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

/*************************************************************************
 * Licence :

Le logiciel portageMM2CPP est une réécriture en langage C++ du modèle
écrit dans le fichier ModelMaker "modele2.mod" (situé dans le répertoire
"leSourceModelMaker"). Voir informations dans ../laDocumentation/help.

Ce logiciel est régi par la licence CeCILL soumise au droit français et
respectant les principes de diffusion des logiciels libres. Vous pouvez
utiliser, modifier et/ou redistribuer ce programme sous les conditions
de la licence CeCILL telle que diffusée par le CEA, le CNRS et l'INRIA
sur le site "http://www.cecill.info".

En contrepartie de l'accessibilité au code source et des droits de copie,
de modification et de redistribution accordés par cette licence, il n'est
offert aux utilisateurs qu'une garantie limitée.  Pour les mêmes raisons,
seule une responsabilité restreinte pèse sur l'auteur du programme,  le
titulaire des droits patrimoniaux et les concédants successifs.

A cet égard  l'attention de l'utilisateur est attirée sur les risques
associés au chargement,  à l'utilisation,  à la modification et/ou au
développement et à la reproduction du logiciel par l'utilisateur étant
donné sa spécificité de logiciel libre, qui peut le rendre complexe à
manipuler et qui le réserve donc à des développeurs et des professionnels
avertis possédant  des  connaissances  informatiques approfondies.  Les
utilisateurs sont donc invités à charger  et  tester  l'adéquation  du
logiciel à leurs besoins dans des conditions permettant d'assurer la
sécurité de leurs systèmes et ou de leurs données et, plus généralement,
à l'utiliser et l'exploiter dans les mêmes conditions de sécurité.

Le fait que vous puissiez accéder à cet en-tête signifie que vous avez
pris connaissance de la licence CeCILL, et que vous en avez accepté les
termes.

Les fichiers Licence_CeCILL_V2-fr.txt et Licence_CeCILL_V2-en.txt
du répertoire ../LICENCE fournissent le texte de la licence,
dans sa version 2, en langue française et en langue anglaise.
(ces 2 fichiers sont aussi directement dans le répertoire courant)

 *************************************************************************/

/*************************************************************************
```

```
 *
 * Fichier     : genericitePortageMM2CPP.h
 *
 * Auteur(s)   : Nathalie Rousse, Nathalie.Rousse@toulouse.inra.fr
 *               de l'INRA - Institut National de la Recherche Agronomique -
 *               (département MIA, UMR AGIR, http://www.modelia.org).
 *
 * Description :
 *
 * Ce fichier fait partie du code source C++ du logiciel portageMM2CPP.
 *
 * Il en contient la partie générique qui "se veut" indépendante du modèle
 * traité (déclarations des classes). Voir ../laDocumentation/help, en
 * particulier "Présentation générale" et "Spécification".
 *
 ************************************************************************
 * Historique :
 *
 * 03/02/06, Nathalie Rousse : création du fichier.
 * 26/10/07, Nathalie Rousse : modification mineure m_bool (utilisation du type predefini "bool" au lieu d'utiliser "in
t").
 * 26/10/07, Nathalie Rousse : modification m_type (definition de types : typedef typeValeur et typeTemps, qui "passent
" de float a au moins double).
 *
 * 02/01/08, Nathalie Rousse : modification m_linearInterpolation (ajout prise en compte des entités ModelMaker qui son
t calculées par interpolation linéaire à partir de données lues dans un fichier de données).
 *
 ************************************************************************/


/* modeDemploi
 * Instruction s'adressant à quelqu'un qui est en train de construire son propre logiciel C++ relativement à son propre
 modèle ModelMaker : dans tout le code de ce fichier, consulter et suivre les consignes/indications marquées du label "
modeDemploi". Voir aussi "Mode d'emploi" dans le fichier "help".
 */

/* modeDemploi
 *
 * Ce fichier (et son ".cpp") ne sont pas sensés être retouchés A PRIORI.
 *
 * Cependant, leur code est susceptible de bouger au titre d'évolutions, qui peuvent être nécessaires pour différentes
raisons (voir les commentaires marqués du label "modeDemploiEvolutions" dans tout le code de ce fichier (et son ".cpp")
).
 */

/* modeDemploiEvolutions
 * Il peut être besoin d'enrichir/modifier le code si le nouveau modèle, plus élaboré que celui de l'exemple "modele2.m
```

```
od", utilise des possibilités de ModelMaker qui ne sont pas prises en compte actuellement. Les besoins de ce type d'évo
lutions se révèleront généralement lors de l'écriture de la classe systemParticulier (en particulier de la méthode "act
ualiserIncrement").
 * A titre d'illustration : Actuellement, il est défini des sous-classes pour les différentes entités ModelMaker que co
ntient le modèle de l'exemple "modele2.mod" : constant, parameter, variable, flow, compartment. Il n'est pas défini de
sous-classe pour l'entité "component event" (n'a pas été nécessaire pour le modèle "modele2.mod" qui n'en comporte pas)
. Une évolution serait d'ajouter une classe pour l'entité ModelMaker "component event" actuellement non prise en compte
.
 */




#ifndef _GENERICITE_H_INCLUS_
#define _GENERICITE_H_INCLUS_


#include "genericiteConfigPortageMM2CPP.h"


// Constantes compte rendu d'exécution de fonctions :
#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, liste instants non strictement croissante...)
#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

// 26/10/07 m_type
/*
 * Type des donnees temps typeTemps, utilisé pour : instant, durée...
 */
typedef long double typeTemps; // typeTemps peut être mis double ou long double mais pas float

// 26/10/07 m_type
/*
 * Type des valeurs typeValeur, utilisé pour value, ddt
 * (remarque : l'homogénéité n'est pas marquée dans le sens où : ddt ~ typeValeur / typeTemps)
 */
```

```
typedef long double typeValeur; // typeValeur peut être mis double ou long double mais pas float

/*
 * Type pour les indices (colonnes, lignes...)
 */
typedef int typeIndice;

/*****************************************************************************
 *
 * classe entityInstantanee
 *
 * Description :
 *
 * Un diagramme ModelMaker contient différentes sortes d'entités : des
 * compartiments, flux, paramètres, variables, etc. Toutes ces entités
 * relèvent de la classe générique entityInstantanée "de base"
 * (instantanée pour relative au temps).
 *
 * La donnée principale de entityInstantanee est la valeur instantanée
 * "value".
 *
 *****************************************************************************/

class entityInstantanee {

        public :
                typeValeur value;
                typeValeur initialValue;
        public :
                /* les constructeurs */
                entityInstantanee(void);
                entityInstantanee(typeValeur iv);

                /* la fonction d'initialisation "de base" :
                 * les constructeurs appellent initialiser */
                void initialiser(typeValeur iv );

                typeValeur getValue(void);
                typeValeur getInitialValue(void);
};

/*****************************************************************************
 *
 * classe entityAvecDerivee
 *
 * Description :
 *
```

```
 * Certaines entités du diagramme ModelMaker sont définies, en plus de leur
 * valeur instantanée (value), par leur dérivée "ddt". La notion de dérivée
 * est l'apport/complément principal de la classe entityAvecDerivee par
 * rapport à sa classe mère entityInstantanee.
 *
 ******************************************************************************/

class entityAvecDerivee : public entityInstantanee {

        public :
                typeValeur ddt; /* dérivée value en fonction du temps */
        public :
                /* constructeur */
                entityAvecDerivee( void );
                entityAvecDerivee( typeValeur iv );
                entityAvecDerivee( typeValeur iv, typeValeur iddt );

                void initialiser(typeValeur iv );

                /* la fonction d'initialisation "de base" :
                 * les constructeurs et autres fonctions initialiser
                 * appellent initialiser */
                void initialiser(typeValeur iv, typeValeur iddt );

                /* calcule la nouvelle valeur instantanée de "value"
                 * en fonction de sa valeur à l'instant précédent et de
                 * la dérivée (relative au deltat de temps "deltat") */
                void actualiser(typeTemps deltat);
};

// m_linearInterpolation : nouvelle classe
/*****************************************************************************
 *
 * la classe fichierDentree
 *
 * Description :
 *
 * Classe 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.
 *
 * Description/explication un peu plus détaillée :
 *
 * La classe fichierDentree porte sur un fichier "à l'état brut". Il est
 * question de fichier uniquement d'entrée (d'accès en lecture et non pas en
 * écriture). La classe traite de questions de base comme de déterminer si le
 * fichier est valide, s'il est prêt à être lu, elle contient la méthode
```

```
 * unitaire lireFloat qui permet de lire une donnée dans le fichier.
 *
 * Comment est utilisée la classe fichierDentree :
 * Une classe de niveau supérieur (ie appelante telle que la classe
 * donneesAinterpoler dont l'attribut fichierSource est un objet
 * fichierDentree) sert de couche entre fichierDentree et ses classes
 * utilisatrices (telles que la classe entityInterpolee).
 * Dans la classe de niveau supérieur (comme donneesAinterpoler), il est
 * écrit les méthodes permettant :
 * - d'accéder aux services fournis par fichierDentree,
 * - de regarder les données privées (comme estValide) de fichierDentree
 *   (cf donneesAinterpoler::getFichierSourceEstValide),
 * - et si nécessaire d'agir sur ces données
 *   (cf donneesAinterpoler::setFichierSourceEstValide).
 * Par exemple, la classe donneesAinterpoler appelle les méthodes unitaires
 * preparerLecture, lireFloat et terminerLecture de la classe fichierDentree
 * pour écrire une méthode de lecture plus élaborée telle que
 * donneesAinterpoler::lireDonneeDeCtrlEtDonneeControlee.
 * La classe entityInterpolee utilisatrice de la classe fichierDentree
 * n'appelle pas directement cette classe. Elle passe par la classe de
 * niveau supérieur donneesAinterpoler (l'attribut lesDonneesAinterpoler de
 * la classe entityInterpolee est un objet donneesAinterpoler).
 * Des méthodes comme leFichierDonneesAinterpolerEstValide,
 * activerFichierDonneesAinterpolerEstValide,
 * desactiverFichierDonneesAinterpolerEstValide de la classe utilisatrice
 * entityInterpolee donnent accès aux informations de la classe fichierDentree
 * (via appel de méthodes de la classe intermédiaire donneesAinterpoler :
 * getFichierSourceEstValide, setFichierSourceEstValide) sous une forme
 * exprimée "d'un point de vue utilisateur".
 *
 ****************************************************************************/

/* Constante associée (voir fichier genericiteConfigPortageMM2CPP.h) :
 * TAILLE_MAX_NOM_FICHIER */

class fichierDentree {

        private :
                char nomFichier[ TAILLE_MAX_NOM_FICHIER ]; // nom du fichier

                ifstream inputFile; // pour opérer sur le fichier

                /*
                 * Des indicateurs d'état du fichier :
                 * preparerLecture() n'agit que si estValide est true.
                 * lireFloat() vérifie que estPreparePourLecture est true pour agir (lire donnée dans fichier).
                 */
```

```
                bool estValide;
                bool estPreparePourLecture;

        public :
                /* Constructeur */
                fichierDentree( void );

                /*
                 * La fonction d'initialisation "de base" est partagée entre :
                 * - initialiser(void) qui est appelée par les constructeurs et autres fonctions initialiser
                 * - et initialiser(char *) qui prend en compte le nom de fichier.
                 */
        private :
                void initialiser( void ); // initialisation par défaut
        public :
                /* Initialise nomFichier, estValide, estPreparePourLecture */
                void initialiser( char *nom );

        public :
                /*
                 * Les fonctions preparerLecture, terminerLecture et lireFloat
                 * servent à lire dans le fichier : elles sont conçues
                 * pour être utilisées dans l'ordre suivant :
                 *  - appel de preparerLecture,
                 *  - (succession d')appel(s) de lireFloat(),
                 *  - appel de terminerLecture.
                 */
                void preparerLecture();
                /* Lit dans le fichier (avec inputFile) la donnée donneeLue
                 * au format float. Retourne true si donneeLue a effectivement
                 * été lue et false sinon (cas de fin de fichier...). */
                bool lireFloat( float *donneeLue );
                void terminerLecture();

                /* set et get */
        private :
                void setNomFichier( char *nom );
        public :
                char *getNomFichier();
        private :
                void setEstPreparePourLecture( bool estPreparePourLectureValue );
        private :
                bool getEstPreparePourLecture();
        public :
                void setEstValide( bool estValideValue );
        public :
                bool getEstValide();
```

```
};

// m_linearInterpolation : nouvelle classe
/*****************************************************************************
 *
 * la classe donneesAinterpoler
 *
 * Description :
 *
 * Classe 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.
 *
 * Format du fichier contenant la liste de données à interpoler :
 * à chaque ligne :
 *    valueDonneeDeCtrl dans la colonneDonneeDeCtrl
 *    valueDonneeControlee dans la colonneDonneeControlee
 * (ie valueDonneeControlee(valueDonneeDeCtrl) )
 *
 * Remarque : Un fichier contenant N colonnes correspondant à N données à
 * interpoler est susceptible de donner lieu à N objets de la classe
 * donneeAinterpoler (et donc d'être lu N fois en parallèle).
 *
 * Remarque : Les conditions de simulation doivent respecter une bonne
 * utilisation du fichier (ie utilisation conforme à la signification de son
 * contenu).
 *
 * Voir aussi "Description/explication un peu plus détaillée" dans l'entête
 * de la classe fichierDentree.
 *
 *****************************************************************************/

class donneesAinterpoler {

        private :
                fichierDentree fichierSource;
        private :

                typeIndice nbColonnes; // nombre de colonnes du fichierSource

                /* Indices de colonnes dans le fichierSource
                 * donneeDeCtrl = valeurFichier( colDonneeDeCtrl )
                 * donneeControlee(donneeDeCtrl) = valeurFichier( colDonneeControlee ) */
```

```
                typeIndice colDonneeDeCtrl; // dans le fichierSource
                typeIndice colDonneeControlee; // dans le fichierSource

        public :
                /* constructeurs et initialisations */
                void initialiser( char *nomFichier,
                                  typeIndice inbColonnes,
                                  typeIndice icolDonneeDeCtrl,
                                  typeIndice icolDonneeControlee );

                /* Méthodes en lien avec fichierDentree (pour plus voir
                 * "Description/explication un peu plus détaillée"
                 * dans l'entête de la classe fichierDentree). */
                // set et get
                void setFichierSourceEstValide( bool estValideValue );
                bool getFichierSourceEstValide( void );
                char *getNomFichierSource( void );

                /* Les méthodes lireDonneeDeCtrlEtDonneeControlee,
                 * preparerLecture et terminerLecture sont en lien
                 * avec fichierDentree (pour plus voir
                 * "Description/explication un peu plus détaillée"
                 * dans l'entête de la classe fichierDentree). */
                /*
                 * lit entièrement la ligne courante du fichierSource
                 * (ie lit nbColonnes float)
                 * et rend donneeDeCtrl et donneeControlee en paramètres.
                 * donneeBesoinNonTrouveeDansFichier vaut true s'il a fallu
                 * lire une donnée dans le fichier et qu'elle n'y a
                 * pas été trouvée (fin de fichier a priori)
                 */
                void lireDonneeDeCtrlEtDonneeControlee( float *ptr_donneeDeCtrl,
                                                        float *ptr_donneeControlee,
                                                        bool *donneeBesoinNonTrouveeDansFichier );
                /*
                 * Appeler preparerLecture avant une succession d'appels de
                 * lireDonneeDeCtrlEtDonneeControlee.
                 * Appeler terminerLecture après une succession de
                 * lireDonneeDeCtrlEtDonneeControlee qui s'est bien déroulée.
                 * Pour plus voir fichierDentree.
                 */
                void preparerLecture( void );
                void terminerLecture( void );

};

// m_linearInterpolation : nouvelle classe
```

```
/*****************************************************************************
 *
 * classe entityInterpolee
 *
 * Description :
 *
 * Certaines entités du diagramme ModelMaker sont calculées par interpolation.
 * Leur valeur instantanée (value) est calculée (*) à partir de celle de
 * 2 instants d'interpolation (un instant antérieur et un instant postérieur),
 * par interpolation linéaire.
 *
 * Les informations nécessaires à la gestion de ce calcul sont
 * l'apport/complément principal de la classe entityInterpolee par
 * rapport à sa classe mère entityInstantanee.
 *
 * 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.
 * Ici (classe entityInterpolee), il n'est plus question de donneeDeCtrl au
 * sens large. donneeDeCtrl est fixé forcément à l'information instant/time.
 *
 * (*) Plus précisément :
 *
 * Un fichier lesDonneesAinterpoler.fichierSource contient la liste des
 * données à interpoler.
 *
 * Calcul de la valeur instantanée (value) à l'instant courant :
 * - Soit l'instant courant est un instant d'interpolation : value est alors
 *   issu du fichier lesDonneesAinterpoler.fichierSource.
 * - Soit l'instant courant se trouve entre 2 instants d'interpolation (après
 *   tPrec et avant tSuiv, de valeurs respectives valuePrec et valueSuiv
 *   issues du fichier lesDonneesAinterpoler.fichierSource) : value est alors
 *   calculé par interpolation à partir de valuePrec et valueSuiv.
 *
 * Voir aussi "Description/explication un peu plus détaillée" dans l'entête
 * de la classe fichierDentree.
 *
 *****************************************************************************/

class entityInterpolee : public entityInstantanee {

        private :
                /*
                 * value aux instants tPrec et tSuiv d'interpolation qui
                 * encadrent au plus près l'instant courant.
```

```
                * valuePrec et valueSuiv sont lues dans
                * lesDonneesAinterpoler.fichierSource ainsi que
                * tPrec et tSuiv.
                */
               typeValeur valuePrec;
               typeValeur valueSuiv;
               typeTemps tPrec;
               typeTemps tSuiv;
               typeTemps tDernier; // instant de la dernière donnée du fichier

               /*
                * donneesAinterpoler relatif au fichier contenant
                * les données à interpoler (où elles sont lues)
                */
               donneesAinterpoler lesDonneesAinterpoler;

       public :
               /* constructeurs */
               entityInterpolee( void );
               entityInterpolee( char *nomFichier,
                                 typeIndice inbColonnes,
                                 typeIndice icolDonneeDeCtrl,
                                 typeIndice icolDonneeControlee );

               /*
                * Initialise lesDonneesAinterpoler.
                * Initialise tPrec, tSuiv, valuePrec, valueSuiv
                * à partir de la 1ère et de la 2ème ligne lues dans
                * lesDonneesAinterpoler.fichierSource
                */
               void initialiser( char *nomFichier,
                                 typeIndice inbColonnes,
                                 typeIndice icolDonneeDeCtrl,
                                 typeIndice icolDonneeControlee );

               /* informations et vérifications */

               bool leFichierDonneesAinterpolerEstValide( void );
               void activerFichierDonneesAinterpolerEstValide( void );
               void desactiverFichierDonneesAinterpolerEstValide( void );
               char *getNomFichierDonneesAinterpoler( void );
               typeTemps getPremierInstantDonneesAinterpoler( void );
               typeTemps getDernierInstantDonneesAinterpoler( void );

               /*
                * Vérifie l'utilisabilité du fichier lesDonneesAinterpoler.fichierSource :
                * vérifie son autocohérence et la cohérence de son contenu
```

```
                * avec son utilisation dans la simulation
                * (cf tDebutDeLaSimulation et tMaxDeLaSimulation).
                * Retourne CR_OK ou CR_NOK_AVERTISSEMENT ou CR_NOK_BLOQUANT.
                */
               int utilisabiliteDonneesAinterpoler( typeTemps tDebutDeLaSimulation,
                                                    typeTemps tMaxDeLaSimulation );

               /* affiche à l'écran les conclusions de la méthode utilisabiliteDonneesAinterpoler
                * si on est hors cas nominal */
               void afficherProblemeUtilisabiliteDonneesAinterpoler( int crUtilisabiliteDonneesAinterpoler,
                              typeTemps tDebutDeLaSimulation, typeTemps tMaxDeLaSimulation );

               /*
                * On décale (avance) l'intervalle d'interpolation linéaire
                * [ (tPrec,valuePrec), (tSuiv,valueSuiv) [ :
                * (tPrec,valuePrec) <-- (tSuiv,valueSuiv) et
                * (tSuiv,valueSuiv) <-- (tNew,valueNew)
                */
               void decalerIntervalleInterpolationLineaire( typeTemps tNew, typeValeur valueNew );

       private :
               /*
                * Calcule la pente d'interpolation linéaire relative à
                * (tPrec, valuePrec) et (tSuiv, valueSuiv).
                * Peut retourner CR_NOK_tentativeDivisionParZero.
                */
               int calculerPenteInterpolationLineaire( typeValeur *pente );
       public :
               /*
                * Interpolation linéaire pour (tCourant, valueCourante) à partir de
                * (tPrec, valuePrec) et (tSuiv, valueSuiv).
                * Remarque : normalement tPrec et tSuiv encadrent tCourant au plus près
                * (tPrec <= tCourant < tSuiv), si ce n'est pas le cas alors peut
                * conduire à une extrapolation linéaire.
                * Peut retourner CR_NOK_tentativeDivisionParZero.
                */
               int interpolationLineaire( typeTemps tCourant, typeValeur *valueCourante );

               /*
                * Calcule la nouvelle valeur instantanée (ie d'instant tCourant)
                * de "value" par interpolation linéaire.
                * donneeBesoinNonTrouveeDansFichier vaut true s'il a fallu
                * lire une donnée dans le fichier et qu'elle n'y a pas été
                * trouvée (fin de fichier a priori).
                * crCalculInterpolationlineaire rend compte du calcul
                * d'interpolation linéaire; peut valoir CR_NOK_tentativeDivisionParZero.
                */
```

```
                void actualiser( typeTemps tCourant,
                                 bool *donneeBesoinNonTrouveeDansFichier,
                                 int *crCalculInterpolationlineaire );

};


/*****************************************************************************
 *
 * Les sous-classes des différentes entités ModelMaker
 *
 * Description :
 *
 * Les entités ModelMaker relèvent de la classe entityInstantanée, voire
 * si nécessaire de la classe entityAvecDerivee.
 *
 *****************************************************************************/

/* classe des constantes ModelMaker */
class constant : public entityInstantanee {
        public : int bidon;
};
/* classe des paramètres ModelMaker */
class parameter : public entityInstantanee {
        public : int bidon;
};
/* classe des variables ModelMaker */
class variable : public entityInstantanee {
        public : int bidon;
};
/* classe des flux ModelMaker */
class flow : public entityInstantanee {
        public : int bidon;
};
/* classe des compartiments ModelMaker */
class compartment : public entityAvecDerivee {
        public : int bidon;
};

// m_linearInterpolation : nouvelle classe
/* classe des données sous contrôle ModelMaker */
class controlled : public entityInterpolee {
        public : int bidon;
};


/*****************************************************************************
```

```
 *
 * la classe horlogeLocale
 *
 *
 *             TOUT SYSTEME DOIT CONTENIR UNE HORLOGE LOCALE.
 *
 *              L'unité de temps est la SECONDE.
 *
 *
 * Description :
 *
 * Les valeurs et dérivées des entités ModelMaker sont relatives au temps.
 * L'horloge locale sert de "référentiel" de temps. Elle contient :
 * l'instant de démarrage "tInitial", l'instant courant "t" évoluant au fil
 * des incréments temporels et le pas de temps "deltat" selon lequel est
 * incrémenté l'instant courant.
 *
 ******************************************************************************/

/* Constantes associées (voir fichier genericiteConfigPortageMM2CPP.h) :
 * INSTANT_INITIAL_DEFAUT, DELTA_T_DEFAUT */

class horlogeLocale {

        public :
                typeTemps t;            /* instant courant */
                typeTemps tInitial;     /* instant initial */
                typeTemps deltat;       /* pas de temps (plus petit increment) */

        public :
                /* les constructeurs */
                horlogeLocale(void);
                horlogeLocale(typeTemps it);
                horlogeLocale(typeTemps it, typeTemps deltatValue);

                void initialiser(void);
                void initialiser(typeTemps it);

                /* la fonction d'initialisation "de base" :
                 * les constructeurs appellent initialiser */
                void initialiser(typeTemps it, typeTemps deltatValue);

                /* redonne à l'instant courant la valeur tInitial */
                void retourAinstantInitial(void);

                typeTemps getInstantCourant(void);
                typeTemps getInstantInitial(void);
```

```
                typeTemps getPasDeTemps(void);

                /* change la valeur de tInitial (sans toucher l'instant courant) */
                void setInstantInitial(typeTemps it);

                /* incrément l'instant courant t de deltat */
                void incrementerHorloge(void);

                void afficherConfigurationHorloge(void);
};


/*****************************************************************************
 * Types utilisés par listeEntitesAvecDerivee et listeEntitesInterpolees
 *****************************************************************************/
typedef entityAvecDerivee *pointeurDentityAvecDerivee;
typedef entityInstantanee *pointeurDentityInstantanee;
typedef entityInterpolee  *pointeurDentityInterpolee;


/*****************************************************************************
 *
 * classe listeEntitesAvecDerivee
 *
 * Description :
 *
 * Cette classe gère une liste (privée) permettant d'automatiser certaines
 * opérations, comme le calcul des valeurs ("value") des entités en fonction
 * des dérivées ("ddt").
 *
 *****************************************************************************/
/* m_linearInterpolation : prise en compte dans listeEntitesAvecDerivee de l'ajout de listeEntitesInterpolees */

/* Constante associée (voir fichier genericiteConfigPortageMM2CPP.h) :
 * NB_MAX_EntitesAvecDerivee */

class listeEntitesAvecDerivee {

        public :
                int nb;
                /* La liste "liste", en pointant l'ensemble des objets
                 * instanciant la classe entityAvecDerivee permet de mettre
                 * en place une boucle de calcul des valeurs des entités en
                 * fonction des dérivées */
                pointeurDentityAvecDerivee liste[ NB_MAX_EntitesAvecDerivee ];
        public :
                listeEntitesAvecDerivee( void );
```

```
                int PL( pointeurDentityAvecDerivee ptr );
                int PL( pointeurDentityInstantanee ptr );
                int PL( pointeurDentityInterpolee ptr );
                void AL( typeTemps deltat );
        private :
                /* Indicateur booléen permettant de repérer
                 * dès qu'il y a eu au moins un appel de "PL".
                 * Pour l'instant est géré mais pas utilisé */
                // 26/10/07 m_bool
                bool aucuneInitialisation;
};


// m_linearInterpolation : nouvelle classe
/*****************************************************************************
 *
 * classe listeEntitesInterpolees
 *
 * Description :
 *
 * Cette classe gère une liste (privée) permettant d'automatiser certaines
 * opérations, comme le calcul des valeurs ("value") des entités interpolées.
 *
 *****************************************************************************/

/* Constante associée (voir fichier genericiteConfigPortageMM2CPP.h) :
 * NB_MAX_EntitesInterpolees */

class listeEntitesInterpolees {

        public :
                int nb;
                /* La liste "liste", en pointant l'ensemble des objets
                 * instanciant la classe entityInterpolee permet de mettre
                 * en place une boucle de calcul des valeurs des entités
                 * interpolées */
                pointeurDentityInterpolee liste[ NB_MAX_EntitesInterpolees ];
        public :
                listeEntitesInterpolees( void );
                int PL( pointeurDentityInterpolee ptr );
                int PL( pointeurDentityAvecDerivee ptr );
                int PL( pointeurDentityInstantanee ptr );
                int AL( typeTemps tCourant );
                int utilisabiliteDonneesAinterpoler( typeTemps tDebutDeLaSimulation,
                                                     typeTemps tMaxDeLaSimulation );
                void afficherNomsFichiersDonneesAinterpoler();
        private :
```

```
                /* Indicateur booléen permettant de repérer
                 * dès qu'il y a eu au moins un appel de "PL".
                 * Pour l'instant est géré mais pas utilisé */
                bool aucuneInitialisation;
};

/*****************************************************************************
 *
 * classe compteRenduInitialisation
 *
 *****************************************************************************/

class compteRenduInitialisation {
        private :
                int crInitValue;
        public :
                compteRenduInitialisation( void );
                void putValue( int value );
                int getValue( void );
                // 26/10/07 m_bool
                bool pasDePbInitReleve( void );
};

// m_linearInterpolation : nouvelle classe
/*****************************************************************************
 *
 * classe gestionAvancementSimulation
 *
 * Description :
 *
 * Cette classe 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...
 *
 *****************************************************************************/
class gestionAvancementSimulation {

        public :
                /* Marqueur des instants correspondant à des pas de
                 * simulation (marqueurPasDeLaSimulation sera incrémenté de
                 * pasDeLaSimulation en pasDeLaSimulation au fur et à mesure
                 * qu'on progresse dans la simulation) */
                typeTemps marqueurPasDeLaSimulation;

                /* dernierInstantPasDeLaSimulation mémorise le dernier
                 * instant passé qui a été traité comme un pas de simulation
```

```
                 * (pasDeLaSimulation).
                 * Notamment dernierInstantPasDeLaSimulation permet de
                 * connaitre avec exactitude l'écart entre les deux derniers
                 * instants de simulation traités
                 * (quand onEstSurUnPasDeLaSimulation, juste avant sa maj cet
                 * écart vaut :
                 * time.getInstantCourant() - dernierInstantPasDeLaSimulation).
                 */
                typeTemps dernierInstantPasDeLaSimulation;

                /* Ce booleen est true si, du fait de l'exécution de
                 * incrementerHorloge, "on se retrouve sur" (en fait on vient
                 * juste de passer) un pas de simulation pasDeLaSimulation */
                bool onEstSurUnPasDeLaSimulation;

                /* Si en cours de route à un instant courant, la simulation
                 * doit est arrêtée, alors finPrematuree passe à true */
                bool finPrematuree;

        public :
                /* Constructeurs et initialisations */
                void initialiser( typeTemps tDebutDeLaSimulation,
                                  typeTemps pasDeLaSimulation );

                /* Mise à jour relativement à l'instant courant
                 * instantCourant (auquel on vient de passer) */
                void maj( typeTemps instantCourant );

                /* Une fois traité l'instant courant instantCourant,
                 * mise à jour en vue des instants suivants */
                void majPourlaSuite( typeTemps instantCourant,
                                     typeTemps pasDeLaSimulation );
};


/*****************************************************************************
 *
 * classe genericSystem
 *
 *****************************************************************************/

/* Constante associée (voir fichier genericiteConfigPortageMM2CPP.h) :
 * PAS_SIM_SUR_PAS_TEMPS_MIN */

class genericSystem {

        /*********************************************************************
```

```
         * Horloge locale
         ************************************************************************/
        public :

                // tout système possède une horloge locale
                horlogeLocale time;

        /************************************************************************
         * Les données principales de simulation
         ************************************************************************/
        private :
                /*
                 * Instant auquel démarre la simulation.
                 * "tDebutDeLaSimulation" est une "sortie" de la fonction
                 * "simuler" de base
                 */
                typeTemps tDebutDeLaSimulation;

                /*
                 * Instant auquel "s'arrete" la simulation; plus exactement,
                 * la simulation s'arrete juste avant de dépasser cet instant.
                 * "tMaxDeLaSimulation" est une sortie de la fonction
                 * "simuler" de base (calcule a partir de "dureeDeLaSimulation")
                 */
                typeTemps tMaxDeLaSimulation;

                /*
                 * Duree de la simulation ie de l'intervalle
                 * [ tDebutDeLaSimulation, tMaxDeLaSimulation ]
                 * "dureeDeLaSimulation" est une "entree" de la fonction
                 * "simuler" de base
                 */
                typeTemps dureeDeLaSimulation;

                /*
                 * pasDeLaSimulation : pas de temps entre 2 itérations de
                 * la simulation.
                 *
                 * "pasDeLaSimulation" est une "entree" de la fonction
                 * "simuler" de base.
                 *
                 * Remarque 1 :
                 *
                 *   pasDeLaSimulation est le pas de temps VISIBLE.
                 *   pasDeLaSimulation est la durée écoulée entre 2 instants
                 *   où sont sortis des résultats de simulation.
                 *   Mais en réalité il est exécuté une simulation à chaque
```

```
                *   instant : tDebutDeLaSimulation + n * deltat, n=0,1,2...
                *
                * Remarque 2 :
                *
                *   La valeur choisie pour pasDeLaSimulation, si elle ne
                *   contient pas au moins PAS_SIM_SUR_PAS_TEMPS_MIN fois le
                *   "pas de temps de base" (deltat de l'horloge), sera
                *   modifiée (augmentée) pour que ça devienne le cas.
                */
               typeTemps pasDeLaSimulation;


       /*************************************************************
        * Les données (liste) pour automatisation des calculs
        *************************************************************/
       public :
               listeEntitesAvecDerivee ead;
               listeEntitesInterpolees eil; // m_linearInterpolation : ajout

       /*************************************************************
        * La donnée de gestion de l'avancement dans la simulation
        *************************************************************/
       public :
               gestionAvancementSimulation gSimu; // m_linearInterpolation : ajout

       /*************************************************************
        * Les données et méthodes de controle d'initialisation
        *************************************************************/
       public :
               /* Précision sur la signification de crInit, la manière dont
                * il est traité/géré :
                * - Si sa valeur (crInit.crInitValue) vaut CR_OK :
                *   L'initialisation s'est bien passée.
                * - Sinon :
                *   Au moins un problème est arrivé en cours d'initialisation.
                *   Au fil du déroulement de l'initialisation, la valeur de
                *   crInit peut avoir été affectée plusieurs fois. La valeur
                *   de crInit n'informe que sur un seul problème arrivé lors
                *   de l'initialisation (celui de la dernière affectation).
                */
               compteRenduInitialisation crInit;

       public :
               int getCrInit( void ){ return crInit.getValue(); };
               // 26/10/07 m_bool
               bool pasDePbInitReleve( void ){
                       return ( crInit.pasDePbInitReleve() );
```

```
		};

	/*************************************************************************
	 * Les méthodes du systeme
	 *************************************************************************/
	public :
		/* initialisation avec horloge locale par défaut */
		void initialiser(void);

		/* initialisation avec configuration de l'horloge locale :
		 * - instant initial "it",
		 * - pas de temps (plus petit increment) "deltatValue" */
		void initialiser( typeTemps it, typeTemps deltatValue);

		virtual int actualiserIncrement(void){
			return CR_NOK_virtualNonDefinie;
		}
		virtual int afficherEntetesEntites(void){
			return CR_NOK_virtualNonDefinie;
		}
		virtual int afficherEntites(void){
			return CR_NOK_virtualNonDefinie;
		}

	/* m_linearInterpolation : suppression de la fonction "private : void incrementer(void);" (son code a ete copie
 a la place de son ancien appel) */

	/*************************************************************************
	 * Les méthodes de simulation
	 *************************************************************************/
	public :

/* modeDemploiEvolutions
		 *
		 * Il peut être besoin d'implémenter de nouvelles fonctions de simulation : pour sa propre utilisation,
 quelqu'un peut être amené à écrire une nouvelle méthode "simuler" publique (méthode "chapeau" appelant la méthode de s
imulation de base "simuler" privée).
		 */


		/*
		 * Une methode de simulation, appel du modele :
		 *
		 * Appelle la methode de simulation de base "simuler" de
		 * maniere que la boucle d'exécution de la simulation
		 * se fasse ainsi :
		 * Demarrage de la simulation a "t", l'instant courant de
```

```
         * l'horloge (qui n'est pas retouche ici avant de commencer
         * la simulation).
         * Arret de la simulation apres une duree
         * de "nbIterations"x"pasDeSimulation".
         * Periode d'affichage des resultats de la simulation (ie
         * l'etat du modele) : "pasSimulation", entre les instants
         * de debut et de fin de la simulation.
         */
        void simuler(int nbIterations, typeTemps pasSimulation);


        /*
         * Une methode de simulation, appel du modele :
         *
         * Modifie l'horloge (l'instant initial "tInitial" et
         * l'instant courant "t" de l'horloge sont mis à la valeur
         * "tDebut").
         *
         * Puis appelle la methode de simulation de base "simuler"
         * de maniere que la boucle d'exécution de la simulation
         * se fasse ainsi :
         * Demarrage de la simulation a "t", l'instant courant de
         * l'horloge (qui a prealablement ete ramene a "tDebut").
         * Arret de la simulation apres une duree de "dureeSimulation".
         * Periode d'affichage des resultats de la simulation (ie
         * l'etat du modele) : "pasSimulation", entre les instants de
         * debut et de fin de la simulation.
         */
        void simuler(typeTemps tDebut, typeTemps dureeSimulation, typeTemps pasSimulation );

    private :

        void initialiserDonneesSimulation(void);

        /*
         * La methode de simulation (appel du modele) "de base" :
         *
         * Toutes les autres methodes de simulation (diverses
         * "simuler" publiques) appellent cette methode privee
         * "simuler".
         *
         * Avant d'appeler cette fonction il faut definir
         * dureeDeLaSimulation et pasDeLaSimulation qui en sont des
         * entrées.
         * Cette fonction definit tDebutDeLaSimulation et
         * tMaxDeLaSimulation qui sont des "sorties" de la fonction.
         */
```

```
                int simuler(void);
};

#endif /* _GENERICITE_H_INCLUS_ */

/*****************************************************************************
 *                                                                           *
 * Fin du fichier : genericitePortageMM2CPP.h                                *
 *                                                                           *
 *****************************************************************************/
```