

Tutoriel : Écrire un module dans le formalisme mathématique des équations différentielles ordinaires (ODE)

Version 1.3 du document

Testé avec la version VLE : **1.0.0**

Paquet support à ce tutoriel : **LotkaVolterra**, disponible sur le site RECORD XXXX

Auteurs : Équipe Plate-forme RECORD

Contact : XXX

Date : 07/11/11

Table des matières

1	Introduction.....	2
2	Extension VLE DESS : Approche par les méthodes d'intégration : "Euler" ou "Runge Kutta 4" (à pas de temps d'intégration fixe).....	3
2.1	Étape 1 : Création sous gyle.....	3
2.2	Étape 2 : Codage de la dynamique en C++.....	3
2.3	Étape 3 : Paramétrage.....	3
2.4	Exemples: intégration par Euler, intégration par Runge Kutta 4, perturbation.....	4
2.4.1	Exemple intégration par Euler : simulateur LV_DESS_euler.vpz.....	4
2.4.2	Exemple intégration par Runge Kutta 4 : simulateur LV_DESS_rk4.vpz.....	5
2.4.3	Exemple de perturbation de la dynamique: simulateur LV_DESS_perturb_x.vpz.....	5
3	Extension VLE QSS::Simple : Approche par la méthode d'intégration : QSS (Quantized State System).....	6
4	Extension VLE QSS::Multiple : Approche par la méthode d'intégration : QSS (Quantized State System).....	7
5	Exemples de couplage de modèles d'EDO	8
6	Annexes :	9
	Annexe 1 : Pattern de base pour l'extension DESS	9
	Annexe 2 : Pattern de base pour l'extension QSS simple.....	11
	Annexe 3 : Pattern de base pour l'extension QSS multiple	13
	Annexe 4 : utilisation de l'option thresholds.....	15
	Annexe 5: Références.....	16

Note : voir aussi <http://vle.toulouse.inra.fr/wiki/DESS/fr> et <http://vle.toulouse.inra.fr/wiki/QSS/fr>

1 Introduction

Ce tutoriel vous donne les éléments de base pour développer vos propres modules en utilisant des systèmes d'équations différentielles ordinaires (EDO). Les exemples de ce tutoriel sont disponibles dans le paquet "LotkaVolterra" fourni par la plate-forme RECORD. Ils sont basés sur un système Lotka-Volterra (désigné aussi sous le nom de modèle proie prédateur) qui correspond à un couple d'équations différentielles non-linéaires du premier ordre. C'est un système couramment utilisé pour décrire la dynamique de systèmes biologiques dans lesquels un prédateur et sa proie interagissent. Il a été proposé indépendamment par Alfred James Lotka en 1925 et Vito Volterra en 1926. Les exemples présentés ici sont appliqués au cas pucerons-coccinelles.

$$\frac{dx}{dt} = x(t) \cdot (\alpha - \beta \cdot y(t)) \quad (1)$$

$$\frac{dy}{dt} = -y(t) \cdot (\delta - \gamma \cdot x(t)) \quad (2)$$

avec:

- $x(t)$ l'effectif des proies au temps t
- $y(t)$ l'effectif des prédateurs au temps t
- α le taux de reproduction des proies en l'absence de prédateurs
- β le taux de mortalité des proies due aux prédateurs
- δ le taux de mortalité des prédateurs en l'absence de proies
- γ le taux de reproduction des prédateurs en fonction des proies mangées

Dans ce tutoriel, on utilisera pour ce modèle :

Les valeurs des paramètres: $\alpha = 1.5$, $\beta = 1$, $\delta = 3$, $\gamma = 1$

Les valeurs initiales x_0 et y_0 sont 10 et 5 respectivement

Ce qui correspond à un état d'équilibre des 2 populations : $x = \delta/\gamma = 3$, $y = \alpha/\beta = 1.5$

Plusieurs extensions sont proposées dans le cadre RECORD-VLE correspondant à des méthodes de résolution différentes des systèmes d'EDO, et vont être détaillées dans les paragraphes suivants:

- **DESS** : Differential Equation System Specification (Zeigler et al., 2000)
Classe "vle::extension::DifferentialEquation::DESS" définie dans <vle/extension/differential-equation/DESS.hpp>
- **QSS Simple** : Quantized State System (E. Kofman, 2001)
Classe "vle::extension::QSS::Simple" définie dans <vle/extension/differential-equation/QSS.hpp>
- **QSS Multiple** : Quantized State System Multiple (E. Kofman, 2001)
Classe "vle::extension::QSS::Multiple" définie dans <vle/extension/differential-equation/QSS.hpp>

2 Extension VLE DifferenceEquation : Approche par les méthodes d'intégration : "Euler" ou "Runge Kutta 4" (à pas de temps d'intégration fixe)

Créer un module en utilisant l'extension DifferenceEquation

L'extension DifferenceEquation permet de développer des modèles qui calculent la valeur d'une ou plusieurs variables réelles en t en fonction d'elles-mêmes à $t-1$, $t-2$, ... et fonction d'autres variables réelles en t , $t-1$, $t-2$, ... Nous avons appelé ça les équations aux différences mais cette extension est plus générale car seule la propriété ci-dessus est à respecter. Il existe trois façons de manipuler les équations aux différences :

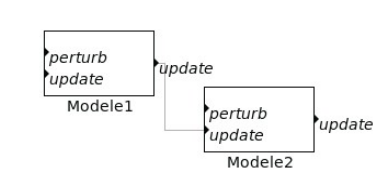
- soit de manière individuelle : un modèle par équation (DifferenceEquation::Simple)
- soit sous forme d'un système : un modèle pour un ensemble d'équations (DifferenceEquation::Multiple)
- soit de manière générique : le nombre de variables misent en jeu dans l'équation est indéterminé (DifferenceEquation::Generic)

A l'aide de l'extension DifferenceEquation il est possible d'implémenter des méthodes d'intégration "Euler" ou "Runge Kutta 4".

2.1 Étape 1 : Création sous gvl

Après avoir créé sous gvl la boîte représentant le module, on crée les ports afférents à cette extension et permettant le couplage. Les seuls ports d'entrée et de sortie possibles pour cette extension :

- Ports d'entrée:
 - **update** : c'est le port de couplage, qui permet de recevoir à chaque pas de temps les valeurs des variables externes
 - **perturb** : port pour perturber la dynamique
- Ports de sortie:
 - **update** : port de couplage qui permet d'envoyer à chaque pas de temps la valeur de la variable d'état du modèle.



Remarque : possibilité d'avoir un port de sortie **out** utilisant des seuils de front montant et descendant de la variable d'état, voir Annexe 4 pour plus de détails sur l'utilisation.

2.2 Étape 2 : Codage de la dynamique en C++

Pour cette extension, il n'y a pas encore d'interface graphique facilitant l'écriture du module (comme c'est le cas pour l'extension DifferenceEquationMultiple), il faut donc directement la coder en C++. Pour cela, vous pouvez reprendre le fichier exemple disponible dans le paquet LotkaVolterra/src/DESS/DESSx.cpp (confer annexe 1).

2.3 Étape 3 : Paramétrage

On distingue deux types de paramétrages :

- **Paramètres de l'équation différentielle** : Ce qui correspond aux α , β de l'équation (1) dans le modèle proie.

Parameters	Type	Overview
alpha	double	1.5
beta		

- **Paramètres de l'extension DESS** :

- **time-step** (type double) : Le pas de temps d'intégration

Parameters	Type	Overview
timeStep	double	0.001

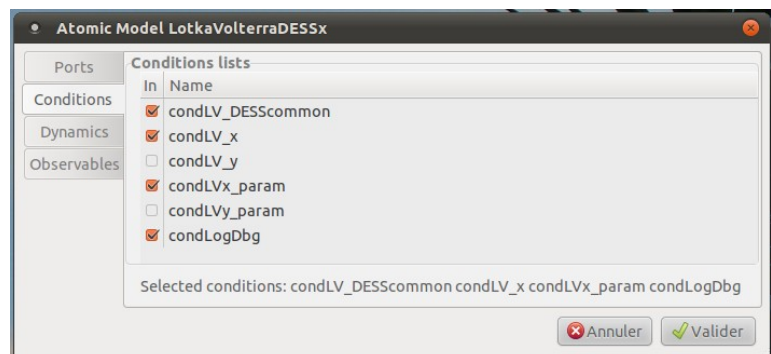
- **mode** (type string) : le schéma numérique d'intégration : **euler** ou **rk4** (par défaut)

Parameters	Type	Overview
method	string	euler

- **variables** (type set) **name** (type string) et **value** (type double) : le nom et la valeur initiale de la variable d'état gérée par le modèle

Parameters	Type	Overview
name	string	x
value		

La définition des paramètres se fait comme pour l'extension EquationsDifférencesMultiples, via l'interface graphique gvl, par le menu Simulation/Conditions. On affecte ensuite à chaque boîte les conditions lui correspondant (double-cliquer sur la boîte et cocher les groupes de conditions)



3 Extension VLE DESS : Approche par les méthodes d'intégration : "Euler" ou "Runge Kutta 4" (à pas de temps d'intégration fixe)

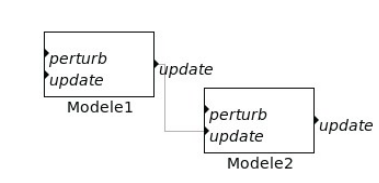
Créer un module en utilisant l'extension DESS

Dans le cas des méthodes d'intégration "Euler" ou "Runge Kutta 4", on utilise l'extension DESS fournie par VLE. Dans sa forme actuelle, elle permet de coder une équation par modèle atomique (1 module ~ 1 équation ~ 1 variable d'état).

3.1 Étape 1 : Création sous gvle

Après avoir créé sous gvle la boîte représentant le module, on crée les ports afférents à cette extension et permettant le couplage. Les seuls ports d'entrée et de sortie possibles pour cette extension :

- Ports d'entrée:
 - **update** : c'est le port de couplage, qui permet de recevoir à chaque pas de temps les valeurs des variables externes
 - **perturb** : port pour perturber la dynamique
- Ports de sortie:
 - **update** : port de couplage qui permet d'envoyer à chaque pas de temps la valeur de la variable d'état du modèle.



Remarque : possibilité d'avoir un port de sortie **out** utilisant des seuils de front montant et descendant de la variable d'état, voir Annexe 4 pour plus de détails sur l'utilisation.

3.2 Étape 2 : Codage de la dynamique en C++

Pour cette extension, il n'y a pas encore d'interface graphique facilitant l'écriture du module (comme c'est le cas pour l'extension DifferenceEquationMultiple), il faut donc directement la coder en C++. Pour cela, vous pouvez reprendre le fichier exemple disponible dans le paquet LotkaVolterra/src/DESS/DESSx.cpp (confer annexe 1).

3.3 Étape 3 : Paramétrage

On distingue deux types de paramétrages:

- **Paramètres de l'équation différentielle** : Ce qui correspond aux α , β de l'équation (1) dans le modèle proie.
- **Paramètres de l'extension DESS** :
 - **timeStep** (type double) : Le pas de temps d'intégration
 - **method** (type string) : le schéma

Parameters	Type	Overview
alpha	double	1.5
beta		

Parameters	Type	Overview
timeStep	double	0.001

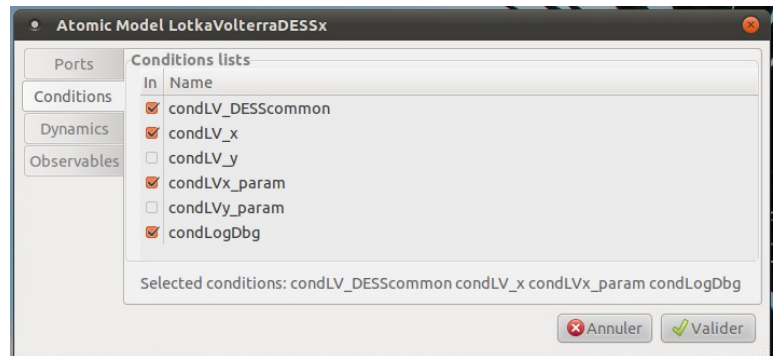
Parameters	Type	Overview
method	string	euler

numérique d'intégration : **euler** ou **rk4** (par défaut)

- **name** (type string) et **value** (type double) : le nom et la valeur initiale de la variable d'état gérée par le modèle

Parameters	Type	Overview
name	string	x
value		

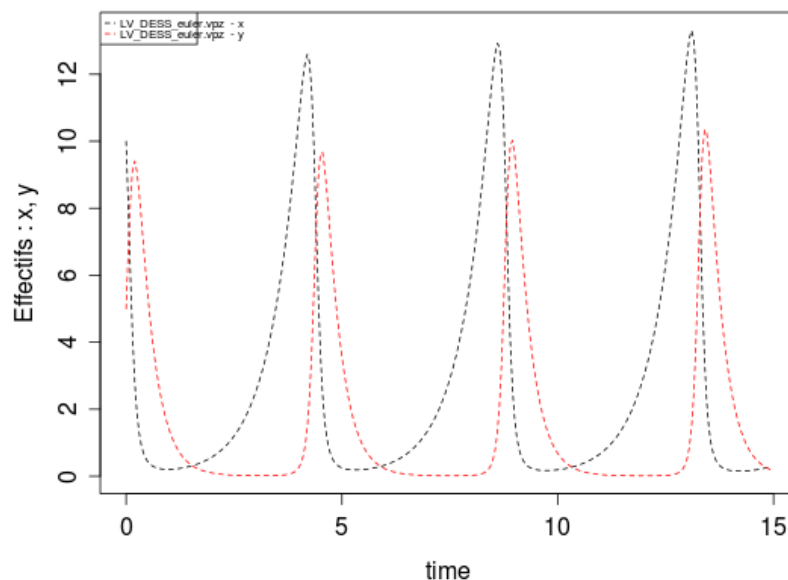
La définition des paramètres se fait comme pour l'extension EquationsDifférencesMultiples, via l'interface graphique gyle, par le menu Simulation/Conditions. On affecte ensuite à chaque boîte les conditions lui correspondent (double-cliquer sur la boîte et cocher les groupes de conditions)



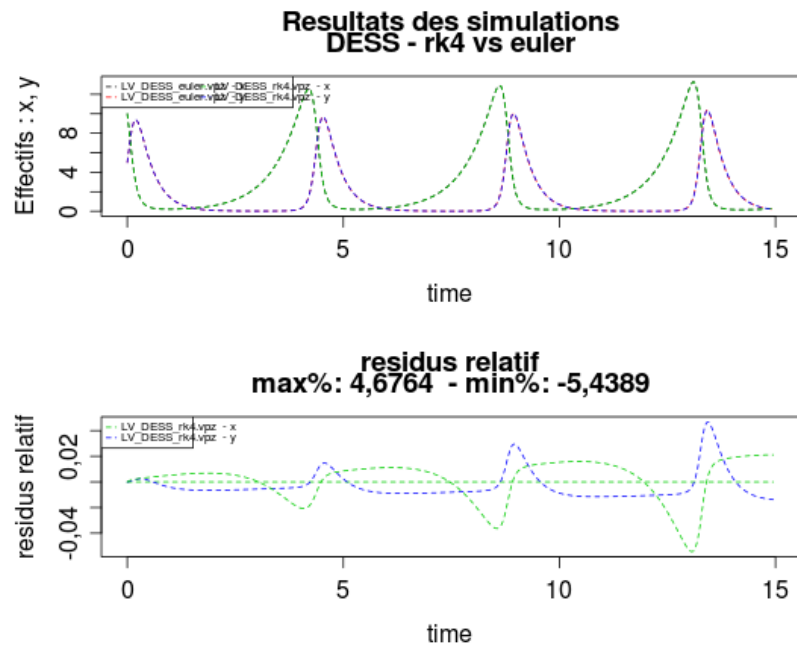
3.4 Exemples : intégration par Euler, intégration par Runge Kutta 4, perturbation

3.4.1 Exemple intégration par Euler : simulateur LV_DESS_euler.vpz

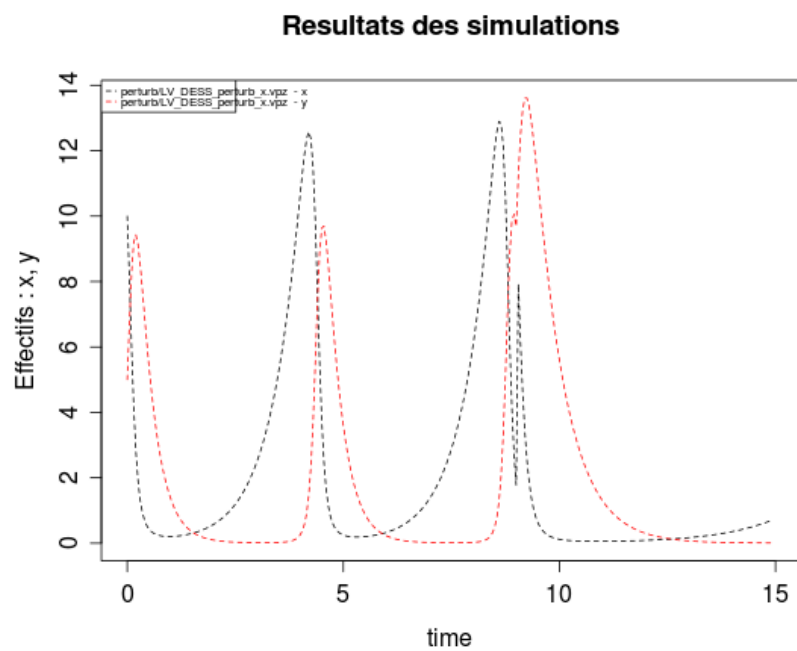
Resultats des simulations



3.4.2 Exemple intégration par Runge Kutta 4 : simulateur LV_DESS_rk4.vpz



3.4.3 Exemple de perturbation de la dynamique: simulateur LV_DESS_perturb_x.vpz



4 Extension VLE QSS::Simple : Approche par la méthode d'intégration : QSS (Quantized State System)

Cette méthode d'intégration a été proposée par E.Kofman en 2001 pour résoudre des systèmes d'équations différentielles du 1er ordre. Elle est basée sur un calcul du pas de temps d'intégration en fonction de la pente de l'équation. Cela veut dire que si on a une équation dont la pente est faible, on n'intégrera pas tous les pas de temps mais toutes les fois où x aura suffisamment varié. Cela permet d'accélérer les temps de simulation (valable uniquement si la pente faible, sinon les temps de simulation sont augmentés).

Ce qui change par rapport à l'extension DESS décrite précédemment:

- Pour le codage C++, utiliser le pattern de l'Annexe 2
- Le paramétrage, les paramètres de l'extension QSS simple :

- **precision** (type double) : le seuil de variation (lié à la pente) qui entraîne l'intégration

Parameters	Type	Overview
precision	double	0.001

- **threshold** (type double) : le seuil en dessous duquel on considère que la dérivée est nulle

Parameters	Type	Overview
threshold	double	0.001

- **dependance** (type booléen) : on indique si le modèle dépend ou non de variables externes

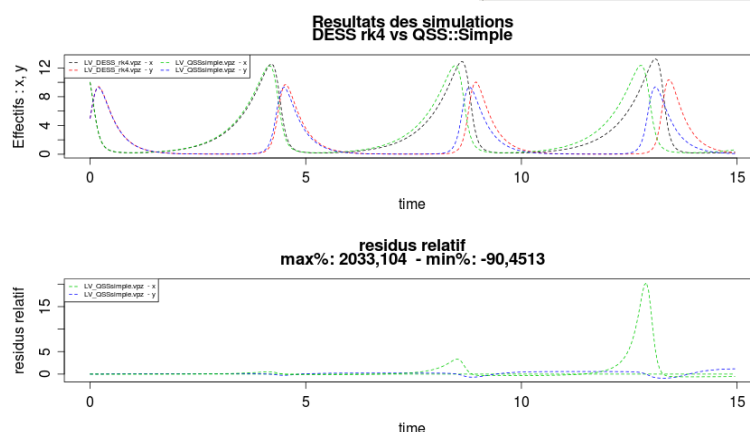
Parameters	Type	Overview
dependance	boolean	1

- **active** (type booléen) : on indique si le modèle est connecté en sortie à d'autres modèles (contrôle si des événements de sortie sont produits – ralenti la simulation si actif)

Parameters	Type	Overview
active	boolean	1

- **name** (type string) et **value** (type double) : le nom et la valeur initiale de la variable d'état gérée par le modèle

Parameters	Type	Overview
name	string	x
value		



5 Extension VLE QSS::Multiple : Approche par la méthode d'intégration : QSS (Quantized State System)

Ce qui change par rapport à l'extension QSS simple décrite précédemment:

- on peut mettre dans un modèle atomique un système d'équations différentielles (on n'est donc pas limité à une unique EDO par modèle atomique)
- Pour le codage C++, utiliser le pattern de l'annexe 3
- Ports de sortie: il y a autant de ports de sortie que de variables gérées par le système d'EDO associé au modèle.
- Le paramétrage, les paramètres de l'extension QSS simple :

- **threshold** (type double) : le seuil en dessous duquel on considère que la dérivée est nulle

Parameters	Type	Overview
threshold	double	0.001

- **dependance** (type booléen) : on indique si le modèle dépend ou non de variables externes

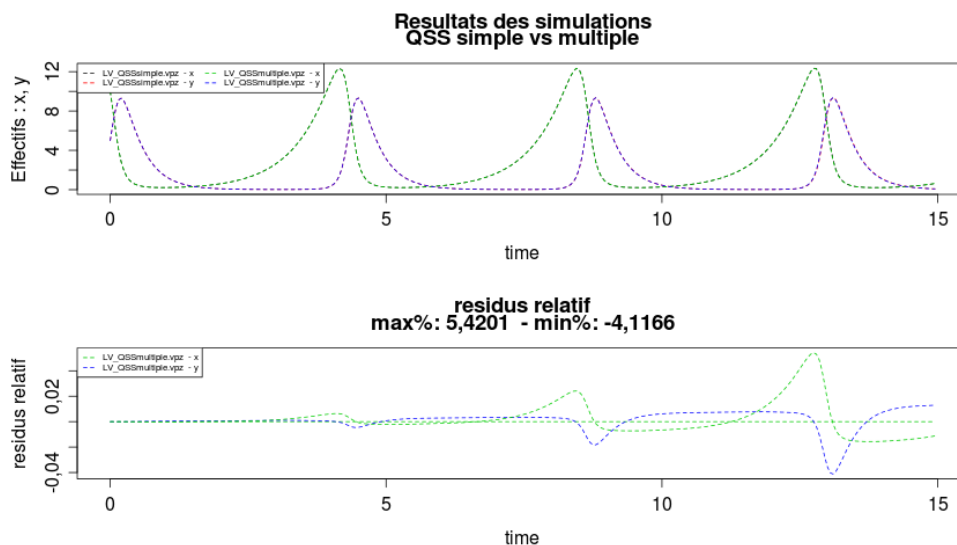
Parameters	Type	Overview
dependance	boolean	1

- **active** (type booléen) : on indique si le modèle est connecté en sortie à d'autres modèles (contrôle si des événements de sortie sont produits – ralenti la simulation si actif)

Parameters	Type	Overview
active	boolean	1

- **variables** (type set) : les initialisations de **name** (type string), **value** (type double) et **precision** (type double), sont quant à elles intégrées (dans cet ordre!) dans une unique structure **variables** prenant en charge les différentes variables d'état (type set pour chaque variable)

Parameters	Type	Overview
variables	set	((x,10,0.001),(y,5,0.001))



6 Exemples de couplage de modèles d'EDO

Simulateur avec QSS simple

LV_QSSsimple.vpz

commenter sur les ajustements numériques

---> graphique avec superposition DESS

Simulateur avec QSS multiple

LV_QSSmultiple.vpz

---> graphique avec superposition

Idem en montrant la perturbation :

---> graphique avec superposition

Exemple de couplage de QSS et DESS

LV_DESSxQSSy.vpz

LV_QSSxDESSy.vpz

---> graphique avec superposition

Pensez à inclure les pg R

Proposer l'expression en équation aux différences

7 Annexes :

Annexe 1 : Pattern de base pour l'extension DESS

$\frac{dx}{dt} = x \cdot (\alpha - \beta \cdot y)$ Avec alpha et beta constante, y gérée par un autre module.

```
#include <vle/extension/differential-equation/DESS.hpp>

namespace vd = vle::devs;
namespace vv = vle::value;
namespace ve = vle::extension;

namespace LotkaVolterra {

class LotkaVolterraDESSx : public ve::DifferentialEquation::DESS
{
public:
    LotkaVolterraDESSx(const vd::DynamicsInit& model, const vd::InitEventList& events) :
        ve::DifferentialEquation::DESS(model, events)
    {
        // Lecture des valeurs de parametres dans les conditions du vpz
        // Ces parametres ont une valeur par default utilise si la condition n'est pas definie
        alpha = (events.exist("alpha")) ? vv::toDouble(events.get("alpha")) : 0.5;
        beta = (events.exist("beta")) ? vv::toDouble(events.get("beta")) : 0.5;

        // Variables d'etat gerees par ce composant
        x = createVar("x");

        // Variables d'etat gerees par un autre composant
        y = createExt("y");
    }

    virtual ~LotkaVolterraDESSx() {}

    virtual double compute(const vd::Time& /*time*/) const
    {
        return dxdt(x(), y());
    }
}
```

```

private :
    //paramètres
    double alpha; // taux de reproduction des proies en l'absence de prédateurs
    double beta; // taux de mortalité des proies due aux prédateurs

    //variables d'etat
    Var x; // effectif des proies en fonction du temps

    //variable d'entree
    Ext y; // effectif des prédateurs en fonction du temps

    double dxdt(double X, double Y) const {
        return(X * (alpha - beta * Y));
    }
};

} // namespace LotkaVolterra

DECLARE_DYNAMICS(LotkaVolterra::LotkaVolterraDESSx)

```

Annexe 2 : Pattern de base pour l'extension QSS simple

$\frac{dx}{dt} = x \cdot (\alpha - \beta \cdot y)$ Avec α et β constante, y gérée par un autre module.

```
#include <vle/extension/differential-equation/QSS.hpp>

namespace vd = vle::devs;
namespace vv = vle::value;
namespace ve = vle::extension;

namespace LotkaVolterra {

class LotkaVolterraQSSx : public ve::QSS::Simple
{
public:
    LotkaVolterraQSSx(const vd::DynamicsInit& model, const vd::InitEventList& events) :
        ve::QSS::Simple(model, events)
    {
        // Lecture des valeurs de parametres dans les conditions du vpz
        // Ces parametres ont une valeur par default utilise si la condition n'est pas definie
        alpha = (events.exist("alpha")) ? vv::toDouble(events.get("alpha")) : 0.5;
        beta = (events.exist("beta")) ? vv::toDouble(events.get("beta")) : 0.5;

        // Variables d'etat gerees par ce composant
        x = createVar("x");

        // Variables d'etat gerees par un autre composant
        y = createExt("y");
    }

    virtual ~LotkaVolterraQSSx() {}

    virtual double compute(const vd::Time& /*time*/) const
    {
        return dxdt(x(), y());
    }

private :
    //paramètres
    double alpha; // taux de reproduction des proies en l'absence de prédateurs
```

```

double beta; // taux de mortalité des proies due aux prédateurs

//variables d'etat
Var x; // effectif des proies en fonction du temps

//variable d'entree
Ext y; // effectif des prédateurs en fonction du temps

double dxdt(double X, double Y) const {
    return(X * (alpha - beta * Y));
}

};

} // namespace LotkaVolterra

DECLARE_DYNAMICS(LotkaVolterra::LotkaVolterraQSSx)

```

Annexe 3 : Pattern de base pour l'extension QSS multiple

$$\frac{dx}{dt} = x(t) \cdot (\alpha - \beta \cdot y(t)) \quad (0)$$

$$\frac{dy}{dt} = -y(t) \cdot (\delta - \gamma \cdot x(t)) \quad (1)$$

Avec alpha, beta, delta et gamma constante.

```
#include <vle/extension/differential-equation/QSS.hpp>

namespace vd = vle::devs;
namespace vv = vle::value;
namespace ve = vle::extension;
namespace vu = vle::utils;

namespace LotkaVolterra {

class LotkaVolterraQSS : public ve::QSS::Multiple
{
public:
    LotkaVolterraQSS(const vd::DynamicsInit& model, const vd::InitEventList& events) :
        ve::QSS::Multiple(model, events)
    {
        // Lecture des valeurs de parametres dans les conditions du vpz
        // Ces parametres ont une valeur par default utilise si la condition n'est pas definie
        alpha = (events.exist("alpha")) ? vv::toDouble(events.get("alpha")) : 0.5;
        beta = (events.exist("beta")) ? vv::toDouble(events.get("beta")) : 0.5;
        delta = (events.exist("delta")) ? vv::toDouble(events.get("delta")) : 0.5;
        gamma = (events.exist("gamma")) ? vv::toDouble(events.get("gamma")) : 0.5;

        // Variables d'etat gerees par ce composant
        x = createVar(0, "x");
        y = createVar(1, "y");
    }

    virtual ~LotkaVolterraQSS() {}

    virtual double compute(unsigned int i, const vd::Time& /*time*/) const
    {
        // l'indice i correspond à celui affecté à chaque variable dans le constructeur
        // à chaque pas de temps cette méthode compute est appelée autant de fois qu'il y a de variables.
        switch (i) {
            case 0: // x
                return dxdt(x(), y());
            case 1: // y
                return dydt(x(), y());
            default:
                throw vu::InternalError(vle::fmt_(_("Compute problem with LotkaVolterraQSS, i == %1%")) % i);
        }
    }

private:
    //paramètres
    double alpha; // taux de reproduction des proies en l'absence de prédateurs
    double beta; // taux de mortalité des proies due aux prédateurs
    double delta; // taux de mortalité des prédateurs en l'absence de proies
    double gamma; // taux de reproduction des prédateurs en fonction des proies mangées

    //variables d'etat
    Var x; // effectif des proies en fonction du temps

```

```
Var y; // effectif des prédateurs en fonction du temps

double dxdt(double X, double Y) const {
    return(X * (alpha - beta * Y));
}

double dydt(double X, double Y) const {
    return(Y * (-delta + gamma * X));
}

};

} // namespace LotkaVolterra

DECLARE_DYNAMICS(LotkaVolterra::LotkaVolterraQSS)
```


Annexe 4 : utilisation de l'option **thresholds**

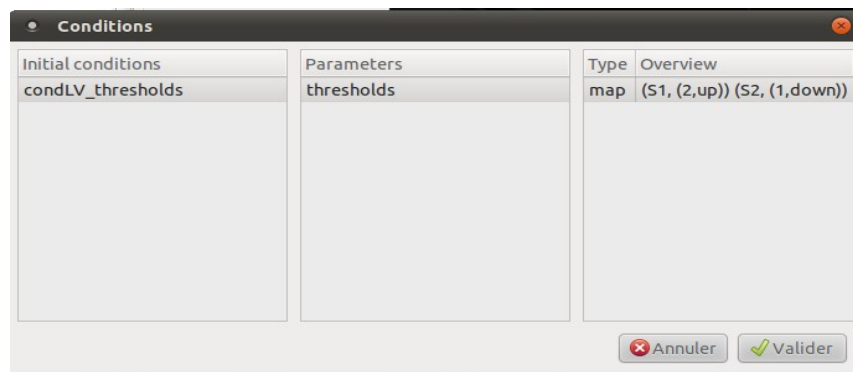
Les extensions DESS et QSS::Simple (mais pas QSS::Multiple) offrent une option pour déclencher un événement instantané sur un port de sortie **out** lorsque la variable d'état franchit un seuil montant (**S1**) et/ou descendant (**S2**).

Pour activer cette option il faut d'une part rajouter dans le vpz un port de sortie **out** sur lequel l'événement sera produit :

```
<port name="out" />
```

Et d'autre part ajouter les initialisations (**thresholds**) avec les valeurs voulues pour les fronts **S1** (montant) et **S2** (descendant) :

```
<port name="thresholds" >
  <map>
    <key name="S1">
      <set>
        <double>1.0000000000000000</double>
        <string>up</string>
      </set>
    </key>
    <key name="S2">
      <set>
        <double>0.0000000000000000</double>
        <string>down</string>
      </set>
    </key>
  </map>
```



Annexe 5: Références

- E. Kofman and S. Junco. Quantized state systems. a devs approach for continuous systems simulation. Transactions of the Society Computer Simulation International, 18(3) :123–132, 2001.
- E. Kofman. A second order approximation for devs simulation of continuous systems. Transactions of the Society Computer Simulation International, 78(2) :76–89, 2002.
- E. Kofman. A third order discrete event method for continuous system simulation. part i : Theory. In In Proceedings of RPIC'05, 2005.
- E. Kofman. A third order discrete event method for continuous system simulation. part ii : Applications. In In Proceedings of RPIC'05, 2005.
- A.J.Lotka (1925) Elements of Physical Biology reprinted by Dover in 1956 as Elements of Mathematical Biology.
- V. Volterra. Variations and fluctuations of the number of individuals in animal species living together. In Animal Ecology. McGraw-Hill, 1931. Traduit de l'édition de 1928 par R. N. Chapman).
- Zeigler, B. P., T.G.Kim and H. Praehofer “Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems, second edition Academic Press, Boston”, 2000. 510 pages