrier d'évènements

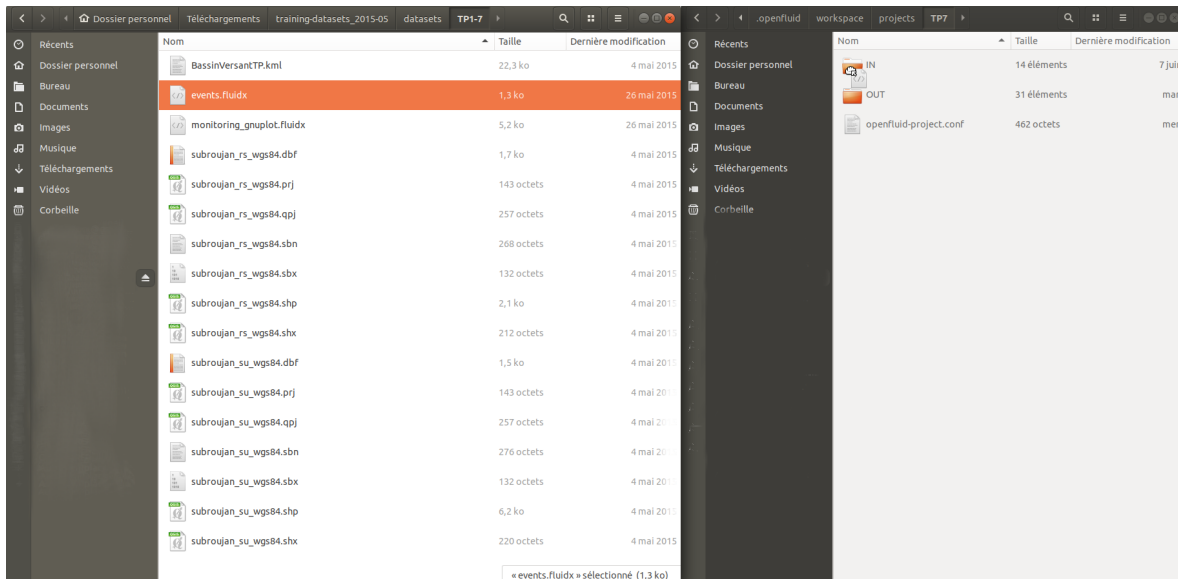
Pour ce TP, les évènements à prendre en compte sont des opérations sur les unités spatiales de type SU. Ces opérations sont de trois types A, B et C, fixant une nouvelle valeur au coefficient de rétention S .

type d'opération	valeur de S
A	0.0001
B	0.00005
C	0.00001

Table 1: Tableau des types de pratiques

Inclusion des données d'événements

Le fichier `events.fluidx` contenant les informations d'événements est présent dans l'archive `datasets`. Pour l'inclure au projet, ouvrir le dossier du projet courant en passant par le menu *Projet, Ouvrir dans un explorateur de fichiers*, puis copier le fichier `events.fluidx` depuis le dossier `datasets/TP1-7` dans le dossier `IN` du projet courant.



2 Simulateur `training.su.S-evol`

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

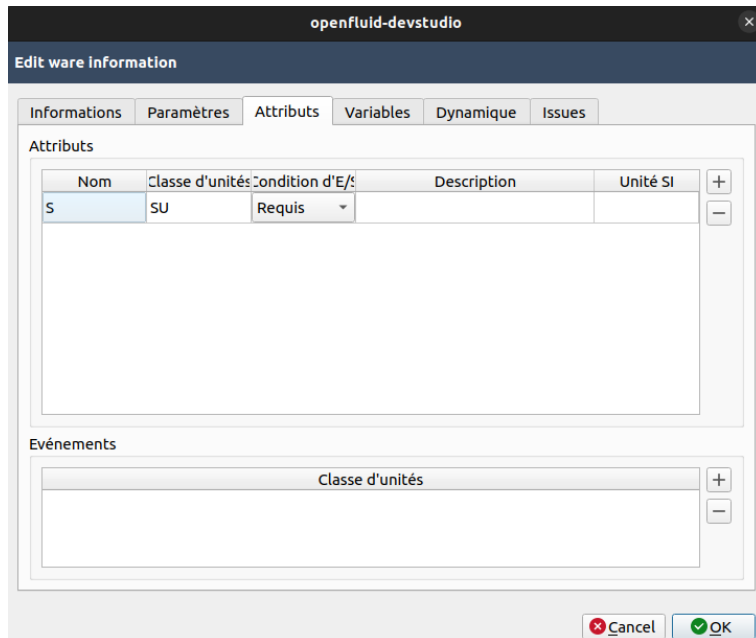
- ID : `training.su.S-evol`
- Classe du simulateur : `SEvolutionSimulator`
- Planification : La planification utilise le `DeltaT` par défaut

A chaque pas de temps, ce simulateur va prendre en compte les évènements discrets sur les SU et produire une nouvelle valeur pour la variable `soil.surf-uz.-.S` représentant les valeurs successives de S . Cette variable sera initialisée avec la valeur contenue dans l'attribut S . Elle ne sera produite que lorsque S nécessitera une évolution, c'est-à-dire lorsqu'un évènement se produira sur le pas de temps en cours.

2.1 Signature

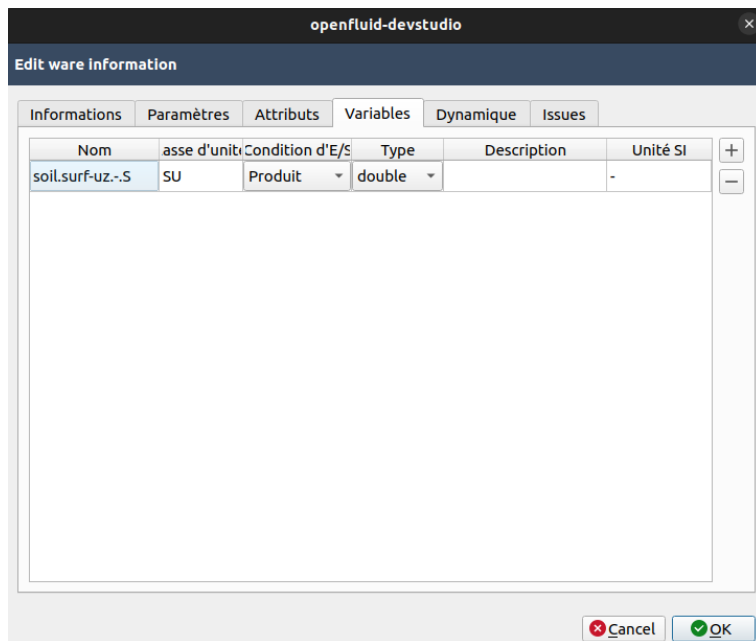
Nous allons déclarer la production de la variable `soil.surf-uz.-.S`. Pour cela, ouvrir le menu d'édition de signature et aller dans l'onglet *Variables* et ajouter une nouvelle variable produite avec le bouton (+). Nous allons également requérir l'attribut *S* qui sera utilisé dans la phase d'initialisation.

Une fois complétée, la signature devrait être similaire à:



The screenshot shows the 'Edit ware information' dialog box with the 'Attributs' tab selected. The table below represents the data shown in the interface:

Nom	Classe d'unités	Condition d'E/S	Description	Unité SI
S	SU	Requis		



The screenshot shows the 'Edit ware information' dialog box with the 'Variables' tab selected. The table below represents the data shown in the interface:

Nom	Classe d'unités	Condition d'E/S	Type	Description	Unité SI
soil.surf-uz.-.S	SU	Produit	double		-

2.2 Attributs privés

Nous allons déclarer trois constantes privées permettant de stocker les valeurs possibles lorsque S évolue sous l'impact des opérations:

- `m_SMAX`, fixé à 0.0001, pour les opérations de type A
- `m_SMED`, fixé à 0.00005, pour les opérations de type B
- `m_SMIN`, fixé à 0.00001, pour les opérations de type C

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

```
private:
    const double m_SMAX = 0.0001;
    const double m_SMED = 0.00005;
    const double m_SMIN = 0.00001;
```

2.3 initializeRun()

Dans la méthode `initializeRun()`, nous allons initialiser la variable `soil.surf-uz.-.S` pour chaque unité spatiale de classe SU à partir de la valeur de l'attribut `S`.

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

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

    OPENFLUID_UNITS_ORDERED_LOOP("SU", SU)
    {
        openfluid::core::DoubleValue SValue;

        OPENFLUID_GetAttribute(SU, "S", SValue);
        OPENFLUID_InitializeVariable(SU, "soil.surf-uz.-.S", SValue);
    }

    return DefaultDeltaT();
}
```

2.4 runStep()

Dans la méthode `runStep()`, nous allons tout d'abord déterminer la période pour laquelle nous allons prendre en compte des événements, à savoir $]Date_{courante} - \Delta t, Date_{courante}]$. Ensuite, pour chaque unité spatiale de la classe SU, nous allons récupérer les événements sur cette période. S'il y a au moins un événement, nous allons 1) récupérer le type d'opération (`operation_type`) associé à l'évènement, 2) déterminer la nouvelle valeur de S en fonction de ce type d'opération, 3) produire une nouvelle valeur pour la variable `soil.surf-uz.-.S`.

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

```

openfluid::base::SchedulingRequest runStep()
{
    openfluid::core::SpatialUnit * SU;

    openfluid::core::DateTime BeginDate,EndDate;

    // calcul des dates de debut et de fin pour recuperer les evenements
    EndDate = OPENFLUID_GetCurrentDate();
    BeginDate = EndDate - (OPENFLUID_GetDefaultDeltaT()-1);

    OPENFLUID_UNITS_ORDERED_LOOP("SU",SU)
    {
        openfluid::core::EventsCollection EvColl;

        // recuperation des evenements sur la periode courante
        OPENFLUID_GetEvents(SU,BeginDate,EndDate,EvColl);

        if (EvColl.getCount())
        {
            openfluid::core::DoubleValue SValue;
            std::string EventType;

            EvColl.eventsList()->back().getInfoAsString("operation_type",
                EventType);

            // traitement de l'information contenue dans "operation_type"
            if (EventType == "A")
                OPENFLUID_AppendVariable(SU,"soil.surf-uz.-.S",m_SMAX);
            else if (EventType == "B")
                OPENFLUID_AppendVariable(SU,"soil.surf-uz.-.S",m_SMED);
            else if (EventType == "C")
                OPENFLUID_AppendVariable(SU,"soil.surf-uz.-.S",m_SMIN);
        }
    }

    return DefaultDeltaT();
}

```

3 Simulation sans prise en compte de l'évolution de S

Pour la simulation, nous allons compléter le jeu de données en ajoutant le simulateur `training.su.S-evol` dans le modèle, et le positionner avant le simulateur `training.su.prod`.

3.1 ... avec l'interface OpenFLUID-Builder

Procéder classiquement à l'ajout du simulateur depuis l'onglet *Modèle*.

3.2 ... 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.S-evol" 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/TP7/ -c
```

(à taper sur une seule ligne)

Si tout s'est bien passé, les résultats de la simulation sont accessibles dans `<Bureau>/formation/projects/TP7/OUT`.

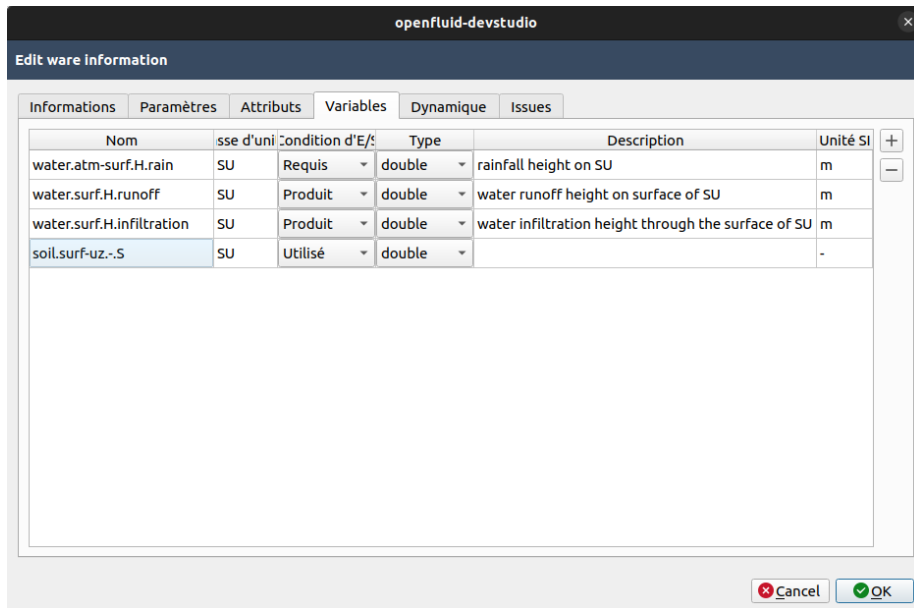
4 Simulateur `training.su.prod`

Nous allons modifier ce simulateur afin qu'il prenne en compte la variable `soil.surf-uz.-.S` si elle existe. Dans le cas où cette variable n'existe pas, le comportement du simulateur devra rester inchangé.

4.1 Signature

Nous allons déclarer l'utilisation de la variable `soil.surf-uz.-.S`, uniquement si elle existe. Pour cela, ouvrir le menu d'édition de signature et aller dans l'onglet *Variables* et ajouter une nouvelle variable utilisée en cliquant sur le bouton (+).

Une fois mise à jour, la signature devrait être similaire à:



4.2 runStep()

Dans la méthode `runStep()`, nous allons tester la présence de la variable `soil.surf-uz.-.S`. Si cette variable existe, nous allons en récupérer la dernière valeur disponible. Si elle n'existe pas, le comportement de ce simulateur reste inchangé, à savoir l'attribut spatial `S` ou l'utilisation du paramètre de simulateur `S` (dans cet ordre de priorité).

Une fois mise à jour, 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;

        // recuperation de la variable soil.surf-uz.-.S si elle existe
        if (OPENFLUID_IsVariableExist(pSU, "soil.surf-uz.-.S"))
        {
            openfluid::core::IndexedValue LatestSVal;

            // recuperation de la derniere valeur disponible pour soil.surf-uz.-.S
            OPENFLUID_GetLatestVariable(pSU, "soil.surf-uz.-.S", LatestSVal);
            S = LatestSVal.value()->asDoubleValue();
        }
    }
}
```

```

else if (OPENFLUID_IsAttributeExist(pSU,"S")) // recuperation de l
    'attribut S s'il existe
{
    OPENFLUID_GetAttribute(pSU,"S",S);
}

// 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();
}

```

5 Simulation avec prise en compte de l'évolution de S

Relancer la simulation comme précédemment.