```
/*************************************************************************
 *  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 -
 *               ( UMR ARCHE / Plate-forme INRA-ACTA-ICTA ).
 *
 * 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.
 *
 ******************************************************************************/


/* 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_

#define VRAI 1
#define FAUX 0

/*
 * 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



/*****************************************************************************
 *
 * 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 :
                float value;
                float initialValue;
        public :
                /* les constructeurs */
                entityInstantanee(void);
```

```
                entityInstantanee(float iv);

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

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

                void initialiser(float iv );

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


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

```
 *
 * 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;
};

/******************************************************************************
 *
 * 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.
```

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

#define INSTANT_INITIAL_DEFAUT 0.0      // instant initial par défaut : 0 seconde
/* modeDemploi_ConfDim
 * Eventuellement, changer la valeur de INSTANT_INITIAL_DEFAUT si c'est souhaité
 */
#define DELTA_T_DEFAUT 0.001    // pas de temps par défaut : 0.001 seconde
/* modeDemploi_ConfDim
 * Eventuellement, changer la valeur de DELTA_T_DEFAUT si c'est souhaité
 */

class horlogeLocale {

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

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

                void initialiser(void);
                void initialiser(float it);

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

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

                float getInstantCourant(void);
                float getInstantInitial(void);
                float getPasDeTemps(void);

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

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

                void afficherConfigurationHorloge(void);
};
```

```
/*****************************************************************************
 *
 * 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").
 *
 *****************************************************************************/

/* nombre maximum d'objets declares de type entityAvecDerivee */
#define NB_MAX_EntitesAvecDerivee 10000

/* modeDemploi_ConfDim
 * Verifier que NB_MAX_EntitesAvecDerivee majore bien le nombre d'objets de type entityAvecDerivee (ou ses classes fill
es) qui sont déclarés dans la classe systemParticulier. Si ce n'est pas le cas, augmenter sa valeur pour que ça devienn
e vrai.
 * Suggestion : comme les sous-classes associées aux entités ModelMaker (constant, parameter, variable, flow, compartme
nt) sont filles soit de la classe entityInstantanee, soit de la classe entityAvecDerivee, il suffit de choisir NB_MAX_E
ntitesAvecDerivee supérieur au nombre d'éléments ModelMaker qui sont déclarés dans la classe systemParticulier.
 */


typedef entityAvecDerivee *pointeurDentityAvecDerivee;
typedef entityInstantanee *pointeurDentityInstantanee;

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 );
                void AL( float deltat );
        private :
                /* Indicateur booléen permettant de repérer
                 * dès qu'il y a eu au moins un appel de "PL" */
                int aucuneInitialisation;
};
```

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

class compteRenduInitialisation {
        private :
                int crInitValue;
        public :
                compteRenduInitialisation( void );
                void putValue( int value );
                int getValue( void );
                int pasDePbInitReleve( void );
};


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

/*
 * Valeur minimale du rapport pasDeLaSimulation/deltat.
 * Cette constante servira à "ajuster" (en fait augmenter) pasDeLaSimulation
 * de telle sorte qu'il y ait au moins PAS_SIM_SUR_PAS_TEMPS_MIN itérations
 * deltat dans la durée pasDeLaSimulation
 */
#define PAS_SIM_SUR_PAS_TEMPS_MIN 1000 // est-ce suffisant ? (il n'y a aucun "filet")

/* modeDemploi_ConfDim
 * S'assurer que PAS_SIM_SUR_PAS_TEMPS_MIN n'est pas trop faible par rapport à son système particulier (en fait, par ra
pport aux équations de "actualiserIncrement"). Si cela est besoin, augmenter sa valeur.
 * Indications : Plus PAS_SIM_SUR_PAS_TEMPS_MIN est grande, plus les calculs seront précis, mais aussi plus l'exécution
 des simulations sera lente. Il s'agit de trouver le compromis entre la précision des calculs et la vitesse de calcul.
 * ATTENTION : Il vaut mieux sur estimer que sous estimer cette valeur. Si PAS_SIM_SUR_PAS_TEMPS_MIN est trop faible, i
l y a risque d'ERREURS de calculs (notamment dans le calcul des dérivées). C'est un POINT TRES DELICAT car il n'est fai
t aucun controle dans le code. Il faut déceler un éventuel problème lors des tests (lancement et vérifications de simul
ations).
 */

class genericSystem {

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

```
        // tout système possède une horloge locale
        horlogeLocale time;

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

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


        /**************************************************************************
         * La donnée (liste) pour automatisation des calculs
         *************************************************************************/
        public :
                listeEntitesAvecDerivee ead;

        /**************************************************************************
         * Les données et méthodes de controle d'initialisation
         *************************************************************************/
        public :
                compteRenduInitialisation crInit;

        public :
                int getCrInit( void ){ return crInit.getValue(); };
                int 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( float it, float 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;
                }
        private :
                void incrementer(void);
```

```
        /*******************************************************************
         * 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, float 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(float tDebut, float dureeSimulation, float 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_ */
```