Documentation technique
à l’usage du site
unit-amesim.insa-rouen.fr

**23/07/2010**

Allison Lacroix & Frédéric Ngo

[Tapez le résumé du document ici. Il s’agit généralement d’une courte synthèse du document.]

Sommaire

[1. Simulation AMESim 1](#_Toc267921081)

[1.1. Comportement longitudinal d’un véhicule 1](#_Toc267921082)

[2. Applet Java 1](#_Toc267921083)

[2.1. Exportation des données AMESim 1](#_Toc267921084)

[2.2. Bibliothèques Java additionnelles 2](#_Toc267921085)

[2.3. Arborescence 4](#_Toc267921086)

[2.4. Fichier index.html 5](#_Toc267921087)

[2.5. Modifications à apporter au code source Java 6](#_Toc267921088)

[3. Structure du site Web 8](#_Toc267921089)

# Simulation AMESim

La plupart des exemples traités sur le site sont des cas d’étude « livrés clés en main », notamment par la société LMS. Cependant, nous avons parfois été amenés à ajouter des modifications afin de les adapter à nos besoins.

## Comportement longitudinal d’un véhicule

Pour ce cas d’étude, le modèle AMESim a dû être légèrement modifié afin d’y apporter les changements suivants :

* suivre la vitesse longitudinale du véhicule en km/h – au lieu de l’unité par défaut, m/s ;
* pouvoir suivre l’évolution de la consommation « instantanée » du véhicule en L/100 km – un peu à la manière de ce qui existe dans les ordinateurs de bord des voitures actuelles.

Afin de convertir la vitesse, initialement exprimée en m/s en km/h, il suffit d’ajouter un élément de la bibliothèque signal en sortie du capteur de vitesse. On peut par exemple appliquer un gain de 3,6 au signal de sortie.

Obtenir la consommation instantanée du véhicule demande quelques manipulations supplémentaires : en effet, ce paramètre est une variable interne du sous-modèle de moteur utilisé (DRVICE01C). Il n’est donc pas possible de récupérer directement cette donnée.



Pour y remédier, nous avons créé un nouveau sous-modèle, baptisé DRVICE01CMOD, que nous avons modifié à l’aide du logiciel AMESet – partie intégrante de la suite LMS Imagine.Lab. Les modifications apportées sont les suivantes :

* Le paramètre de consommation instantanée est passé en variable externe, sortant au niveau du port 2. Pour ce faire, il a fallu créer une variable externe de type sortie et identifiée par son nom de variable conso – c’est ce nom de variable qui permet de récupérer les données lors de la simulation.
* Afin d’éviter toute confusion, les autres variables externes du port 2 – données concernant les émissions polluantes essentiellement – sont passées en variables internes.



Il n’y a donc plus au port 2 qu’une seule sortie signal, qu’il est alors possible de récupérer et de traiter afin d’obtenir le type de donnée souhaité.

# Applet Java

## Exportation des données AMESim

 Le logiciel AMESim offre deux possibilités d’exportation de données : sous la forme d'un fichier texte basique ou bien d'un fichier XML. C'est cette dernière solution que nous avons retenue, car dans le fichier XML figurent un certain nombre d'informations qui nous ont été utiles par la suite, telles que le nombre de courbes, le nombre de points du tracé, etc.

 Pour créer le fichier de données AMESim, nous sommes passés par la fonction d'affichage graphique du logiciel (Plot), en suivant à chaque fois la démarche suivante :

* Dans un premier temps, nous affichons les courbes que nous voulons voir figurer sur un même tracé, ce qu’AMESim désigne par *CURVE* dans son fichier de données.
* Ensuite, nous transformons le graphique en graphique par batch (*Convert to batch plot*) afin d'afficher toutes les courbes (groupées par *CURVES*) obtenues à la suite du calcul par batch : en effet, la façon dont nous avons pensé le fonctionnement de l’applet requiert que toutes les données soient présentes dans le même fichier. Il faut néanmoins garder en mémoire que l'applet permettra de sélectionner quelles courbes on souhaite afficher.
* Enfin, nous exportons les données sous la forme d'un fichier XML à partir de la fenêtre graphique.

 Le fichier de données obtenu possède la structure suivante :



* **(1)** : le nombre de *CURVES*, c'est-à-dire de tracés – un tracé peut contenir plusieurs courbes.
* **(2)** : le nombre d’*ITEMS*, c’est-à-dire de colonnes dans le fichier de données. Chaque *CURVE* est composée au minimum de 2 *ITEMS* : une colonne pour le temps, et une colonne par courbe.
* **(3)**: le nombre de points.
* **(4)** : la légende de chaque courbe sous AMESim, associée à l’unité du paramètre affiché. Nous n'avons pas utilisé ces légendes dans l'applet car elles ne décrivent pas de manière précise les éléments constitutifs du système modélisé
* **(5)**: les données présentées sous forme de colonnes. La répartition des différentes courbes au sein de ces colonnes est très importante, notamment lors des calculs par batch, puisqu’elle permet à l’applet de retrouver la courbe demandée. Par défaut, les données sont réparties par AMESim comme suit :

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Paramètre / Colonne** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
| **Valeur (A)** | Temps | 1 | Temps | 1 | Temps | 2 | Temps | 2 | 1 | 1 | 2 | 2 |
| **Valeur (B)** | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 |

Le fichier de données est rempli un peu à la manière d’une table de vérité. Dans l’exemple ci-dessus, deux paramètres A et B peuvent prendre chacun une valeur 1 ou 2 ; il y a 4 *CURVES* et 12 *ITEMS*, soit 2 courbes par tracé ; les colonnes de la même couleur appartiennent au même tracé.

Avec cette organisation structurée des données, il est possible pour l’applet de retrouver à chaque fois les données correspondant à la courbe demandée quel que soit le nombre de paramètres, grâce à une méthode de recherche récursive.

## Bibliothèques Java additionnelles

 À plusieurs reprises, nous nous sommes retrouvés dans une situation où les fonctionnalités que nous souhaitions intégrer à l'applet dépassaient le champ de nos connaissances en Java. Nous avons alors dû faire appel aux possibilités offertes par des bibliothèques Java additionnelles. Ainsi, l'applet utilise deux bibliothèques Java libres : JDOM et JFreeChart.

### JDOM

 *Version utilisée : JDOM 1.1.1*

 La bibliothèque libre **JDOM (Java Document Object Model)** permet l'accès, la manipulation et l'édition de fichiers XML, de façon optimisée pour une utilisation dans un programme en Java (en utilisant notamment les possibilités de la programmation objet).

 Nous avons utilisé cette bibliothèque dans le projet afin de permettre à l'applet d'accéder et de lire les fichiers de données exportés depuis AMESim, sous la forme de fichiers XML. L'utilisation de JDOM a permis de récupérer certaines informations importantes contenues dans des balises XML, telles que le nombre de courbes ou le nombre de points du tracé correspondant au fichier. L'utilisation du XML et de JDOM permet un accès simple, rapide et fiable à ces informations.

### JFreeChart

 *Version utilisée : JFreeChart 1.0.13*

 La bibliothèque libre **JFreeChart** permet la création de graphiques de manière relativement aisée en Java. Elle permet, entre autres, de tracer des graphiques 2D, des histogrammes, des diagrammes circulaires, etc. Son principe de fonctionnement utilise les possibilités de la programmation objet. La bibliothèque utilise une classe spécifique pour collecter les données avant de tracer un graphique dans un dérivé de *JPanel*. Cela facilite l'intégration des graphiques ainsi réalisés dans un programme Java.

 Nous avons utilisé cette bibliothèque afin de tracer des graphiques 2D correspondant à ceux affichés par AMESim (temps en abscisse, paramètre en ordonnée). Les courbes sont légendées afin de pouvoir les identifier lors de l'affichage de résultats par batch. JFreeChart offre en outre un grand nombre de possibilités à l'utilisateur, qui peut zoomer, imprimer ou enregistrer le graphique affiché sous la forme d'une image.



Figure : Exemple d'utilisation de la bibliothèque JFreeChart dans l'applet Java : affichage simultané de plusieurs courbes dans un graphique 2D légendé.

## Arborescence

 Une applet est utilisée pour chaque cas d’étude (fichier exécutable .jar généré à l’aide de la fonction *Export* d’Eclipse). L’ensemble des fichiers nécessaires au fonctionnement de toutes les applets se trouvent dans un répertoire applet/, qui possède la structure décrite ci-dessous :

|  |
| --- |
| * applet/
	+ jar/
		- jcommon-1.0.16.jar
		- jdom.jar
		- jfreechart-1.0.13.jar
	+ cas1/
		- data/
			* affiche\_0
			* affiche\_1
			* …
		- cas1.jar
		- index.html
		- modele.png
	+ cas2/
	+ …
 |

Le répertoire jar/ contient les fichiers jar communs à toutes les applets, en fait les fichiers correspondant aux bibliothèques Java additionnelles utilisées.

 Chaque cas d’étude est stocké dans son propre répertoire.

 Les fichiers de données AMESim sont stockés dans le sous-répertoire data/. Le nom de ces fichiers n’est pas anodin : le chiffre à la fin du nom affiche\_x correspond à l’indice du paramètre dans la liste des paramètres affichés. Ainsi, si les paramètres sont ordonnancés de la façon suivante :

* parametreAffiche[0] : déplacement de la masse et de la roue ;
* parametreAffiche[1] : vitesse de la caisse ;
* etc.

les données du déplacement de la masse et de la roue seront alors stockées dans le fichier affiche\_0, la vitesse de la caisse dans affiche\_1, etc.

 Le fichier Java exécutable cas1.jar contient l’applet compilée ; ce fichier est créé depuis le logiciel Eclipse.

 Le fichier modele.png est l’image du modèle AMESim légendé qui est affiché dans l’applet. Ce fichier doit être impérativement nommé ainsi si l’on ne veut pas effectuer de modifications supplémentaires dans l’applet.

## Fichier index.html

 C’est le fichier index.html qui est ouvert dans un premier temps par le navigateur et qui va appeler les fichiers .jar nécessaires au lancement de l’applet. Ce fichier se présente de la façon suivante :

|  |
| --- |
| <html> <head> <title>1er cas d'étude : modèle de suspension simplifiée [Cas test de simulation AMESim]</title> <meta http-equiv="Content-Type" content="text/html; charset=utf-8" /> </head> <body> <applet code="AppletGraph.class" width=100% height=100% archive="/amesim/applet/jar/jcommon-1.0.16.jar, /amesim/applet/jar/jfreechart-1.0.13.jar, /amesim/applet/jar/jdom.jar, cas1.jar"> <param name="java\_arguments" value="-Xmx256m">  </applet> </body></html> |

C’est la balise <applet> qui appelle l’ensemble des fichiers Java. Les seuls paramètres à modifier sont les adresses des fichiers Java communs : l’adresse doit être spécifiée sous la forme absolue depuis la racine du serveur ; ainsi que le nom du fichier cas1.jar suivant les cas.

 La balise <param name="java\_arguments" value="-Xmx256m"> porte la quantité de mémoire utilisable par la machine virtuelle Java à 256 Mo, ceci afin d’éviter tout problème de déficit de mémoire au cours de l’exécution de l’applet. Nous avons ajouté cette balise après avoir été confrontés à de tels problèmes, notamment sous Windows.

## Modifications à apporter au code source Java

 Les modifications à apporter au code source Java de l’applet se limitent à la classe principale, AppletGraph.java. Elles concernent surtout la définition des différents paramètres et leurs valeurs numériques. Il faut donc effectuer les modifications au niveau des déclarations des variables locales, à la fin du fichier.

###  parametreVariable

|  |
| --- |
| private String[] parametreVariable = {"masse suspendue [kg]", "raideur ressort [N/m]", "amortissement [N/(m/s)]"}; |

Ce tableau de chaînes de caractères liste les paramètres variables, qui correspondent aux paramètres que l’on a fait varier par batch sous AMESim. Il est important de conserver le même ordre des paramètres que sous AMESim, afin d’être cohérent avec les fichiers de données.

### parametreAffiche

|  |
| --- |
| private String[] parametreAffiche = {"déplacement [m]", "vitesse [m/s]", "accélération [m/s2]", "vitesse amortisseur [m/s]", "force amortisseur [N]"}; |

Ce tableau de chaînes de caractères liste les paramètres affichés. Il est important que la position d’un paramètre soit en cohérence avec le nom du fichier de données correspondant, afin de rendre possible la lecture de ce dernier.

### valeur

|  |
| --- |
| private String[] valeurM1 = {"300", "500", "700"}; |

Ce tableau de chaînes de caractères liste les valeurs numériques proposées pour un paramètre. Il est important de conserver le même ordre des valeurs que sous AMESim, afin d’être cohérent avec les fichiers de données.

### legende

|  |
| --- |
| private String[] legendeD1 = {"roue", "caisse"}; |

Pour les tracés avec plusieurs courbes, il est utile d’ajouter les noms du composant correspondant dans la légende de chacune des courbes, afin de faciliter leur identification. Pour ce faire, on peut créer autant de tableaux de chaînes de caractères que nécessaire. Chaque tableau doit comporter autant d’éléments que le nombre de courbes du tracé qu’il légende.

### listeValeur

|  |
| --- |
| private String[][]listeValeur = {valeurM1, valeurR1, valeurAm1}; |

Ce tableau 2D de chaînes de caractères rassemble les listes de valeurs numériques définies précédemment. Il doit comporter autant d’éléments qu’il y a de paramètres variables.

###  listeLegende

|  |
| --- |
| private String[][] listeLegende = {legendeD1, legendeV1, legendeV1, legendeA1, legendeA1}; |

Ce tableau 2D de chaînes de caractères rassemble les listes de légendes de courbes définies précédemment. Il doit comporter autant d’éléments qu’il y a de paramètres affichés.

### nombreCourbes

|  |
| --- |
| private int[]nombreCourbes = {2,1,1,1,1}; |

Ce tableau d’entiers liste le nombre de courbes dans chacun des tracés. Il doit comporter autant d’éléments qu’il y a de paramètres affichés. Il est important de conserver le même ordre que celui défini dans le tableau parametreAffiche.

### adresseDonnees

|  |
| --- |
| private String adresseDonnees = "http://fredericngo.com/amesim/applet/cas1/"; |

Cette chaîne de caractères stocke l’adresse du dossier où se trouve l’applet. Il est possible de spécifier une adresse Internet ou le chemin d’un dossier sur l’ordinateur si l’on souhaite tester l’applet en local. Il est important de ne pas oublier le / à la fin de l’adresse.

# Structure du site Web

Comme recommandé dans la notice d’utilisation OptimOffice, nous avons divisé notre espace de travail en quatre parties :

* pub/ : contient les items de publication : les pages Web et les dossiers de pages Web ;
* ref/ : contient les références, ici les définitions ;
* res/ : contient les ressources : images, équations, tableaux et animations ;
* src/ : contient les items de contenu (il n’y en a aucun sur le site).



La page d’accueil du site index.website regroupe plusieurs éléments :

* une marge où se trouvent les noms des partenaires du projet (ainsi que des liens vers leurs sites internet) ;
* un bloc contenant le plan du site ;
* un index contenant des pages ou dossiers web : la page « Réalisation » et les dossiers de pages web (.webfolder) des sous-espaces de niveau 1 y sont référencés.

Les **sous-espaces de niveau 1** correspondent au cours et exemples. Il y en a quatre :

* mecagen/ : le cours « Généralités sur la mécatronique » ;
* suspension/ : le cours « Analyse de différents types de suspensions » ;
* modelisation/ : le cours « Modélisation des systèmes dynamiques et commande avancée » ;
* exemples/ : regroupe tous les exemples traités.

Chaque sous-espace contient obligatoirement un dossier de pages web regroupant soit les pages web dudit sous-espace, soit les dossiers de pages web des sous-espaces de niveau inférieur. Cela permet de hiérarchiser le site.

**

Les items de référence (res/) ne possèdent pas d’organisation particulière compte tenu du faible nombre de définitions.

Pour ce qui est des items de ressources (res/), ils sont organisés de la même manière que dans l’espace publication. Il y a une succession de plusieurs sous espaces.À l’intérieur des sous-espaces, les images sont nommées de manière explicite afin de pouvoir les retrouver plus facilement si besoin est ; les équations sont rangées dans leur ordre d’apparition.

Il faut savoir que la mise en page du site web est intimement liée à l’organisation des sous-espaces :

**