

Démonstration de VLE

Gauthier Quesnel

INRA - MIAT

13 avril 2016

VLE : Qu'est ce ?

Comment développer des modèles ?

Lexique avant la démonstration

Démonstration

Conclusion

VLE

About ...

- VLE est environnement pour développer, simuler, étudier, optimiser des modèles DEVS.
- Ce n'est pas un environnement **user friendly**.
- Il repose sur une bibliothèque dynamique appelée **VFL**
 - ▶ De développer des modèles atomiques, couplés
 - ▶ D'exécuter des simulations ou des plans d'expériences
 - ▶ Un système de paquets pour échanger
 - ▶ Extensibilité via des *plug-ins*



VLE

Techniquement ...

- VLE propose un ensemble de programmes et outils :
 - ▶ **vfl** : une bibliothèque C++ stable au cœur de l'environnement
 - ▶ **gvle** : un IDE pour développer des modèles (éditeur, générateur de code C++ etc.).
 - ▶ **rvle** ou **pyvle** : des portages pour R ou Python afin d'exploiter les ressources de ces outils ; analyses de sensibilités, optimisation, développement service Web.
 - ▶ **vle**, **cvle**, **mvle** : outils en ligne de commande pour exécuter les modèles sur SMP, cluster etc.
- Mais, il reste un environnement un peu austère car, il est fourni sans visualisation de résultats, sans interface graphique poussée etc.

VLE

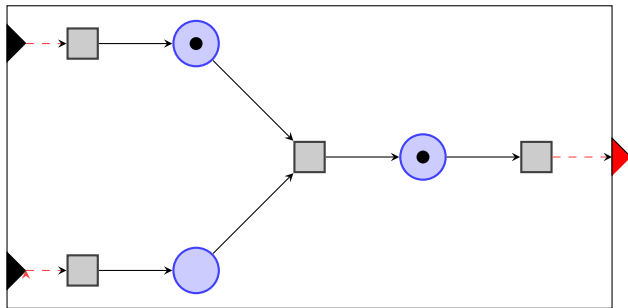
Côté DEVS...

- Un noyau DSDE (F. Barros) : combine **Parallel DEVS** et **Dynamic-Structure DEVS**
 - ▶ Gestion des conflits et événements simultanés sans utiliser de fonctions `select` (DEVS)
 - ▶ Étendre les possibilités de modélisation en ajoutant une dynamique sur la structure du modèle
- Une hiérarchie de coordinateur/simulateur mise à plat
 - ▶ Optimiser les temps calculs de simulation
- Un sous-système d'observation
 - ▶ Formaliser le principe d'observation d'un modèle DEVS
- Un ensemble de formalismes de modélisation encapsulés
 - ▶ Simplifier la modélisation avec des formalismes mathématiques ou informatiques existants

VLE

Côté DEVS... : Multimodélisation, Réseau de Petri

Encapsulation d'un formalisme intemporel comme les réseaux de Petri.



*Le réseau de Petri possède des extensions temporelle, à priorité, stochastique, etc. : **High Level PetriNet***

VLE : Qu'est ce ?

Comment développer des modèles ?

Lexique avant la démonstration

Démonstration

Conclusion

VLE

Développement de modèles atomiques

- Le développement de simulation :
 - ▶ le comportement des modèles atomiques, exécutifs ou issus de formalismes : C++.
 - ▶ la structure des modèles dans un fichier XML (VPZ) avec la définition des paramètres et des observations.

Classe de base (version simplifiée) :

```
class vle::devs::Dynamics {
    Time timeAdvance() const;
    void internalTransition(const Time& time);
    void externalTransition(const Time& time,
                           const ExternalEventList& lst);
    void output(const Time& time,
               ExternalEventList& out) const;
    void confluentTransition(const Time& time,
                            const ExternalEventList& lst);
    Value* observation(const ObservationEvent& event) const;
};
```



VLE

Développement de modèles issus de sous-formalismes

- Nous proposons une API simplifiée
- Nous proposons des générateurs de code C++

Par exemple, avec le formalisme des équations aux différences, nous fournissons une seule fonction `compute`. L'api des modèles atomiques est cachée.

Par exemple :

```
class MyModel: public vle::extension::DifferenceEquation {  
    virtual double compute(const Time& time)  
    {  
        x = y(-1) + z(-1);  
        y = z(-1);  
    }  
};
```

VLE

Développement de modèles à l'aide GVLE

Pour développer des codes sources, la structure du modèle, les conditions initiales, les observations, les cadres expérimentaux :

The screenshot displays the GVLE 1.1.0-dev application window. The title bar reads "GVLE 1.1.0-dev - vle.examples - petrinet_nand2.vpz". The menu bar includes "File", "Edit", "Tools", "Project", "View", "Simulation", "Zoom", and "Help". The toolbar contains icons for "Annuler", "Retablir", "Select", "Add Models", "Add Links", "Add Coupled", "Delete", "Zoom", and "Question".

On the left, a file explorer shows a list of files under "vle.examples", with "petrinet_nand2.vpz" selected. The main workspace shows a PetriNet diagram with components: "PetriNet" (containing places p1 and p2), "counter", "beepB", and "beepA".

On the right, a "Model" panel shows a tree structure: "top" (expanded) containing "PetriNet", "beepA", "beepB", and "counter". Below it is an empty "Class" panel.

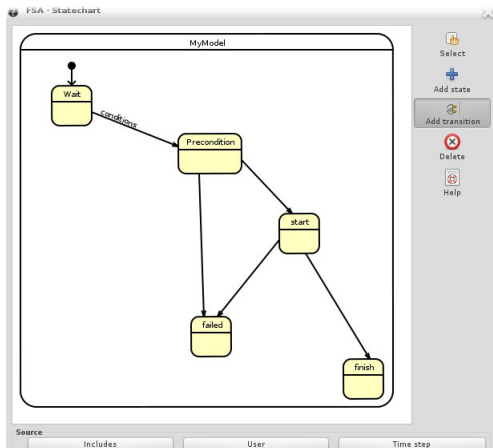
At the bottom, a console window displays the following error messages:

```
-- The VLE_HOME directory not assigned try default
-- The VLE_HOME directory not assigned try default
-- The VLE_HOME directory not assigned try default
-- The VLE_HOME directory not assigned try default
-- The VLE_HOME directory not assigned try default
-- The VLE_HOME directory not assigned try default
-- The VLE_HOME directory not assigned try default
-- The VLE_HOME directory not assigned try default
-- The VLE_HOME directory not assigned try default
-- The VLE_HOME directory not assigned try default
CMake Error at /home/gqth/vle/pkg/vle.examples/cmake/VleCheckPackageConfig.cmake:103 (message):
  Package 'vle.extension.calldevs' not found
Call Stack (most recent call first):
  CMakeLists.txt:35 (VLE_CHECK_PACKAGE)
```

VLE

Développement de modèles à l'aide GVLE

Un *plug-in* graphique pour développer un automate à états :



VLE

Développement de modèles à l'aide GVLE

Génération de code source :

GVLE 1.1.0-dev - tmp - MyModel.cpp

File Edit Tools Project View Simulation Zoom Help

Annuler Rétablir Select Add Models Add Links Add Coupled Delete Zoom Question

tmp

- + cmake
- + data
- + doc
- + exp
 - CMakeLists.txt
 - empty.vpz
 - output
- src
 - CMakeLists.txt
 - Simple.cpp
 - MyModel.cpp
- + test
 - Authors.txt
 - CMakeCPack.cmake
 - CMakeLists.txt
 - Description.txt
 - License.txt
 - News.txt
 - Readme.txt

```

23 MyModel(
24     const vd::DynamicsInit& init,
25     const vd::InitEventList& evts)
26     : vf::Statechart(init, evts)
27 {
28     @@begin:constructorUser@@
29
30     @@end:constructorUser@@
31     // structure
32     states(this) << Precondition << Wait << failed << finish << start;
33
34     transition(this, Wait, Precondition) << event("conditions");
35     transition(this, Precondition, start);
36     transition(this, Precondition, failed);
37     transition(this, start, failed);
38     transition(this, start, finish);
39
40
41     initialState(Wait);
42 }
43
44 virtual ~MyModel()
45 {}
46
47 ..
  
```

Model

- Top model

Class

VLE

Systèmes de paquets

La bibliothèque VFL est étendue par des *plug-ins* : simulation, graphique, de sortie. Pour gérer ces codes nous proposons un système de paquets :

- Un paquet contient des sources C++, des données d'entrée, des documents etc.
- Un paquet peut dépendre d'autre paquets.
- Une distribution de paquets sources (inspirée de R et Debian).

Techniquement :

```
cd $HOME # le paquet source
vle-1.1 -P toto create
vle-1.1 -P toto configure build
```

```
cd $HOME/.vle/pkgs-1.1 # le paquet binaire (distribuable)
ls
toto
```

VLE : Qu'est ce ?

Comment développer des modèles ?

Lexique avant la démonstration

Démonstration

Conclusion

VLE

Quelques termes

- dynamics** la classe de base pour le développement de modèle DEVS.
- executive** la classe de base pour développer des modèles à structure dynamique (possède la même interface que la classe `dynamics`).
- extension** une API spécifique (souvent sous forme d'un héritage vers la classe `dynamics`) qui encapsule un formalisme dans le but de simplifier son utilisation.
- value** une hiérarchie de classes C++ pour stocker, échanger des données de types simples (entier, réel, chaînes ou complexes vecteur, tableau, dictionnaire).

VLE

Quelques termes

vpz application XML pour encoder :

- la structure du modèle DEVS (modèles, ports, connexions etc.)
- la dynamique (*plug-in*) attachée a chacun des modèles
- les initialisations
- les observations
- son plan d'expérience, etc.
- une structure de modèle DEVS instanciable depuis un modèle *executif*
- la date de début et de fin de simulation

VLE

Quelques termes

plug-ins les codes des modèles sont compilés et distribués dans des modules (dll, so ou dylib).

paquet un dossier ou une archive pour stocker le code source ou binaire des modèles, des données, des documentations, des plug-ins graphiques, de sortie etc. Un paquet peut dépendre d'autres paquets.

distribution un ensemble de paquets interdépendants. `vle -R update/search/install`.

vpz application XML pour encoder la structure du modèle DEVS, ses initialisations, ses observations, son plan d'expérience, etc.

VLE : Qu'est ce ?

Comment développer des modèles ?

Lexique avant la démonstration

Démonstration

Conclusion

Démonstration

Un modèle *Lotka Volterra* très simple :

$$\dot{x} = x\alpha - \beta xy$$

$$\dot{y} = -\gamma y + \delta xy$$

Un exécutif, une grille, une classe de modèle :

$$\dot{x}_2 = -2\kappa x + \sum_{v=1}^2 \kappa x_v$$

$$\dot{x}_3 = -3\kappa x + \sum_{v=1}^3 \kappa x_v$$

$$\dot{x}_4 = -4\kappa x + \sum_{v=1}^4 \kappa x_v$$

VLE : Qu'est ce ?

Comment développer des modèles ?

Lexique avant la démonstration

Démonstration

Conclusion

Conclusion

VLE 1.0, 1.1 et 1.3

- + Très stable et en évolution constantes pour les utilisateurs
 - ▶ v1.0 maintenue depuis 2011
 - ▶ v1.1 maintenue depuis 2014
 - ▶ v1.3 bientôt disponible avec une nouvelle interface graphique et un nouveau *web*
 - ▶ Noyau de simulation relativement rapide
 - ▶ Travail collaboratif assez aisée
- Trop stable pour des travaux plus théoriques ou pratiques
 - ▶ Contraint par les types des données échangées entre modèles
 - ▶ Contraint par la représentation du temps
 - ▶ Contraint par la structure des données
 - ▶ Mono-processeur

Conclusion

Echll : un nouveau noyau de simulation pour VLE 2.x

Un noyau C++ léger, multi-simulateurs abstraits avec une meilleure séparation des codes.

- DEVS, DTSS, PDEVs, DSDE, HFSS, DEV&DESS
- Transfert entre modèles : plus de souplesse (template)
- Modélisation du temps : plus de souplesse (template)
- Modèles couplés génériques et spécifiques
- Modèle et simulation *copyable*, *moveable*
- Modèles distribués : MPI ou *thread*
- Modèles parallélisés : MPI et *thread*

Conclusion

Echll : un nouveau noyau de simulation pour VLE 2.x

- 2016 pour une première intégration dans une version de VLE ?
- Aucune API n'est encore définitive
- Une licence plus souple que la GPL v3.

Venez tester, regarder ou même discuter sur ce que vous aimeriez avoir ...