Ingénieur en Sciences Informatiques

Rapport de Stage 4ème année

Extension logicielle de GVLE

Stagiaire :
Geneviève Cirera (SI4)

Maître de stage :
Patrick Chabrier

Entreprise INRA (Toulouse)

16 juin 2014 - 20 septembre 2014

Résumé

Remerciements

Table des matières

1 Introduction

1.1 Sujet de stage

L'équipe-projet RECORD [1] au sein de l'INRA de Toulouse a pour but de développer un logiciel écrit en C++ de modélisation et de simulation des agro-écosystèmes appelé VLE [2].
Ce logiciel modélise des individus grâce au formalisme DEVS [3], d'équations différentielles et notamment du plugin Forrester. Ce plugin permet la modélisation d'un individu sous les règles du formalisme de forrester, c'est-à-dire de générer des classes C++ correspondant à la description de l'individu.
Ce stage a pour but de faire une extension de ce plugin afin de pouvoir cloner un ou plusieurs individus et permettre à l'utilisateur de prendre des décisions en fonction de l'état du système et/ou du temps qui s'écoule.

1.2 Contexte

Ce stage s'inscrit dans mon cursus universitaire pour clôturer ma 4ème année en ingénieurie informatique à Polytech'Nice Sophia. Durant ce stage, j'ai travaillé de manière indépendante au sein de l'équipe-projet RECORD sur un plugin global du logiciel VLE qui utilise le formalisme DEVS.
Ce formalisme est un formalisme modulaire et hiérarchique pour la modélisation, la simulation et l'analyse de systèmes complexes. Il permet de donner une dynamique à un modèle qui peut être amplifiée si celle-ci est déclaré comme exécutive. La différence est que la dynamique prend en considération des évènements en

1. Rénovation et CooRDination de la modélisation de cultures pour la gestion des agro écosystèmes.
2. Virtual Laboratory Environment
3. Discrete Event System Specification

entré, effectue une ou plusieurs actions et renvoye des évènements en sortie au cours du temps alors que l'exécutive peut modifier, créer, supprimer et avoir d'autres actions sur les modèles en plus de toutes les actions réalisées par la dynamique.

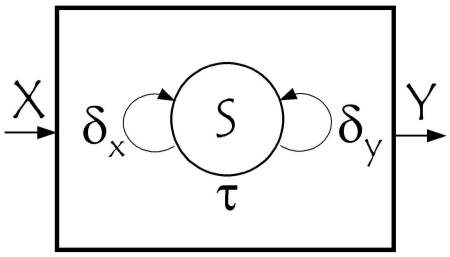

FIGURE 1.1 – Dynamique DEVS

Dans ce schéma représentant le fonctionnement de DEVS, les entrées du modèle sont représentées par X, les sorties par Y, et δ_x la dynamique lors de la réception d'évènement, δ_y la dynamique interne du modèle c'est-à-dire ce qui arrive régulièrement au cours du temps τ. S est l'état du système.

2 Description du travail proposé

2.1 Besoin

Actuellement, le logiciel VLE permet de créer des individus un à un leur associer une dynamique, de les relier entre eux et de lancer des simulations. La dynamique de plusieurs individus peut être la même, et peut varier selon les conditions expérimentales qui décrivent les valeurs des paramètres si l'utilisateur les édite.
Cependant, il est nécessaire que l'utilisateur édite chaque individu un à un. Comment faire pour modéliser des troupeaux entiers sans avoir ce travail répétitif ?
C'est pourquoi, il était nécessaire de créer une extension du logiciel pour ces cas là. Une extension qui permettrait la création, suppression et prise de décision sur les individus de manière simple et instinctive pour l'utilisateur.
L'utilisateur n'aurait donc pas besoin de programmer en C++ ni d'éditer des graphes ? ? ? et réutiliserait le plugin Forrester pour définir les classes d'individus qui serveront de base pour créer chaque individu lors de la simulation.

Deux tp m'ont été donné afin de me donner une idée des fonctionnalités attendues et de diriger la conception et le développement de mon plugin. Ces exemples ont été réalisés avec le logiciel Modelmaker. Il me fallait alors adapter les fonctionnalités proposées par Modelmaker afin que ces tp soient réalisables avec VLE.

Le premier tp avait pour but de créer 5 modèles dont chacun d'eux possédait un compartiment, le premier se vidant dans le deuxième et si le deuxième atteingnait une certaine valeur, se vidait immédiatement dans un troisième compartiment. Chaque modèle ayant des paramètres de vidange différents.
D'autres part, il devait être possible d'offrir à l'utilisateur la possibilité d'observer une ou plusieurs valeurs en particulier. Dans ce cas précis, la somme de tous les

compartiments qui se vident.

Le second tp était plus développé. Il s'agissait de faire naitre 10 cellules à t=10, faire croître leur poids suivant des paramètres distints puis à t=30, repérer la plus grosse cellule et faire décroître les 9 autres.
Lorsque la cellule la plus grosse atteint 0.99, 10 autres naissances sont lancées. Si une cellule a un poids inférieur à 0.1, la tuer.

2.2 IBM

L'objectif est de pouvoir modéliser des individus qui ont chacun une dynamique qui peut être la même ou pas afin de lancer des simulations sur une durée déterminée par l'utilisateur.
Dans VLE, chaque individu est représenté par une ou plusieurs "boite" appelé modèle, associée chacune à une dynamique et il est possible d'étudier le comportement de cet individu lors de la simulation. C'est ce que nous appelons l'individu-centré.
Dans le cadre du stage, nous souhaitons démultiplier ces individus afin de pouvoir réaliser des simulations à l'échelle de groupe, de troupeau par exemple, en gardant une indépendance entre chaque individu. Nous parlerons donc ici d'IBM, Individual Based Model.

2.2.1 IBM Conceptuel

Chaque individu est un système d'équations différentielles défini par une classe C++. Cette classe est appelée classe d'individu. C'est elle qui doit être défini par l'utilisateur puis qui servira de base pour créer tous les individus de ce type là. C'est pourquoi chaque individu de même type, a la même dynamique, seuls les paramètres peuvent diverger entre eux si l'utilisateur souhaite en modifier un ou plusieurs.
Comme dit précédemment, chaque individu est indépendant, ils n'intéragissent pas directement les un avec les autres. Ils doivent communiquer par l'intermédiaire d'un

même controleur auquels ils sont tous connectés.
Chaque port de sortie de chaque individu est relié aux ports d'entrée du controleur, ce qui permet à ce dernier de recevoir des évènements, les traiter et envoyer des réponses adéquates par l'intermédiaire de ses ports de sorties reliés à chaque individu.
Lors de la simulation, la dynamique du controleur se met en marche. Elle initialise tous les individus, exécute sa dynamique interne, reçoit les évènements externes et envoie des réponses aux individus. Durant la simulation, le controleur peut à tout moment, créer un nouvel individu, en supprimer ou en modifier les paramètres grâces aux évènements qu'il reçoit en entrée et qu'il peut envoyer en sortie.
Par exemple :

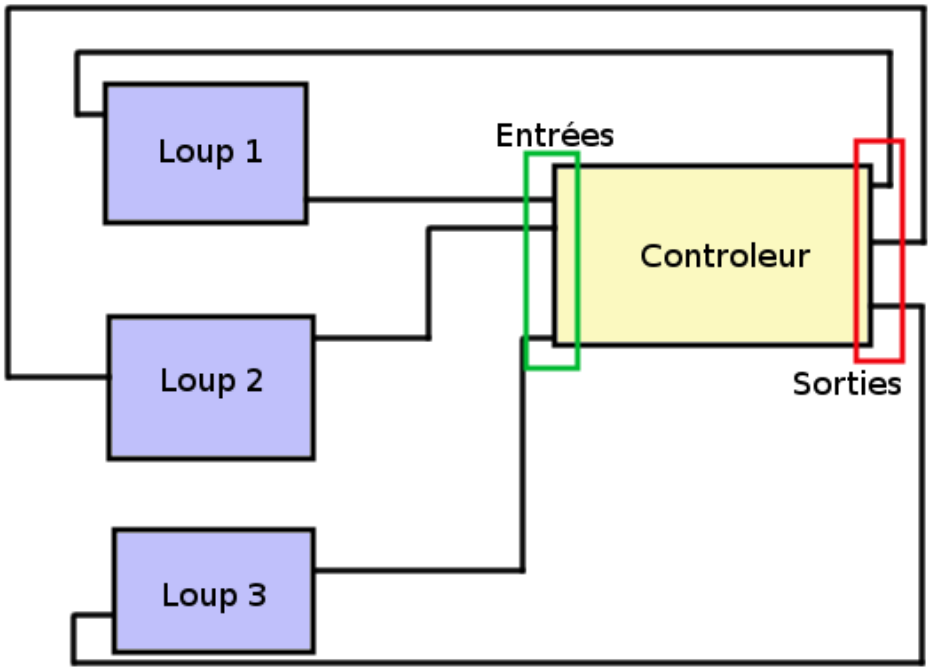

FIGURE 2.1 – Trois Loups créés par le controleur lors de la simulation

Le controleur crée les individus "Loup 1", "Loup 2" et "Loup 3" qui sont bien indépendants les uns des autres et tous reliés au controleur comme expliqué au paragraphe précédent.

2.2.2 IBM dans VLE

Dans VLE, chaque système est représenté par un Vpz qui est un fichier xml où est décrit tout le système. Modèles présents (individus), conditions expérimentales (valeurs des paramètres), classes d'individus, ports d'entrées et de sorties...
Dans ce Vpz, un controleur est obligatoirement présent, il se crée automatiquement lors de la première ouverture du plugin IBM.
Ensuite, les modèles souhaités par l'utilisateur sont créés lors de la simulation par le controleur grâce aux classes d'individu définies dans le vpz.
Par exemple :

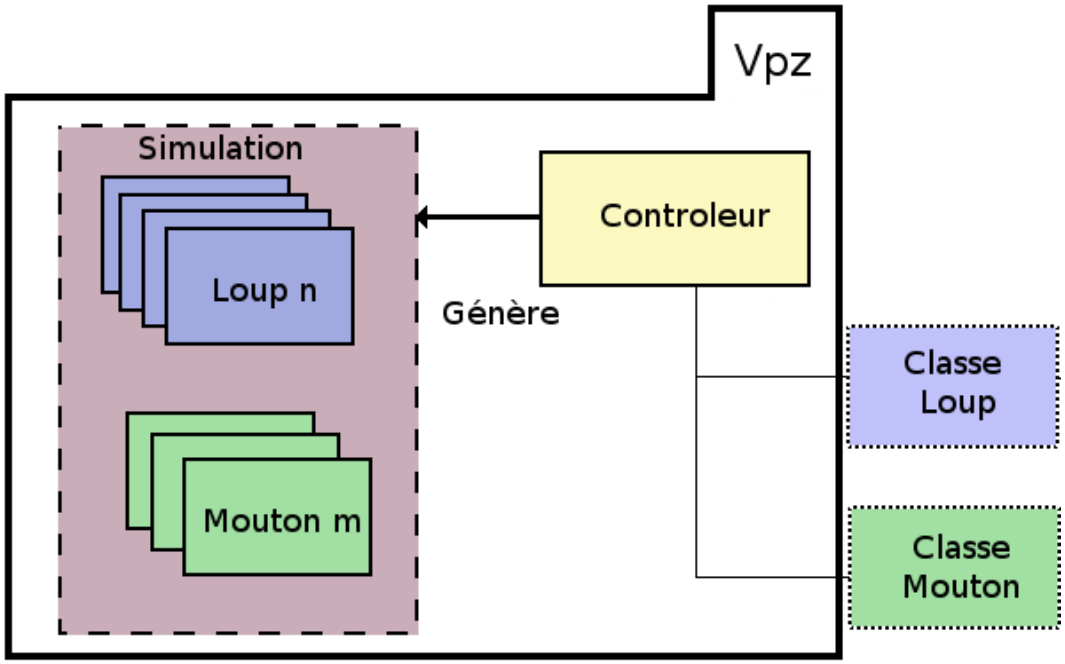

FIGURE 2.2 – Composition d'un Vpz

Ici, le controleur crée n Loup et m Mouton lors de la simulation, mais seul le modèle "Controleur" est présent dans le vpz. Le controleur est cependant lié aux classes "Loup" et "Mouton" afin de pouvoir créer les individus.

2.3 Fonctionnalités du plugin

À son ouverture, le plugin récupère toutes les classes d'individu déjà présente dans le Vpz. Il offre ensuite la possibilité d'ouvrir le plugin Forrester afin de créer d'autres classes et modifier ou supprimer les classes existantes.
Afin de manipuler les individus lors de la simulation, le plugin propose un champs de texte afin que l'utilisateur exprime ces besoins par l'intermédiare d'un petit langage simple appelé Lua et quelques extensions que j'aurai développées.
Ces besoins peuvent être multiple, ils peuvent avoir un effet sur les individus, création, suppression, modification ou bien renvoyer des informations, valeur d'une variable, nombre d'individus, identifiant d'un individu...
D'autre part, le plugin crée automatiquement un exécutive appelé "Controleur" à son ouverture. Controleur qui sera chargé de manipuler les individus selon le script qu'aura écrit l'utilisateur, et la dynamique DEVS.

3 Description du travail réalisé

3.1 Architecture

Afin de répondre au besoin, il a été décidé de développer un plugin global appelé vle.extension.ibm.
Les raisons de cette décision sont variées. L'extension a besoin d'accéder à toutes les classes d'individu déjà créées par l'utilisateur précedemment dans le Vpz et qu'il puisse en créer de nouvelles depuis le plugin, afin que le clonage d'individu soit faisable. Le plugin a aussi besoin d'avoir accès au Vpz entier (conditions expérimentales, ports,...) et donc de VLE, car il intéragit avec ce dernier lors de la simulation.
De plus, une interface graphique du plugin devait être développée afin d'en faciliter l'utilisation, c'est pourquoi l'accès à GVLE [1] était également indispensable.

Le plugin développé permet le lancement du plugin Forrester afin de créer les classes d'individu et est en lien avec un proxy permettant la communication entre le langage de script proposé à l'utilisateur et le controleur.

un schéma présentant ou se trouve quoi!!!

3.2 Controleur

L'élément principal du système IBM est le controleur, il a pour but de gérer tous les évènements qui voyagent dans le système en les faisant passer par lui. Il hérite de nombreuses classes ce qui lui permet d'avoir les capacités d'une dynamique,

1. GUI pour VLE

d'un exécutive et d'un GenericAgent.

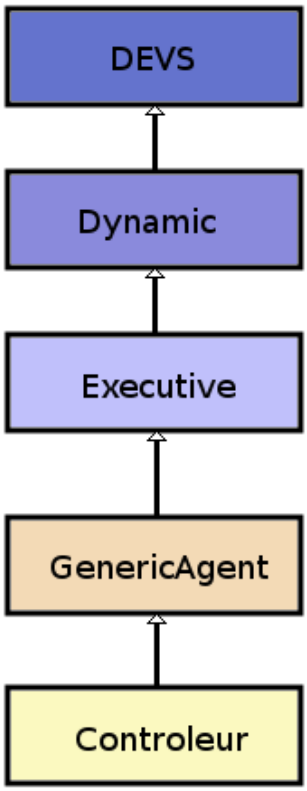

FIGURE 3.1 – Graphe d'héritage du controleur

Il se crée automatiquement à la première ouverture du plugin IBM. Ces capacités sont multiples grâce à la hiérarchie d'héritage dont il est formé.

3.2.1 Une dynamique

Il possède la capacité d'une dynamique DEVS, recevoir des évènements par ses ports d'entrées, envoyer des évènements par ses ports de sorties, exécuter un programme interne à chaque pas de temps (transition interne), gérer les évènements externes (transition externe). Voici un résumé de sa dynamique.

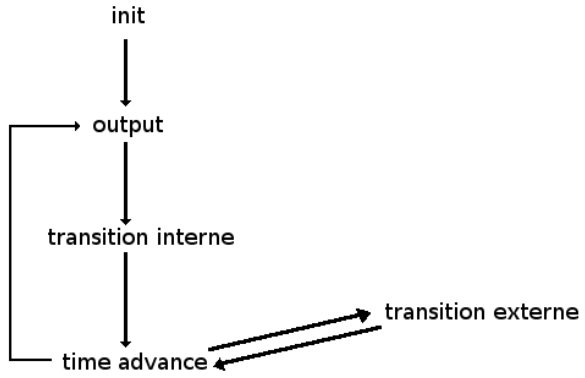

FIGURE 3.2 – Dynamique du controleur

Le controleur s'initialise dans init, puis exécute en boucle output, transition interne et time advance.
Output envoie les évènements par les ports de sortie, transition interne exécute les tâches régulières du controleur à chaque pas de temps t rendu par time advance.
La transition externe quant à elle s'exécute lors de la réception d'évènements, les traite puis la dynamique continue son cycle depuis time advance.

3.2.2 Un executive

Héritant d'Exécutive, le controleur a le pouvoir de créer des modèles mais aussi de les détruire et les modifier.
En effet, afin de créer des clones et les manipuler suivant les souhaits de l'utilisateur, le controleur a besoin de ces capacités.
Pour cela, un petit langage sera mis à la disposition de l'utilisateur afin qu'il puisse programmer les agissement du Controleur durant la simulation.
Ce petit programme est une chaine de caractère exécutée par le Controleur au bon moment grâce à sa dynamique afin de respecter les dates d'agissement souhaitées par l'utilisateur.

3.2.3 Le scheduler

Le Controleur hérite également de GenericAgent, ce qui lui donne accès à une sorte de calendrier. Il peut ainsi créer des évènements, les enmagasiner et exécuter les scripts envoyés par ces évènements à la bonne date.

3.3 GUI

Une interface simple a été développée pour l'utilisateur. Deux champs, la colonne de droite en vert qui liste toutes les classes d'individu présentes dans le Vpz ouvert et en rouge la zone de texte permettant l'écriture du script lua.
En survolant le nom d'une classe une infobulle s'ouvre pour afficher la liste des paramètres et compartiments de la classe en question, ce qui permet à l'utilisateur de programmer plus facilement. Il a directement accès à toutes les variables potentiellement modifiable au cours de la simulation.
Par un clique-droit sur la colonne des classes, un menu déroulant s'affiche où l'utilisateur peut décider de créer une nouvelle classe, modifier la classe sélectionnée ou la supprimer.
Ici l'exemple du tp Modelmaker sur le remplissage et vidange des compartiments :

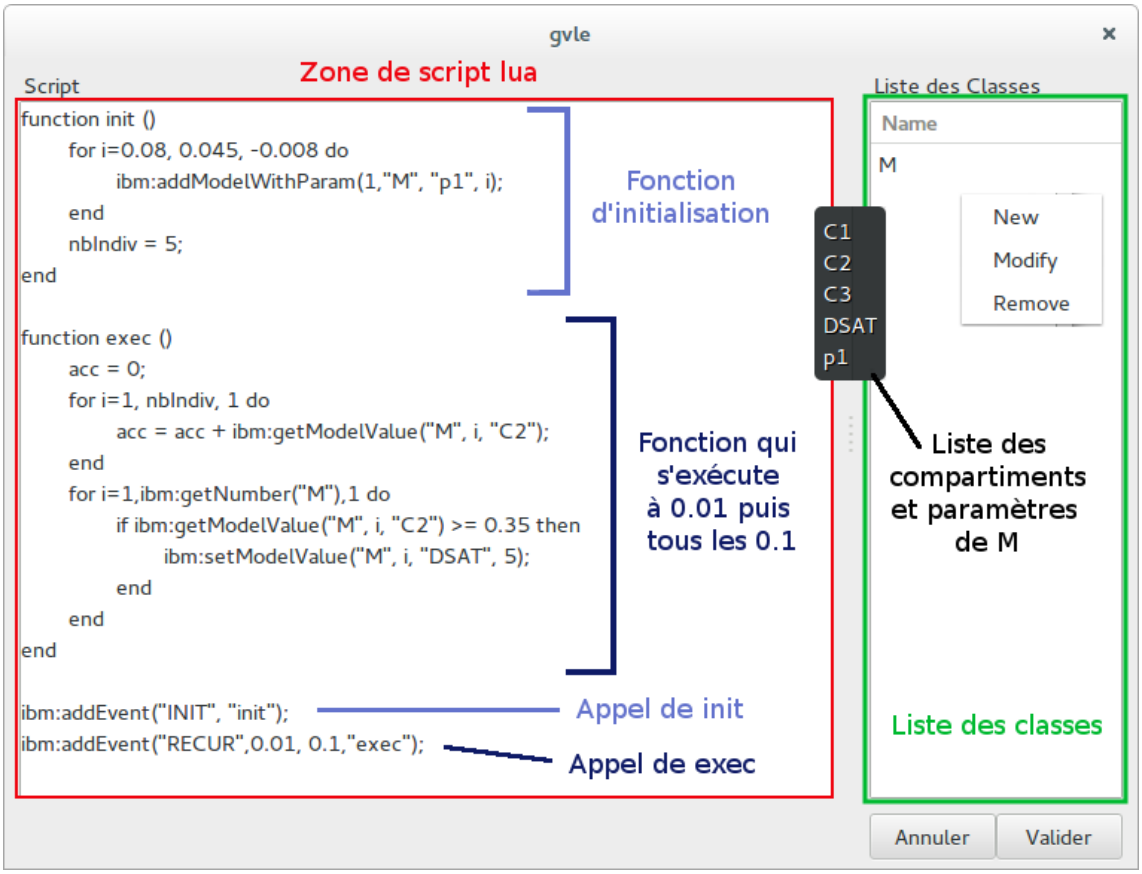

FIGURE 3.3 – Interface graphique du plugin IBM

3.4 Lua & addons

Lua est un langage de script libre créé en 1993. Il a été conçu de manière à pouvoir être embarqué et augmenter les possibilités du système hôte.
Dans notre cas, il sert d'interprète entre ce que l'utilisateur souhaite faire et le controleur. Ainsi, on peut profiter de tout le langage lua, plus quelques fonctionnalitées que j'ai développées qui permettent certaines actions précises sur les individus.

3.4.1 Mise en oeuvre

Un script lua peut très facilement être exécuté depuis un programme C++.
On retrouvera les principales structures for, while, if, l'utilisation de variable... que l'utilisateur pourra se servir dans son script.
Mais ce langage ne permet pas à l'utilisateur d'effectuer des actions sur les individus. C'est pourquoi, des extensions à ce langage ont été développées et sont exécutées grâce à l'intermédiare d'un proxy.

Dans l'exemple ci-dessous, le controleur récupère le script lua puis l'exécute. Comme addModel n'existe pas dans le langage lua, le proxy comblera cette anomalie en appelant la fonction du controleur associée à addModel à chaque fois qu'elle sera appelé. Dans ce cas, trois fois.

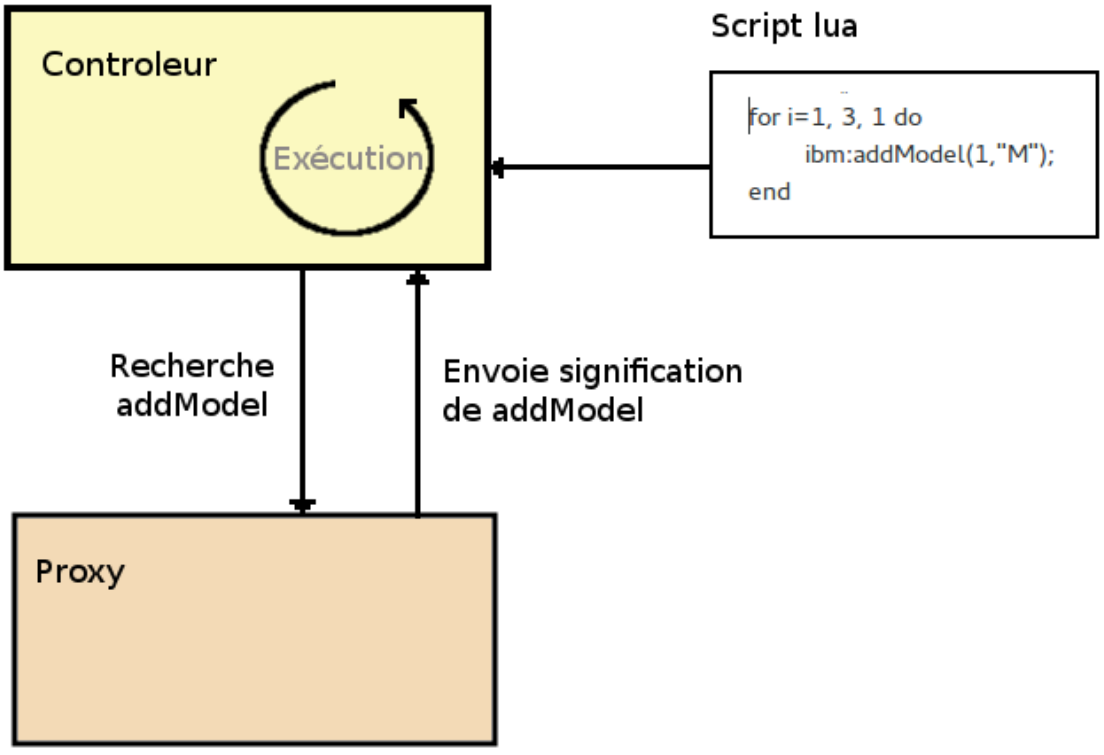

FIGURE 3.4 – Exécution d'un script lua depuis le controleur

3.4.2 Les extensions

Chaque extension du langage lua ont comme préfixe “ibm :”. Elles permettent d'avoir des effets sur les modèles, d'obtenir des informations ou lancer des évènements.
Voici la liste des extensions développées qui ont des effets sur les individus :

- addModel(n,“M”)
  Ajoute n model de type “M”.
- delModel(“ModelName”)
  Supprime le modèle dont le nom est “ModelName”.
- addModelWithParam(“Class”, n, “Param1”, value1, “Param2”, value2, ...)
  Ajoute n model de type “Class” en modifiant les paramètres de telle manière à ce que Param1 = value1 et Param2 = value2. Le nombre de paramètre à modifier est indéfini.
- setModelValue(class, i, varName, varValue)
  Modifie la valeur de “varName” par “varValue” du i$^{\text{ème}}$ modèle de la classe “class”.

Les extensions qui permettent d'avoir des informations sur les individus.

- getModelValue(“modelName”, “varName”)
  Renvoie la valeur de la variable “varName” du modèle dont le nom est “modelName”.
- getModelValue(“class”, i, “varName”)
  Renvoie la valeur de la variable “varName” du i$^{\text{ème}}$ modèle de la classe “class”.
- getModelName(class,i)
  Renvoie le nom du i$^{\text{ème}}$ modèle de la classe “class”.
- getNumber(“class”)
  Renvoie le nombre d'individu de la classe “class”.
- getTime()
  Renvoie la date actuelle durant la simulation.

L'extension permettant de lancer des évènements est addEvent. Cette fonction peut être de trois types, “INIT”, “SINGLE” ou “RECUR” précisé en premier paramètre.

L'évènement lancé se traduit par l'exécution d'une fonction écrite par l'utilisateur dans le script. Cette fonction est toujours précisé en dernier paramètre de addEvent.
"INIT" permet de lancer un premier script à l'initialisation de la simulation.
"SINGLE" permet de lancer un évènement ponctuel dans la simulation, c'est-à-dire l'exécution d'une fonction lua à un temps déterminé par l'utilisateur. La syntaxe sera addEvent("SINGLE", date, "fonction").
"RECUR" permet de lancer un évènement qui s'exécutera à une certaine fréquence depuis une date déterminée par l'utilisateur. On aura addEvent("RECUR", date, frequence, "fonction").

3.5 Déroulement du stage

3.5.1 Découverte

Les premiers jours ont été destinées à l'installation de toutes les parties nécessaires au fonctionnement du logiciel et à la compréhension de ce dernier, à quoi sert-il et comment fonctionne-t-il du point de vue de l'utilisateur.

Durant cette phase, j'ai lu la documentation associée à l'utilisation du logiciel afin de comprendre comment était représenté un individu, comment une simulation fonctionnait, l'envoie de messages entre modèles...
De plus, j'ai effectué un TP destiné à apprendre à utiliser le plugin Forrester, lancer des simulations et observer les valeurs de certains paramètres.
J'ai également réalisé à la main le clonage d'un individu en modifiant un de ses paramètres afin d'avoir une idée plus claire de ce qu'on attendait de moi durant ce stage.

3.5.2 Langage de script

Il était dès le départ évident que l'utilisateur devra programmer une petite partie afin de décrire la vie des individus qu'il aura créé.

J'ai tout d'abord commencé à développer les extensions de lua décrites précédemment par l'intermédiaire d'un parser en C++ dans le controleur sans le langage lua. Et seules ces fonctionnalités pouvaient être appelées.
Cette méthode était très limitée, c'est pourquoi l'insertion d'un langage existant était nécessaire.
Lua, un langage de programmation très simple, se mélange parfaitement au C++ et permet une plus grande liberté à l'utilisateur. Créant un proxy afin de faire l'intermédiaire entre lua et les fonctions C++, l'utilisateur peut profiter de tous les avantages et multiplier ces possibilités.
Après la mise en place du proxy, le développement de nouvelles extensions s'est faite sans difficultés.

3.5.3 La dynamique

Après avoir pris connaissance du logiciel et ses fonctionnalités, il m'a fallu comprendre la dynamique de fonctionnement c'est-à-dire la dynamique DEVS. Elle se traduit par l'ordre dans l'exécution des fonctions dans une dynamique, reception d'évènement, envoie d'évènement, transition interne et externe...
A la base, l'utilisateur devait programmer deux script, un pour l'initialisation et un autre qui s'exécutait en continu en transition interne.
Cependant, j'ai été confronté à un problème. L'utilisateur ne pouvait pas décider d'exécuter une commande à un moment déterminé au cours de la simulation, par exemple si l'utilisateur souhaite faire naître deux loups à la moitié de la simulation. Pour cela, il devait passer par l'intermédiaire de booléen, ce qui compléxifiait énormément le code et le ralentissait.
Afin de résoudre se problème, j'ai mis en place un Scheduler. Il permet l'ordénation d'évènements par le temps auxquel ils doivent s'exécuter puis l'envoie de ses derniers à la bonne date lors de la simulation.
Grâce au scheduler, l'utilisateur simplifie son code en ne faisant que des fonctions lua puis appelle ses dernières dans l'ordre et la fréquence qu'il souhaite grâce à la commande d'ajout d'évènement addEvent.

3.5.4 Bugs & tests

Au cours du développement, de nombreux bugs se sont manifestés. Leurs résolutions étaient parfois évidente, mais il pouvait être parfois plus difficil de trouver leurs sources.
J'ai donc utilisé gdb pour essayer de repérer les sources d'erreurs plus facilement, mais parfois ce système ne fonctionnait pas car c'était gdb lui-même qui provocait les bugs.
D'autres bugs n'étaient visibles que lors de l'analyse des résultats obtenus après simulation. Pour remonter jusqu'aux erreurs, il a fallu songer à des tests, des scripts lua plus simples mais qui pourtant reproduisaient le bug.
Après avoir réalisés ces tests, les plus pertinents ont été insérés dans le paquet vle.extension.ibm avec les TP donnés à titre d'exemple des besoins, afin que l'utilisateur en téléchargeant le plugin, puissent avoir des exemples d'implémentations simples.

3.5.5 Outils & organisation

J'ai développé le plugin vle.extension.ibm par l'intermédiaire de gedit de manière totalement libre grâce à au gestionnaire de version GIT et le dépôt Github.
Nous avons fait une prémière réunion avec l'équipe-projet Record afin de clarifier ce que je devrais développer en début de stage. Des réunions très régulières avec mon tuteur ont eu lieu dans le but de vérifier mon avancée et envisager les possibilités futures de développement.
De plus, j'avais à ma disposition des User Stories qui décrivaient ce que l'utilisateur devait pouvoir réaliser avec le plugin afin de me diriger dans mon développement. Ces User Stories étaient progressives, elles partaient de fonctionnalités simples comme afficher le bouton de démarrage du plugin à la mise en place de Scheduler. Afin de vérifier que toutes les fonctionnalités étaient bien mise en place, j'ai modélisé les systèmes demandés dans les deux TP des besoins, croissance des cellules et vidange des compartiments.

4 Conclusion