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

/*****************************************************************************
 *
 * Fichier     : help_specification
 *
 * Auteur(s)   : Nathalie Rousse, Nathalie.Rousse@toulouse.inra.fr
 *               de l'INRA - Institut National de la Recherche Agronomique -
 *               ( Département MIA, UMR AGIR, RMT modelia ).
 *
 * Description : Fichier faisant partie de la documentation du logiciel
 * portageMM2CPP. L'ensemble des fichiers de documentation est récapitulé
 * dans le fichier help.
 *
 *****************************************************************************
 * Historique :
 *
 * 23/11/07, Nathalie Rousse : création du fichier.
 *
 * Le fichier est créé par déplacement de texte qui se trouvait jusque là
 * dans le fichier help qui est ensuite complété ici.
 *
 *****************************************************************************/
/*

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

******
****** Généralités :
******

-------------------------------------------------------------------------------

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

1. généricité (genericitePortageMM2CPP.cpp, genericitePortageMM2CPP.h et genericiteConfigPortageMM2CPP.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 bou
ger. Cependant, elle est susceptible d'évoluer si elle était reprise pour construire le logiciel C++ d'un modèle ModelM
aker 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 marq
ués du label "modeDemploiEvolutions" dans le code).
```

2. spécificité (specificitePortageMM2CPP.cpp, specificitePortageMM2CPP.h et specificiteConfigPortageMM2CPP.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 ...).

Représentations UML :

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".

A propos des représentations UML, voir "Les représentations UML" dans le fichier help.

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

******
****** Affichages :
******

Certains affichages sont exécutés sous condition d'activation de la clé de compilation CLE_AVEC_TRACE_ECRAN_DONNEES. D'autres sont inconditionnels comme : informations à l'initialisation, en début de simulation (présentation des conditions de la simulation lancée), information de sortie en erreur.

******
****** Initialisation du système :
******

Présentation de la séquence d'initialisation du système :

Pseudo-code du programme principal (*) :

```
        initialisation_phase1 : systemParticulier S;

        si S.pasDePbInitReleve() est true alors :
                (b) Lancement de la simulation : S.simuler( ... );
        sinon :
                (a) Affichage du problème puis arrêt/fin :
                  " SIMULATION NON LANCEE POUR CAUSE DE PROBLEME RELEVE
                    A L'INITIALISATION DU SYSTEME :
                    code erreur CrInitValue = valeur de S.getCrInit() "
        fin si.
```

L'affichage (a) est systématiquement exécuté en cas d'erreur/problème rencontré. Le code erreur rend compte d'une erreur survenue en cours d'initialisation (une seule, quel que soit le nombre d'erreurs survenues). En activant la clé de compilation CLE_AVEC_TRACE_ECRAN_DONNEES, il est possible d'obtenir l'affichage d'informations supplémentaires sur les problèmes rencontrés/erreurs survenues.

(b) initialisation_phase2 : la fonction "simuler" commence par effectuer un certain nombre de vérifications qui peuvent conduire à stopper l'exécution (sortie en erreur). Exemple de codes d'erreurs possibles à ce niveau : CR_NOK_BornesIncoherentes, CR_NOK_PasSimulNegatif, CR_NOK_InutilisabiliteFichiersDonneesAinterpoler.
Remarque : ces vérifications peuvent aussi donner lieu à des réactions moins radicales (non bloquantes) comme faire des retouches (modification/augmentation de pasDeLaSimulation par rapport à la demande, pour des questions de précision des calculs)
ou donner des avertissements (avertir/signaler si, étant données les conditions de simulation, on risque d'être amené à extrapoler par rapport aux données d'un fichier de données à interpoler).

(*) Pseudo-code du programme principal plus détaillé par rapport aux traces/messages affichés :

- initialisation_phase1 : systemParticulier S;

- si S.pasDePbInitReleve() est true alors :

  (b) Lancement de la simulation S.simuler(...) : ----------
  |
  | Affichage message "***** COMMANDE DE LA SIMULATION ... les conditions de simulation demandées..."
  |

```
          |      Appel de la fonction de simulation de base cr = simuler() : ---
          |      |
          |      |      initialisation_phase2
          |      |
          |      |      si initialisation_phase2 NOT OK alors retourne cr compte rendu de l'erreur.
          |      |
          |      |      sinon affichage message "***** DEBUT DE LA SIMULATION EFFECTUEE ... les conditions de la simula
tion effectuée ..." ... Déroulement de la simulation ...  Affichage message " ***** FIN (PREMATUREE) DE LA SIMULATION."
 retourne compte rendu cr CR_OK.
          |      |      fin si
          |       ------------------------------------------------------------
          |
          |      si cr est CR_OK alors affichage message "Fin de fonction simuler sans erreur." puis fin.
          |      sinon affichage message "Fin de fonction simuler en erreur, code erreur : valeur de cr" puis fin.
          |      fin si.
          |
           ------------------------------------------------------------------
```

- sinon :

(a) Affichage du problème " SIMULATION NON LANCEE POUR CAUSE DE PROBLEME RELEVE A L'INITIALISATION DU SYSTEME : code erreur CrInitValue = valeur de S.getCrInit() " puis arrêt/fin.

- fin si.

******
****** Cas d'erreurs, traces et codes d'erreurs :
******

Cas/codes d'erreurs :

Certaines fonctions retournent un code d'erreur. Les codes d'erreurs sont définis dans le fichier genericitePortageMM2CPP.h dont un extrait est donné ici :

```
#define CR_OK                           1
#define CR_NOK                          0
#define CR_PAS_DE_PB_RELEVE             3
#define CR_NOK_virtualNonDefinie        10
#define CR_NOK_BornesIncoherentes       11
#define CR_NOK_PasSimulNegatif          12
#define CR_NOK_DepassementDim           13
#define CR_NOK_FichierInvalide          14 // (fichier non trouvé, vide...)
#define CR_NOK_FIN_FICHIER              15
#define CR_NOK_InutilisabiliteFichiersDonneesAinterpoler 16
#define CR_NOK_tentativeDivisionParZero 17
```

```
#define CR_NOK_BLOQUANT                18
#define CR_NOK_AVERTISSEMENT           19
```

Informations/traces détaillées sur les erreurs/problèmes :

En activant la clé de compilation CLE_AVEC_TRACE_ECRAN_DONNEES, il est possible d'obtenir l'affichage d'informations supplémentaires sur les problèmes rencontrés/erreurs survenues.

Erreurs selon les phases du logiciel :

- Erreurs en phase d'initialisation : voir "présentation de la séquence d'initialisation du système".

- Erreurs en cours de simulation : en cas d'erreur bloquante, la simulation est stoppée et c'est signalé par affichage écran ("FIN DE LA SIMULATION" ou "FIN PREMATUREE DE LA SIMULATION", "Fin de fonction simuler en erreur", affichage du code erreur). Le code erreur rend compte d'une erreur survenue (une seule, quel que soit le nombre d'erreurs survenues).

******
****** Données calculées par interpolation linéaire :
******

Les entités calculées par interpolation :

(voir aussi "fichiers des entités interpolées" juste après).

La classe fichierDentree : est associée à un fichier contenant une succession de données à lire. A terme, cette classe est destinée à traiter des fichiers du type de ce qui sont donnés en entrée de ModelMaker.

La classe donneesAinterpoler : est associée à une liste de données qui seront interpolées (ie qui servent d'information d'entrée à une interpolation). En l'état actuel, donneesAinterpoler est issu d'un fichierDentree, qui contient : la liste des données à interpoler (valueDonneeControlee) dans une colonne, et la liste des valeurs d'indexation (valueDonneeDeCtrl) correspondante dans une autre colonne.

La classe entityInterpolee : est dédiée aux entités du diagramme ModelMaker qui sont calculées par interpolation.

Remarque -- Restriction au niveau de la classe entityInterpolee par rapport à la classe donneesAinterpoler : au niveau de la classe donneesAinterpoler, on parle de donneeDeCtrl et de donneeControlee(donneeDeCtrl), chacune correspondant à une colonne du fichier des données à interpoler. Dans la classe entityInterpolee, il n'est plus question de donneeDeCtrl au sens large. donneeDeCtrl est fixé forcément à l'information instant/time.

Fichiers des entités interpolées :

(voir aussi "les entités calculées par interpolation" juste avant).

Format d'un fichier contenant une liste de données à interpoler :
à chaque ligne :
  - valueDonneeDeCtrl dans la colonneDonneeDeCtrl
  - valueDonneeControlee dans la colonneDonneeControlee (ie valueDonneeControlee(valueDonneeDeCtrl))

Caractéristiques des fichiers des entités interpolées :
Des caractéristiques des fichiers sont données sous forme de constantes dans le fichier specificiteConfigPortageMM2CPP.h : nomFichier_DONNEE_Ixxx, nbColonnes_DONNEE_Ixxx, colValeur_DONNEE_Ixxx, et colInstant_DONNEE_Ixxx. nomFichier_DONNEE_Ixxx est le nom du fichier (avec chemin d'accès complet) contenant la liste des données "Ixxx" à interpoler. Le fichier contient nbColonnes_DONNEE_Ixxx colonnes, parmi lesquelles la liste des données à interpoler dans la colonne colValeur_DONNEE_Ixxx, et la liste des valeurs des instants correspondant dans la colonne colInstant_DONNEE_Ixxx.

******
****** Divers :
******

Classe gestionAvancementSimulation :

La classe gestionAvancementSimulation gère des informations relatives à l'avancement dans la simulation : des indicateurs de situation/stade dans la simulation, des informations sur le positionnement de l'instant courant dans la simulation (booleens onEstSurUnPasDeLaSimulation et bool finPrematuree...).

****

*/