```
/*************************************************************************
 *  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.cpp
 *
 * 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 (corps 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".
 */

#include <iostream>
using namespace std;

#include "genericitePortageMM2CPP.h"

/************************************************************************
 *
 * corps de la classe entityInstantanee
 *
 ************************************************************************/

entityInstantanee::entityInstantanee(void){ initialiser(0.0); } // par défaut valeur nulle
entityInstantanee::entityInstantanee(float iv){ initialiser(iv); }

void entityInstantanee::initialiser(float iv ){
        value=iv; initialValue=iv;
}

float entityInstantanee::getValue(void){ return value; }
float entityInstantanee::getInitialValue(void){ return initialValue; }
```

```
/*****************************************************************************
 *
 * corps de la classe entityAvecDerivee
 *
 *****************************************************************************/

entityAvecDerivee::entityAvecDerivee( void ){
        initialiser(0.0, 0.0); // par défaut valeur et dérivée nulles
}
entityAvecDerivee::entityAvecDerivee( float iv ){
        initialiser(iv, 0.0); // par défaut dérivée nulle
}
entityAvecDerivee::entityAvecDerivee( float iv, float iddt ){
        initialiser(iv, iddt );
}

void entityAvecDerivee::initialiser (float iv){
        initialiser(iv, 0.0); // dérivée nulle par défaut
}
void entityAvecDerivee::initialiser(float iv, float iddt ){
        entityInstantanee::initialiser(iv); ddt=iddt;
}

/*
 * calcule/met à jour la valeur instantanée "value" en fonction de
 * sa d/dt ("ddt")
 */
void entityAvecDerivee::actualiser(float deltat){
        value = value + ddt * deltat;
}

/*****************************************************************************
 *
 * le corps de la classe horlogeLocale
 *
 *****************************************************************************/

horlogeLocale::horlogeLocale(void ){
        initialiser( INSTANT_INITIAL_DEFAUT, DELTA_T_DEFAUT );
}
horlogeLocale::horlogeLocale( float it){
        initialiser( it, DELTA_T_DEFAUT );
}

horlogeLocale::horlogeLocale( float it, float deltatValue){
        initialiser( it, deltatValue);
}
```

```
void horlogeLocale::initialiser( void ){ // initialisation par défaut
        initialiser( INSTANT_INITIAL_DEFAUT, DELTA_T_DEFAUT );
}

void horlogeLocale::initialiser( float it){
        initialiser( it, DELTA_T_DEFAUT );
}

void horlogeLocale::initialiser( float it, float deltatValue){
        tInitial = it; deltat = deltatValue;
        retourAinstantInitial(); // ramène l'instant courant "t" à tInitial
}

/* redonne à l'instant courant la valeur tInitial */
void horlogeLocale::retourAinstantInitial(void){
        t = tInitial - deltat; // "-deltat" en vue de la 1ère incrémentation
}

float horlogeLocale::getInstantCourant(void){ return t; }
float horlogeLocale::getInstantInitial(void){ return tInitial; }
float horlogeLocale::getPasDeTemps(void){ return deltat; }

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

void horlogeLocale::incrementerHorloge(void){
        t = t + deltat;
}

void horlogeLocale::afficherConfigurationHorloge(void){
        cout << " ***** les données horloge locale : " << endl;
        cout << "l'instant initial est : tInitial=" << tInitial << " s, " << endl;
        cout << "le pas de temps (plus petit incrément) est : deltat=" << deltat << " s." << endl;
        cout << "(valeur 'actuelle' de l'instant courant : t=" << t << " s.)" << endl << endl;
}

/*****************************************************************************
 *
 * corps de la classe genericSystem
 *
 *****************************************************************************/

/*
 *
 * Constructeurs, initialisations
```

```
 *
 */

// Initialisation par défaut
void genericSystem::initialiser(void){

        /* horloge locale */
        time.initialiser(); // initialisation aux valeurs par défaut

        /* simulation */
        initialiserDonneesSimulation();
}

// Initialisation avec configuration de l'horloge locale
void genericSystem::initialiser( float it, float deltatValue){

        /* horloge locale */
        time.initialiser( it, deltatValue );

        /* simulation */
        initialiserDonneesSimulation();
}
// Initialisation par défaut des données de simulation
void genericSystem::initialiserDonneesSimulation(void){

        tDebutDeLaSimulation = time.getInstantInitial();
        pasDeLaSimulation = time.getPasDeTemps() * PAS_SIM_SUR_PAS_TEMPS_MIN;

        /* tMaxDeLaSimulation est mis ici à une valeur cohérente qui n'est
         * pas définitive (sera changée) : il est calculé à chaque
         * démarrage de simulation */
        tMaxDeLaSimulation = tDebutDeLaSimulation + 3.0 * pasDeLaSimulation;
}



/*
 *
 * Fait évoluer (incrémente) l'ensemble du système
 * du pas de temps horloge "deltat"
 *
 */
void genericSystem::incrementer(){
        time.incrementerHorloge();
        actualiserIncrement();
}
```

```
/*
 *
 * Les méthodes de simulation
 *
 */

/* 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 simulation de bas
e "simuler" privée).
 */

/*
 * La methode de simulation "de base".
 *
 * Remarque : toutes les autres methodes de simulation ("simuler" avec
 * divers parametres) appellent cette methode "simuler".
 *
 * Description du déroulement :
 *
 * La valeur de l'instant de début de simulation
 * (= valeur de l'instant courant de l'horloge en entree de la fonction)
 * est enregistree dans "tDebutDeLaSimulation".
 * La valeur de l'instant de fin de simulation
 * (= "tDebutDeLaSimulation" + "dureeDeLaSimulation")
 * est enregistree dans "tMaxDeLaSimulation".
 *
 * Boucle d'appel du modele (ie de la fonction "incrementer") :
 * - pour t (instant courant de l'horloge) allant de "tDebutDeLaSimulation"
 *   à "tMaxDeLaSimulation",
 * - avec, a chaque pas de la boucle, increment de t de la valeur deltat (de
 *   l'horloge).
 *
 * Les resultats de la simulation (ie l'etat du modele) sont affiches avec
 * comme periode "pasDeLaSimulation", entre les instants de debut et de fin
 * de simulation ("tDebutDeLaSimulation" et "tMaxDeLaSimulation").
 *
 */
int genericSystem::simuler(void){

        int cr = CR_OK;

        /* enregistrement/calcul
         * de tDebutDeLaSimulation et tMaxDeLaSimulation */
        tDebutDeLaSimulation = time.getInstantInitial();
        tMaxDeLaSimulation = tDebutDeLaSimulation + dureeDeLaSimulation;
```

```
	/*
	 * vérifications
	 */
	if ( tDebutDeLaSimulation > tMaxDeLaSimulation ) return CR_NOK_BornesIncoherentes;
	if ( pasDeLaSimulation <= 0.0 ) return CR_NOK_PasSimulNegatif;

	/*
	 * Dans la durée "pasDeLaSimulation", il faut que le "pas de temps de
	 * base" (deltat de l'horloge) contienne au moins
	 * PAS_SIM_SUR_PAS_TEMPS_MIN fois. Si ce n'est pas le cas, alors
	 * pasDeLaSimulation est ici augmenté pour que ça le devienne.
	 */
	int pasDeLaSimulationRetouche = FAUX; // indicateur pour information à l'écran
	float p = (float)PAS_SIM_SUR_PAS_TEMPS_MIN * time.getPasDeTemps();
	if ( pasDeLaSimulation < p ){
		pasDeLaSimulation = p;
		pasDeLaSimulationRetouche = VRAI;
	}

	// trace
	cout << " ***** DEBUT DE LA SIMULATION EFFECTUEE :" << endl << endl;
	cout << " ***** les conditions de la simulation effectuee : " << endl;
	cout << "la simulation sort des résultats pour des itérations ";
	cout << "écartées de pasDeLaSimulation=" << pasDeLaSimulation << " s, ";
	cout << "sur une durée allant de l'instant tDebutDeLaSimulation=" << tDebutDeLaSimulation << " s, ";
	cout << "jusqu'à l'instant tMaxDeLaSimulation=" << tMaxDeLaSimulation << " s." << endl;
	if ( pasDeLaSimulationRetouche == VRAI ){
	cout << "Remarque : pour des questions de précision des calculs, ";
	cout << "pasDeLaSimulation a été modifié (augmenté) par rapport à la demande, ";
	cout << "de telle sorte que dans la durée pasDeLaSimulation, il contienne ";
	cout << "au moins PAS_SIM_SUR_PAS_TEMPS_MIN (=" << PAS_SIM_SUR_PAS_TEMPS_MIN << ") fois ";
	cout << "le pas de temps (plus petit incrément) deltat (=" << time.getPasDeTemps() << " s)." << endl;
	}
	cout << endl;

	time.afficherConfigurationHorloge();

	afficherEntetesEntites();

	/*
	 * boucle d'exécution de la simulation
	 *
	 * Demarrage de la simulation a "t", l'instant courant de l'horloge.
	 * Arret de la simulation a "tMaxDeLaSimulation".
	 *
	 * Debut des affichages des resultats de simulation a "tDebutDeLaSimulation".
	 * Fin des affichages des resultats de simulation a "tMaxDeLaSimulation".
	 */
```

```
        float prochainInstantAffichage = tDebutDeLaSimulation;

        while ( time.getInstantCourant() <= tMaxDeLaSimulation ){

                incrementer();

                /* Sorties des résultats "tous les" pas de simulation */
                if ( time.getInstantCourant() >= prochainInstantAffichage ){
                // on vient juste de passer un pas
                        prochainInstantAffichage = prochainInstantAffichage + pasDeLaSimulation;
                        afficherEntites();
                }
        }
        cout << " ***** FIN DE LA SIMULATION." << endl;

        return cr;
}

/*
 * Une methode de simulation.
 *
 * 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 genericSystem::simuler(int nbIterations, float pasSimulation){

        // trace
        cout << " ***** COMMANDE DE LA SIMULATION :" << endl << endl;
        cout << " ***** les conditions de simulation demandées : " << endl;
        cout << "il est demandé que la simulation sorte des résultats pour nbIterations="
                << nbIterations << " itérations successives, ";
        cout << "écartées de pasSimulation=" << pasSimulation << " s, ";
        cout << "à partir de l'instant courant." << endl << endl;

        pasDeLaSimulation = pasSimulation;
        dureeDeLaSimulation = nbIterations * pasSimulation;
        /* tDebutDeLaSimulation et tMaxDeLaSimulation
         * seront definis dans "simuler" appele plus bas */

        int cr = simuler();
        if ( cr == CR_OK ){
                cout << "Fin de fonction simuler sans erreur." << endl;
        } else {
```

```
                cout << "Fin de fonction simuler en erreur, code erreur : " << cr << "." << endl;
        }
}

/*
 * Une methode de simulation.
 *
 * 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 genericSystem::simuler(float tDebut, float dureeSimulation, float pasSimulation ){

        cout << " ***** COMMANDE DE LA SIMULATION :" << endl << endl;
        cout << " ***** les conditions de simulation demandées : " << endl;
        cout << "il est demandé que la simulation sorte des résultats pour des itérations ";
        cout << "écartées de pasSimulation=" << pasSimulation << " s, ";
        cout << "sur une durée dureeSimulation=" << dureeSimulation << " s, ";
        cout << "à partir de l'instant tDebut=" << tDebut << " s ";
        cout << "(auquel l'horloge est recalee avant de commencer la simulation)." << endl << endl;

        // on recale l'horloge locale ("tInitial" et "t") à tDebut
        time.setInstantInitial( tDebut );
        time.retourAinstantInitial();

        pasDeLaSimulation = pasSimulation;
        dureeDeLaSimulation = dureeSimulation;
        /* tDebutDeLaSimulation et tMaxDeLaSimulation
         * seront definis dans "simuler" appele plus bas */

        int cr = simuler();
        if ( cr == CR_OK ){
                cout << "Fin de fonction simuler sans erreur." << endl;
        } else {
                cout << "Fin de fonction simuler en erreur, code erreur : " << cr << "." << endl;
        }
}

/*****************************************************************************
 *
 * le corps de la classe listeEntitesAvecDerivee
```

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

/* Construction/initialisation de la liste privée des entités avec dérivée */
listeEntitesAvecDerivee::listeEntitesAvecDerivee( void ){
        nb = 0;
        for ( int i=0; i<NB_MAX_EntitesAvecDerivee; i++){ liste[i] = NULL; }
        aucuneInitialisation = VRAI;
}

/*
 * Entrer un element (pointeur) dans "liste", la liste privée des entités
 * avec dérivée. L'element n'est ajoute a la liste que s'il s'agit d'un
 * pointeur d'entité avec dérivée (si c'est un pointeur d'entité instantanée,
 * il n'est pas ajouté à la liste).
 */
int listeEntitesAvecDerivee::PL( pointeurDentityAvecDerivee ptr ){

        int cr = CR_OK;

        if ( nb >= NB_MAX_EntitesAvecDerivee){
                // impossible d'ajouter l'entite à la liste qui est pleine
                cr = CR_NOK_DepassementDim;
        } else {
                liste[ nb ] = ptr; nb++;
                aucuneInitialisation = FAUX;
        }
        return cr;
}
int listeEntitesAvecDerivee::PL( pointeurDentityInstantanee ptr ){

        /* une entité instantanée n'est pas saisie */

        aucuneInitialisation = FAUX;
        return CR_OK;
};

/*
 * Actualisation de la liste privée des entités avec dérivée :
 * Boucle de calcul des valeurs en fonction des dérivées, pour tous les
 * elements de la liste "liste".
 */
void listeEntitesAvecDerivee::AL( float deltat ){

        entityAvecDerivee *ptr;

        // Mise à jour des valeurs "value" en fonction des dérivées "ddt"
        for ( int i=0; i<nb; i++){
```

```
                ptr = liste[ i ];
                ptr->actualiser( deltat );
        }
}


/*****************************************************************************
 *
 * le corps de la classe compteRenduInitialisation
 *
 *****************************************************************************/

compteRenduInitialisation::compteRenduInitialisation( void ){
        putValue( CR_PAS_DE_PB_RELEVE );
}

void compteRenduInitialisation::putValue( int value ){ crInitValue = value; }

int compteRenduInitialisation::getValue( void ){ return crInitValue; }

int compteRenduInitialisation::pasDePbInitReleve( void ){
        if ( getValue()==CR_PAS_DE_PB_RELEVE ){
                return VRAI;
        } else {
                return FAUX;
        }
}
```