```
/*************************************************************************
 *  Logiciel portageMM2CPP de portage de code ModelMaker en langage C++      *
 *  Copyright INRA, février 2006                                             *
 *************************************************************************/

/*************************************************************************
 *
 * Fichier     : help
 *
 * Auteur(s)   : Nathalie Rousse, Nathalie.Rousse@toulouse.inra.fr
 *               de l'INRA - Institut National de la Recherche Agronomique -
 *               ( UMR ARCHE / Plate-forme INRA-ACTA-ICTA ).
 *
 * Description : Fichier de documentation du logiciel portageMM2CPP
 *
 *************************************************************************
 * Historique :
 *
 * 03/02/06, Nathalie Rousse : création du fichier.
 *
 *************************************************************************/
/*

****** PRESENTATION GENERALE ****************************************************

Le logiciel portageMM2CPP propose une base d'écriture en langage C++ d'un modèle qui a été programmé sous ModelMaker.

Ce logiciel s'appuie sur un cas précis, il est une réécriture en langage C++ du modèle écrit dans le fichier ModelMaker
 "modele2.mod" (situé dans le répertoire "leSourceModelMaker").

Ce logiciel peut servir d'exemple, être repris/adapté par quelqu'un qui souhaiterait construire son propre logiciel en
C++ à partir de son propre modèle ModelMaker.

Ce logiciel comporte une partie générique qui "se veut" indépendante du modèle traité et une partie complètement spécif
ique du modèle traité. Il est clairement fait la distinction entre ces deux parties, elles sont écrites dans des fichie
rs distincts. Ainsi d'un côté la partie générique peut être réutilisée tandis que de l'autre, la partie spécifique peut
 servir d'exemple et être adaptée à d'autres cas de modèles ModelMaker (fichiers ".mod").

Avertissement / limitation :
Ce logiciel dans son état actuel ne peut pas être repris pour recoder dans son intégralité n'importe quel modèle écrit
avec ModelMaker. Ce logiciel répond aux besoins du modèle ici traité ("modele2.mod" pris en exemple), il ne prend pas e
n compte toutes les possibilités offertes par ModelMaker. Il pourrait dans le futur être enrichi dans ce sens.

*********************************************************************


****** FICHE D'IDENTITE *********************************************************
```

****** Titulaire des droits patrimoniaux sur le logiciel portageMM2CPP :

INRA - Institut National de la Recherche Agronomique - http://www.inra.fr

****** Auteur(s) :

Nathalie Rousse, Nathalie.Rousse@toulouse.inra.fr
de l'INRA / UMR ARCHE / Plate-forme INRA-ACTA-ICTA (http://www.modelia.org)

****** Licence :



****** Environnement/outils de développement :

Cygwin, ModelMaker 4, Poseidon For UML : voir "Environnement de développement".

****** Site de publication/diffusion du logiciel :

Le logiciel portageMM2CPP est diffusé/téléchargeable sur modelia
http://www.modelia.org, le site de la plate-forme et du club INRA-ACTA-ICTA.

**********

****** POUR COMMENCER **********

Pour prendre connaissance du logiciel, consulter la documentation (regroupée dans ce fichier "help"). Toute personne ay
ant besoin de renseignements complémentaires est invitée à contacter Nathalie Rousse (adresse mél ci-dessus).

Comment se servir du logiciel ?  Voir "Guide d'utilisation" et "Mode d'emploi" (dans ce fichier "help").

**********

****** LES REPRESENTATIONS UML **********

Les représentations UML : voir les fichiers UML1portageMM2CPP.pdf,UML2portageMM2CPP.pdf,... (issus du fichier UMLportag
eMM2CPP.zuml).

**********

****** STRUCTURE, ORGANISATION **********

Liste des répertoires du logiciel :

- laDocumentation
- leSourceModelMaker
- leSourceCPP
- executionCPP
- lesSortiesCPP
- lesTestsCPP
- LICENCE

L'organisation et le contenu des répertoires et fichiers du logiciel sont présentés dans les représentations UML : "Dia
gramme de déploiement portageMM2CPP" et " Diagramme de déploiement Répertoire "lesTestsCPP" ".

Voir aussi les fichiers "readme" des répertoires.

**********

****** ENVIRONNEMENT DE DEVELOPPEMENT **********

L'environnement dans lequel le logiciel portageMM2CPP a été développé en C++ sous Windows est Cygwin (http://cygwin.com ) et son compilateur C++ "g++".

Le modèle sous sa forme ModelMaker ("modele2.mod" et .txt associés) a été produit avec le logiciel "ModelMaker" (versio n 4) de ModelKinetix (http://www.modelkinetix.com).

Les représentations UML ("UMLportageMM2CPP.zuml" et .pdf associés) ont été produites avec le logiciel "Poseidon For UML " (Community Edition 4.x) de Gentleware (http://www.gentleware.com).

**********

****** GUIDE D'UTILISATION **********

(Voir en illustration le "Diagramme de déploiement portageMM2CPP" dans les représentations UML)

1) Première utilisation,
   Exécution du logiciel tel qu'il a été généré (c'est à dire exécutable actuel) :

Dans le répertoire "executionCPP", lancer la commande en ligne d'exécution du logiciel : "./portageMM2CPP.exe".

Les résultats des simulations exécutées sont alors sortis/affichés à l'écran et dans les fichiers du répertoire "lesSor tiesCPP". Les résultats de simulation affichés à l'écran sont plus ou moins détaillés : en fonction de la valeur de la clé de compilation CLE_AVEC_TRACE_ECRAN_DONNEES - voir "Seconde utilisation" - les valeurs des données au cours du temp s sont ou non affichées à l'écran.

Le répertoire "lesSortiesCPP" contient les sorties de la dernière exécution effectuée. Notamment, lors de l'exécution d es tests du logiciel (voir "Tests du logiciel" et le répertoire "lesTestsCPP"), les fichiers du répertoire "lesSortiesC PP" sont écrasés par les résultats des tests. C'est pourquoi, tout de suite après avoir exécuté une simulation dont on souhaite conserver les résultats, il est préférable d'enregistrer/copier le répertoire "lesSortiesCPP" dans un répertoi re de sauvegarde.

Remarque : s'il est souhaité conserver les traces écran dans un fichier "nom_fichier", la commande à lancer est : "./po rtageMM2CPP.exe > nom_fichier".

2) Seconde utilisation,
   Compilation et génération de l'exécutable à partir du code source :

Préalable requis : Le logiciel est écrit en langage C++. Pour compiler ce logiciel et générer l'exécutable à partir du code source, il faut avoir installé un compilateur C++. Par exemple si on travaille sous Windows, le compilateur g++ fa it partie de Cygwin, environnement pour Windows. Cygwin (cf "http://cygwin.com") s'exécute au-dessus de Windows, ressem ble par bien des aspects à Linux, Unix.

Dans le répertoire "executionCPP", lancer la commande en ligne de compilation du code source et de génération de l'exéc utable à partir du code source : "./compiler".

Note : Le choix du mode d'affichage des traces à l'écran se fait en (dés)activant la clé de compilation CLE_AVEC_TRACE_ ECRAN_DONNEES (dans le fichier "compiler", avant de lancer la commande de compilation "./compiler").

3) Troisième utilisation,
   Construire son propre logiciel en C++ à partir de son propre modèle ModelMaker (fichier ".mod") :

Voir "Mode d'emploi".

**********

****** SPECIFICATION **********

Le logiciel est composé de 3 "parties" associées à 3 "fichiers" :

1. généricité (genericitePortageMM2CPP.cpp, genericitePortageMM2CPP.h) :
Contient la partie générique qui "se veut" indépendante du modèle traité.
Cette partie prend en compte les besoins liés au modèle ici traité ("modele2.mod" pris en exemple). Tant qu'elle sert à construire des logiciels C++ de modèles ModelMaker "du même type" que "modele2.mod", cette partie n'est pas sensée bouger. Cependant, elle est susceptible d'évoluer si elle était reprise pour construire le logiciel C++ d'un modèle ModelMaker plus élaboré que "modele2.mod" (qui est ultra simple). Elle pourrait alors être modifiée/enrichie par rapport aux (aspects génériques des) spécifications du nouveau modèle (voir aussi "Mode d'emploi", voir aussi les commentaires marqués du label "modeDemploiEvolutions" dans le code).

2. spécificité (specificitePortageMM2CPP.cpp, specificitePortageMM2CPP.h) :
Contient la partie complètement spécifique du modèle traité : déclaration des différentes entités composant le modèle, configuration, paramétrage du modèle, écriture des relations entre les différentes entités composant le modèle ...
Quelqu'un qui reprendrait ce logiciel pour construire le logiciel C++ d'un autre modèle (autre que "modele2.mod") doit réécrire cette partie (la déclaration des données, le contenu des fonctions). Par ailleurs cette partie est aussi susceptible d'évoluer par rapport aux (aspects spécifiques des) spécifications du nouveau modèle (voir aussi "Mode d'emploi", voir aussi les commentaires marqués du label "modeDemploiEvolutions" dans le code).

3. main (mainPortageMM2CPP.cpp) :
Contient une utilisation du modèle : configuration et appel des simulations.
Selon le contexte d'utilisation du modèle, ce moyen d'appel du modèle (appel de manière autonome et très simpliste) est susceptible d'être remplacé par une IHM plus élaborée, ou encore par l'invocation de la fonction de simulation dans un autre programme/logiciel (d'un autre modèle, d'une application logicielle ...).

Le logiciel est décrit dans les représentations UML :
- la vue d'ensemble des classes du code C++ (contenu dans "leSourceCPP") dans le "Diagramme des classes du code C++".
- les classes dans les "Diagramme des classes génériques" et "Diagramme des classes spécifiques" (pour alléger la description, quelques classes secondaires ne sont pas mentionnées).
- quelques algorithmes dans "Diagramme d'activité de méthodes".

Niveau de granularité/modularité :
Le système (dans l'exemple traité : "systemParticulier") est l'objet réutilisable. C'est lui qui peut être appelé par un autre programme C++, il forme "un bloc". En effet, le système est composé de plusieurs objets (voir classe "systemParticulier"), mais ces objets sont interdépendants (voir la fonction "actualiserIncrement") et cela n'aurait aucun sens d

e les appeler unitairement (c'est-à-dire "hors système").

**********

****** TESTS DU LOGICIEL **********

Il s'agit de vérifier que le logiciel écrit en C++ est conforme au modèle sous sa forme ModelMaker ("modele2.mod"), c'est à dire qu'il se comporte à l'identique, sort les mêmes résultats. Il n'est pas du tout question ici de faire de la validation du modèle (ajustement de paramètres, analyse de sensibilité ...).

Les tests sont mis en place et en oeuvre selon le principe suivant : après avoir programmé en C++ l'appel des simulations "équivalentes" à celles qui sont effectuées dans ModelMaker, comparer les sorties ainsi obtenues avec les résultats des simulations effectuées intra-ModelMaker ; voir en illustration la "Séquence de construction et déroulement d'un scénario de test" dans les représentations UML.

Le répertoire des tests est "lesTestsCPP". Il contient :
- le fichier "tester" qui est le script de compilation et exécution des tests.
- les répertoires des scénarios de test. Chacun de ces répertoires regroupe les informations du scénario de test auquel il est associé : ses entrées (ce qui permet de dérouler le test) et ses sorties (les résultats du test).

Parmi les répertoires des scénarios de test, le répertoire "scnParDefaut" est particulier dans le sens où c'est le répertoire du scénario de test par défaut.

Le répertoire "scnParDefaut" contient :
- le fichier "mainTestPortageMM2CPP.cpp" qui contient le programme principal des tests du scénario. Ce fichier est écrit par le testeur.
- le répertoire "svg_leSourceCPP" qui est une recopie/sauvegarde (qui est effectuée automatiquement lors de l'exécution du test) du répertoire "leSourceCPP" tel qu'il a été utilisé à l'exécution du test.
- le fichier "testPortageMM2CPP.exe" qui est l'exécutable du test (fichier généré et appelé automatiquement lors de l'exécution du test).
- le répertoire "lesSorties" qui contient les résultats des tests (ie les fichiers .res résultats de la simulation en C++), qui sont rangés là automatiquement lors de l'exécution du test (ils sont récupérés dans "lesSortiesCPP").
- le répertoire "lesEntrees" qui contient les données qui serviront d'éléments de comparaison lorsque le testeur fera ses vérifications. C'est le testeur qui remplit "lesEntrees". A priori il y dépose des données issues de ModelMaker (résultats des simulations effectuées intra-ModelMaker).
- le fichier "rapportTest" où le testeur notifiera les conclusions du test (la trame du fichier "rapportTest" est créée automatiquement lors de l'exécution du test si le fichier n'existe pas encore).

Les autres répertoires de scénarios de test suivent le même schéma que "scnParDefaut".

Procédure de développement puis de déroulement du scénario de test par défaut :
- Dans "scnParDefaut", écrire le programme du scénario de test "mainTestPortageMM2CPP.cpp".
- Dans "scnParDefaut/lesEntrees", déposer les données de comparaison.
- Compiler/exécuter le scénario de test par défaut, en lançant dans le répertoire "lesTestsCPP" la commande en ligne : "./tester". Les résultats du test sont alors rangés dans le répertoire "scnParDefaut/lesSorties".
- Dans "scnParDefaut", faire les vérifications voulues par le test : a priori comparaison des fichiers résultats de la simulation en C++ (contenus dans "lesSorties") avec les données de comparaison issues de ModelMaker (contenues dans "le

sEntrees").
- Dans "scnParDefaut", notifier les conclusions du test dans le fichier "rapportTest".

Développement de plusieurs scénarios de test scni=scn1,scn2,...,scnN :
- Dans le répertoire "scni", écrire le programme du scénario de test scni : "mainTestPortageMM2CPP.cpp" (on peut aussi particulariser son nom, par exemple : "mainTestPortageMM2CPP_scni.cpp"),
- Dans le répertoire "scni/lesEntrees", déposer les données de comparaison.
- Compiler/exécuter le scénario de test scni, en lançant dans le répertoire "lesTestsCPP" la commande en ligne : "./tester scni" (ou "./tester scni mainTestPortageMM2CPP_scni.cpp" si on avait donné au fichier un nom particulier). Les résultats du scénario de test scni sont alors rangés dans le répertoire "scni/lesSorties".
- Dans le répertoire "scni", faire les vérifications voulues par le test : a priori comparaison des fichiers résultats de la simulation en C++ (contenus dans "lesSorties") avec les données de comparaison issues de ModelMaker (contenues dans "lesEntrees").
- Dans "scni", notifier les conclusions du test dans le fichier "rapportTest".

**********

****** MODE D'EMPLOI **********

**********
*
*  Comment, à partir de ce logiciel,
*  construire son propre logiciel en C++ (.cpp, .h)
*  relatif à son propre modèle ModelMaker (fichier ".mod").
*
**********

Les modifications qu'il faut apporter au code C++ sont notées directement dans le code (*.cpp et *.h) : ce sont les commentaires marqués du label "modeDemploi" (b). De plus on trouve dans les représentations UML une aide à la mise en oeuvre des modifications : ce sont les commentaires sur fond jaune dans le "Diagramme des classes spécifiques" (a).

Le déroulement de la procédure à suivre pour construire son propre logiciel C++ est décrit ci-dessous :

1) Récupérer en l'état le logiciel exemple "portageMM2CPP" :

Copier/coller dans son propre espace de travail tout l'espace de travail (répertoires sources, tests, exécution ...) :

- leSourceModelMaker
- leSourceCPP
- executionCPP
- lesSortiesCPP
- lesTestsCPP
- laDocumentation
- LICENCE

2) Modifier le contenu du répertoire leSourceModelMaker

Remplacer modele2.mod de l'exemple par son propre fichier ModelMaker, de même pour modele2.txt (voir explications dans "readme" de "leSourceModelMaker").

Remarque : Le fichier "modele2.txt" pourra par ailleurs servir pour la vérification de cohérence entre versions logicielles. En effet, une fois que le modèle aura été réécrit en C++, il existera deux formes logicielles du modèle (formes ModelMaker et C++) susceptibles d'évoluer en parallèle. Pour repérer les différences/changements entre une version "vCPP" du logiciel C++ (dont le .txt associé est modele2.txt.vCPP) et une version "vMM" du modèle ModelMaker modele2.mod (dont le .txt est modele2.txt.vMM), on pourra comparer leur .txt : commande "diff modele2.txt.vCPP modele2.txt.vMM".

3) Modifier le contenu du répertoire leSourceCPP

3.1) la partie générique (genericitePortageMM2CPP.cpp, genericitePortageMM2CPP.h) :

A priori il n'est pas besoin de retoucher à cette partie.

Balayer toutefois les instructions marquées du label "modeDemploi" jalonnant le code (voir "(b)").

En effet il peut être nécessaire de faire évoluer ce code pour différentes raisons :
- besoin d'enrichir/modifier cette partie si le nouveau modèle, plus élaboré que celui de l'exemple, utilise des possibilités de ModelMaker qui ne sont pas prises en compte actuellement (voir les commentaires marqués du label "modeDemploiEvolutions" dans le code).
- besoin/choix d'implémenter une(de) nouvelle(s) fonction(s) de simulation "simuler" par rapport à l'utilisation prévue du logiciel C++ (voir description dans les représentations UML : "Diagramme des classes génériques" et "Diagramme d'activité de méthodes", voir les commentaires marqués du label "modeDemploiEvolutions" dans le code).
- dimensionnements : voir "3.4)"

3.2) la partie spécifique (specificitePortageMM2CPP.cpp, specificitePortageMM2CPP.h) :

Reécrire le code de ces fichiers : la déclaration des données, le contenu des fonctions. Pour cela suivre les instructions marquées du label "modeDemploi" jalonnant le code (voir "(b)") et s'aider des explications et illustrations de "(a)".

Par ailleurs, cette partie est susceptible de bouger au titre d'évolutions. Par exemple, quelqu'un peut être conduit à reprendre les fonctions qui restituent les résultats sortis des simulations ("afficherEntetesEntites" et "afficherEntites") s'il a besoin de sortir des résultats à d'autres formats... (voir les commentaires marqués du label "modeDemploiEvolutions" dans le code).

3.3) le main (mainPortageMM2CPP.cpp) :

Reprendre le code pour appeler la simulation comme voulu (paramètres d'appel de "simuler" ...). Pour cela suivre les instructions marquées du label "modeDemploi" jalonnant le code (voir "(b)") et s'aider des explications et illustrations de "(a)".

3.4) Configuration/dimensionnement

Dans tous les fichiers, vérifier les configurations/dimensionnements actuellement codés/définis. Ces éléments sont repérables grâce au label "modeDemploi_ConfDim" dont ils sont marqués. Si besoin, les modifier : changer des constantes/valeurs par défaut, augmenter des tailles de tableaux ...

3.5) label "modeDemploi"

Cette étape "3)" n'est pas achevée tant qu'on ne s'est pas assuré, dans TOUT LE CODE de TOUS LES FICHIERS, d'avoir consulté et suivi les consignes/indications marquées du label "modeDemploi" (voir "(b)").

4) Essai et validation.

Mettre en place et effectuer les tests permettant de vérifier son propre logiciel : dans son propre fichier ModelMaker (".mod") et dans le répertoire "lesTestsCPP" ; voir "Tests du logiciel".

5) Utilisation du logiciel

Une fois le logiciel validé, l'intégrer dans son propre contexte. Il peut s'agir de remplacer le main par une IHM plus élaborée, ou encore d'invoquer la fonction de simulation "simuler" dans un autre programme/logiciel C++ (d'un autre modèle, d'une application logicielle ...) etc.

**********

*/