Analyse, conception, développement et exploitation d'un simulateur dans l'environnement de développement DIESE

PackMan : Remplissage de briques de jus de pomme et conditionnement en packs

Roger Martin-Clouaire, Jean-Pierre Rellier
INRA - Centre de Recherche de Toulouse-Auzeville
Unité de Biométrie et Intelligence Artificielle

Décembre 2007

Résumé

Ce document est une sorte de support de cours qui complète la formation théorique et pratique à l'environnement de modélisation/simulation DIESE.
On décrit d'abord un système réel (ou supposé tel) dont on souhaite étudier le fonctionnement par une approche de simulation informatique. La description du système est informelle, à la manière dont l'énoncerait celui qui l'a conçu ou installé, ou celui chargé de le piloter.
On analyse ensuite cette description informelle à l'aide des concepts que fournit une ontologie des systèmes pilotés, et avec une orientation informatique « objet ».
Les auteurs ont par ailleurs développé un modèle, et développé le simulateur correspondant, à titre de solution de référence au problème de la modélisation de notre système. Le corps du document consiste à éclairer les choix faits lors de ce travail.
Une dernière partie donne des exemples d'utilisation du simulateur développé, pour répondre à des questions typiques en expérimentation virtuelle.
Le lecteur pourra donc s'exercer à développer lui-même sa version du simulateur, guidé par la compréhension des choix de référence, ou bien utiliser ce document comme une référence lors du développement de simulateurs dans d'autres domaines d'application.
Le code complet du simulateur est fourni en annexe, ainsi qu'un jeu complet de ses fichiers d'entrée.

Ce document se réfère à la version 1.1 de PackMan, développée sous la version 4.4 de DIESE.

Plan

Annexes : spécifications structurelles et fonctionnelles complètes. Un jeu complet de fichiers arguments.

1) Introduction

La lecture de ce document accompagne le développement de la base de connaissances sur le système qui est pris comme terrain d'apprentissage de la modélisation d'un domaine, du développement d'un simulateur et de la pratique de simulation dans l'environnement DIESE[1] : un atelier de remplissage de briques de jus de fruit.

On souhaite préparer le lecteur à jouer les deux rôles suivants :

- celui du développeur d'un simulateur, qui utilise l'interface Solfege pour identifier, définir et coder les classes d'objets qui modélisent la structure et le fonctionnement du système ;
- celui de l'utilisateur du simulateur ainsi développé, qui utilise l'interface MI_Diese pour simuler sur machine la dynamique du système dans diverses hypothèses sur sa propre constitution et sur son environnement.

Ce document n'est pas une approche complète ni systématique des concepts, méthodes et techniques de modélisation sous DIESE, ni une description étape par étape ce que devrait faire le lecteur devant son écran pour construire le simulateur. Sa fonction est seulement d'éclairer les choix faits dans le développement du simulateur qui est livré[2] avec le présent document. Ce simulateur livré doit être considéré comme une solution de référence au problème de la modélisation de notre atelier de remplissage. La compréhension des choix qui le fondent, et leur discussion, peut amener le lecteur à développer une solution différente.

Puisque le lecteur n'est pas guidé pas à pas dans son travail, il doit donc avoir un bon aperçu du contenu de l'ontologie des systèmes de production[3] qui fonde l'outil DIESE, et un début d'expérience pratique du développement d'un simulateur dans cet environnement.

Cet aperçu peut s'acquérir par la lecture personnelle de l'ontologie et l'auto-apprentissage des l'interfaces Solfege et MI_Diese incluse dans DIESE, ou bien lors d'un stage de formation théorique et pratique dont notre atelier est le domaine support. Dans ce dernier cas, le présent document est un support de cours dans un premier temps, puis une référence lors du développement de simulateurs dans d'autres domaines d'application.

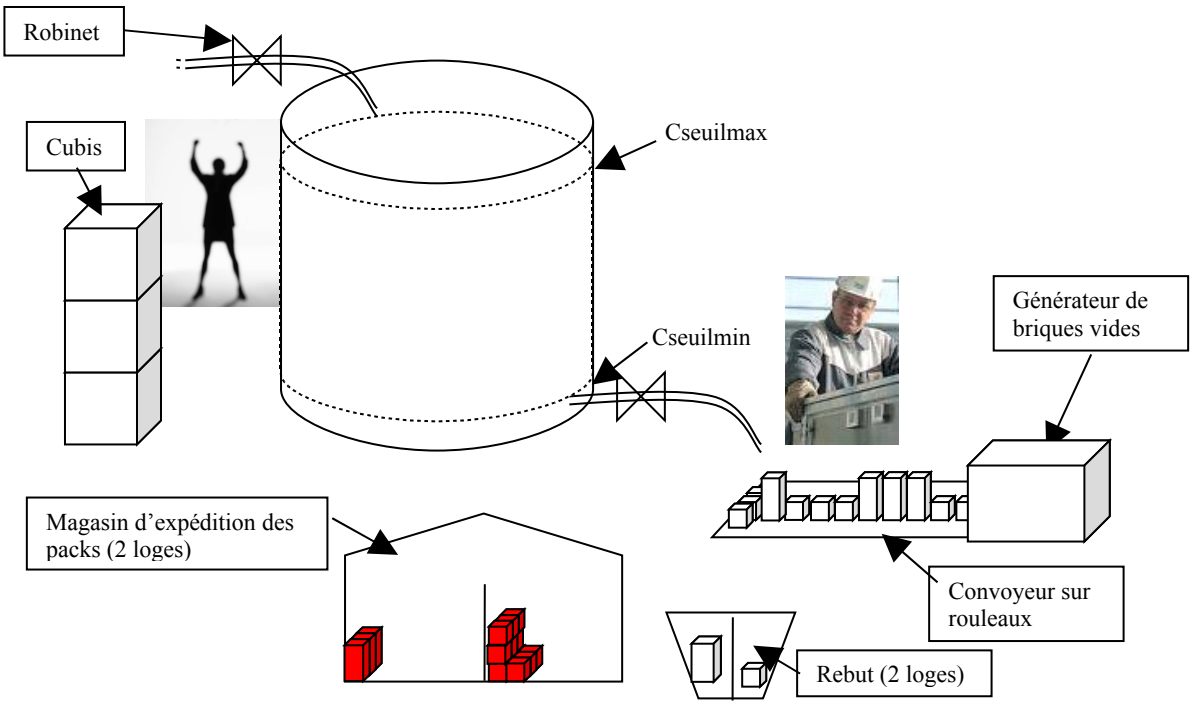
[1] La documentation de DIESE est fournie, en format HTML, avec l'environnement logiciel. Chaque composant de l'environnement possède sa propre documentation : les deux bibliothèques BASIC DIESE et CONTROL DIESE, les deux interfaces Solfege et MI_Diese.

[2] Le système hôte doit déjà disposer de l'environnement de développement, avec notamment l'interface Solfege. Le simulateur est livré avec un script d'installation d'une part, et d'autre part avec la dernière version compatible des librairies DIESE.

[3] Fondements ontologiques ds systèmes pilotés. R. Martin-Clouaire et J.-P. Rellier. Rapport interne UBIA. 86p. + annexes. 2006.

2) Présentation informelle du système

On considère un système de remplissage de briques de jus de fruit et de conditionnement en packs de 3.



Le jus de pomme provient d'une cuve de volume Cmax munie d'un robinet. Elle est alimentée de deux manières :

- automatiquement et périodiquement, une quantité aléatoire arrive en amont du robinet. Si celui-ci est ouvert et le reste, cette quantité passe dans la cuve (avec un débit fixé et constant) avant la prochaine arrivée. Si le robinet est fermé ou se ferme ou si le débit est insuffisant pour écouler tout ce qui arrive, la quantité qui ne passe pas dans la cuve est perdue.
- manuellement en versant un cubitainer de volume Vcubi.

Les briques ont deux tailles possibles de volume respectif Bpetite et Bgrande. Les briques visibles reposent sur un convoyeur à rouleaux qui permet d'aligner un certain nombre de briques (10 pour exemple). La tête de la file des 10 briques est positionnée sous le robinet de vidange de la cuve.

En régime de croisière, l'ouvrier chargé du remplissage procède de la manière suivante :

- constitution d'un lot de 3 briques identiques (regroupement sous le robinet)
- remplissage des 3 briques du lot, dans l'ordre d'ajout au lot
- fermeture des briques, conditionnement du lot sous forme de pack, transfert du pack dans le lieu de stockage.

Ce procédé n'est mis en œuvre que s'il peut être mené à son terme, c'est-à-dire jusqu'au stockage.

Constituer un lot de 3 briques se fait de la manière suivante : il faut qu'il existe 2 autres briques (parmi les 10 présentes sur la file) de même taille que la première. Il faut alors prendre la deuxième, la placer à côté de la première, puis faire de même avec la troisième. Chaque fois qu'un trou est créé, la file est automatiquement tassée et une nouvelle brique est insérée (en fin de file), de taille

aléatoire (petite avec la probabilité *p*, grande avec la probabilité *1-p*). La figure montre une configuration de la file en fin de constitution du lot (avant remplissage).
Voici un exemple de séquence d'états de la file dont l'état final est celui de la figure :
Etat initial = (p G p p p p p G G G)
Séquence d'états suivants =
(2p G . p p p p G G G) le point symbolise un espace vide où a été prélevée une brique pour la placer en tête de file
(2p G p p p p G G G p) par tassement, l'espace vide est supprimé et une brique nouvelle (petite ici) est insérée
(3p G . p p p G G G p)
(3p G p p p G G G p p) = état final

S'il n'existe pas sur la file 2 autres briques de même taille que la première, la première est mise au rebut, la file est automatiquement tassée et une nouvelle brique insérée.

Le remplissage d'une brique a pour effet de transférer la quantité Bpetite ou Bgrande de la cuve vers la brique, avec le débit fixé et constant du robinet de vidange.

La tâche de fermeture-conditionnement-transfert se ramène pour le besoin de cette étude à : la fermeture de chacune des briques, la création d'un pack avec le lot de 3 briques fermées, son placement dans le compartiment approprié du magasin d'expédition (avec tassement et insertion).

Si le volume de jus de fruit dans la cuve passe au-dessous du seuil Cseuilmin, un ouvrier (autre que celui près du convoyeur) alimente la cuve en y versant un cubitainer.

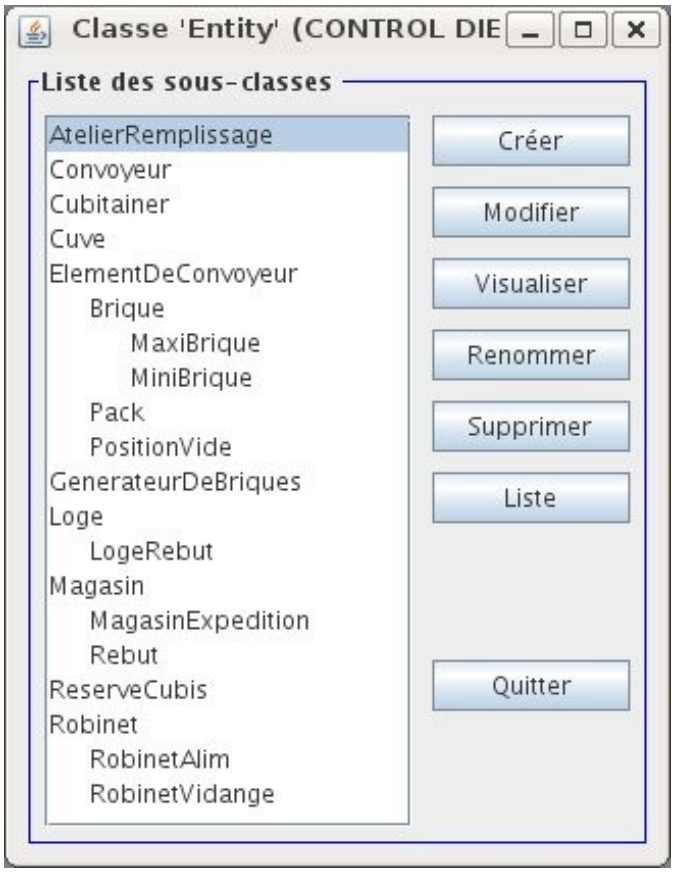
Si le volume de jus de fruit dans la cuve passe au-dessus du seuil Cseuilmax, un dispositif automatique ferme le robinet d'alimentation automatique et si le volume passe en dessous du seuil (Cseuilmax + Cseuilmin)/2 le dispositif le rouvre.

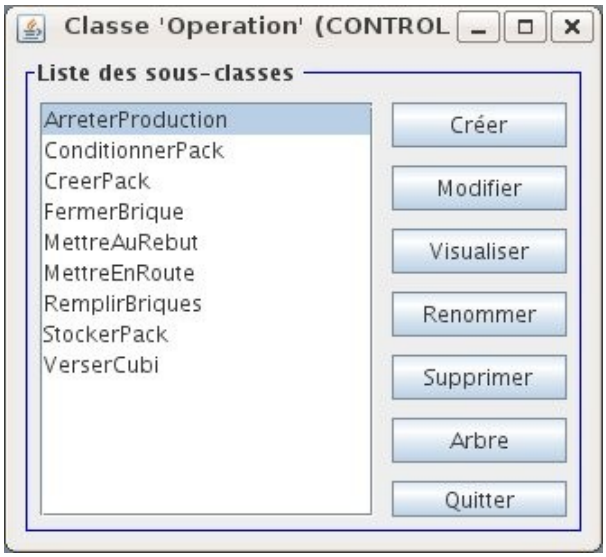
Si le nombre de petites (resp. grandes) briques dans le rebut est supérieur à 10 (ou un seuil quelconque) et est supérieur à 2 fois le nombre de grandes (resp. petites) alors corriger le générateur aléatoire de briques vides pour qu'il inverse la tendance.

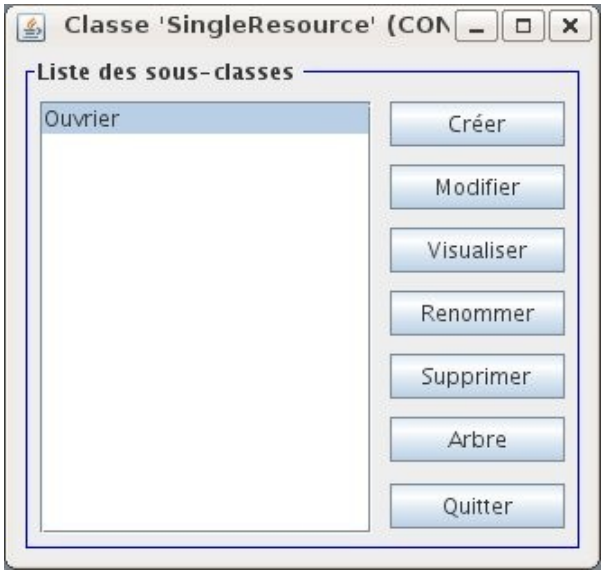
La mise en route (resp. l'arrêt) de la production consiste à ouvrir (resp. fermer) le robinet d'alimentation automatique, et à placer une brique sur chaque position encore inoccupée du convoyeur. L'ouvrier, et lui seul, ne peut ouvrir le robinet que si le niveau dans la cuve est inférieur au seuil maxi.

A la première mise en route, le niveau de la cuve est à son seuil maximum. Le nombre de cubitainers est un paramètre du système. Le magasin d'expédition et le rebut sont vides.

La production de packs s'opère sur une période entre deux dates. Au cours de cette période, la production peut être arrêtée si l'ouvrier devient indisponible (puis éventuellement reprise s'il redevient disponible), si le magasin d'expédition est plein, s'il n'y a plus de cubis.


Classe 'Entity' (CONTROL DIE
Liste des sous-classes
AtelierRemplissage
Convoyeur
Cubitainer
Cuve
ElementDeConvoyeur
Brique
MaxiBrique
MiniBrique
Pack
PositionVide
GenerateurDeBriques
Loge
LogeRebut
Magasin
MagasinExpedition
Rebut
ReserveCubis
Robinet
RobinetAlim
RobinetVidange
Créer
Modifier
Visualiser
Renommer
Supprimer
Liste
Quitter


Classe 'Operation' (CONTROL
Liste des sous-classes
ArreterProduction
ConditionnerPack
CreerPack
FermerBrique
MettreAuRebut
MettreEnRoute
RemplirBriques
StockerPack
VerserCubi
Créer
Modifier
Visualiser
Renommer
Supprimer
Arbre
Quitter


Classe 'SingleResource' (CON
Liste des sous-classes
Ouvrier
Créer
Modifier
Visualiser
Renommer
Supprimer
Arbre
Quitter

3) Analyse et éléments de conception

3.1) Structure du système

Système technique

Atelier = {cuve, stock de cubitainers, convoyeur, lieu de stockage des packs, rebut}
Cuve = {robinet d'alimentation, robinet de vidange}
Robinet : robinet d'alimentation ou de vidange
Stock de cubitainers = ensemble de cubitainers
Cubitainer
Convoyeur = {ensemble ordonné d'éléments de convoyeur, générateur de briques}
Deux conceptions envisageables :
- (1) deux composants : une entité ensembliste (packs, briques) et le générateur
- (2) deux descripteurs, l'un de type « ensemble d'entités », l'autre pointant sur le générateur (c'est la conception adoptée)

Elément de convoyeur : pack ou brique ou position vide
Selon les deux conceptions pour le convoyeur :
- (1) éléments de l'entité ensembliste qui est le premier composant du convoyeur
- (2) éléments du tableau d'entités qui est la valeur du descripteur modélisant l'ensemble

Pack = ensemble de briques
Brique : petite brique ou grande brique
Petite brique
Grande brique
Position vide : entité virtuelle placée à chaque position du convoyeur occupée ni par un pack ni par une brique
Générateur de briques
Lieu de stockage : lieu de stockage des packs ou rebut = ensemble de loges
Lieu de stockage des packs
Rebut
Loge = ensemble d'éléments de convoyeur (pack ou brique, en pratique)
Loge de rebut ⇨ loge, munie d'une alarme sur le nombre d'éléments

Système opérant (ressources et opérations)

Ouvrier ⇨ SingleWorker
Immobilisation/mobilisation d'un ouvrier
Mise en route de l'atelier
Arrêt de l'atelier
Assemblage des briques d'un pack
Remplissage d'une brique
Fermeture des briques d'un pack
Conditionnement d'un pack
Stockage d'un pack
Mise au rebut d'une brique
Alimentation manuelle de la cuve

Les ressources (ici l'ouvrier) sont structurées en « pools », et le système opérant est un ensemble de pools disjoints. Notre système opérant est donc un ensemble d'un seul pool d'une seule ressource.

```
// ********************************************************************
// la strategie (vide) du manager
// ********************************************************************

+ I activityIteration iterationPlagesTravail;

+ I activitiesResourcesBlock arb1
   <- C <I><, iterationPlagesTravail>;
   <- C <I><, lePoolOuvrier>;
;
+ I activitiesResourcesBlockSet arbSet
   <- E <I><,arb1>;
;
+ I strategy laStrategie
   <- C <I><,arbSet>;
;
```

```
// ********************************************************************
// le systeme opérant
// ********************************************************************

+ I ouvrierOperant,;

+ I ouvrier lOperateurConvoyeur
   availabilityStatus = 0;
   power = 0.25;
;

+ I simpleResourcePool lePoolOuvrier,
   <- E <I><, lOperateurConvoyeur>;
;

+ I resourcePoolDisjunction lesPools,
   <- E <I><, lePoolOuvrier>;
;

+ I operatingSystem pOS,
   <- C <I><, lesPools>;
;
```

```
// ********************************************************************************
// spécification du système de production
// ********************************************************************************

+ I controlledSystem cs
   attachedEntity = <I><,latelier>;
;

+ I manager leManager
   <- C <I><, laStrategie>;
;

+ I productionSystem ps,
   <- C <I><controlledSystem,>;
   <- C <I><operatingSystem,>;
   <- C <I><manager,>;
;

ENTITE_SIMULEE <I><,ps>;
```

Système décisionnel

```
Plan = iterate(                          // itération sur les plages de travail
        before(                          // une plage de travail = succession de 3 phases
            Mise en route,
            iterate(                     // la production d'une série de packs
                and(
                    or(                  // pack OK ou bien brique au rebut …
                        before(          // gamme de fabrication
                            Assemblage du pack,
                            Remplissage des briques,
                            Fermeture des briques,
                            Conditionnement du pack,
                            Stockage du pack),      // fin gamme
                        Mise au rebut),
                    optional(Alimentation manuelle),   // … et alim de temps en temps
                )
            )                            // fin production d'une série de packs
            Arrêt)                       // fin plage de travail
        )                                // fin plan
```

légende : *activité* : opérateur prédéfini, ou spécialisation (propre au domaine) d'un opérateur prédéfini
Activité : activité primitive spécifique au domaine

Le Plan est un composant d'un « bloc Activités/Ressources » (A/R).
Il y joue le rôle de l' « activité ».
Le rôle « ressource » est joué par le second composant du bloc : l'ensemble de ressources (le « pool ») dédié à la réalisation des activités du bloc et seulement de ce bloc.

Il n'y a qu'un seul bloc « A/R » identifié dans l'application.

Cet ensemble d'un seul bloc « A/R » est un des composants de la stratégie de pilotage. Les autres composants (trajectoire réactive et ensemble de règles de préférences) ne sont pas étudiés dans cette application.

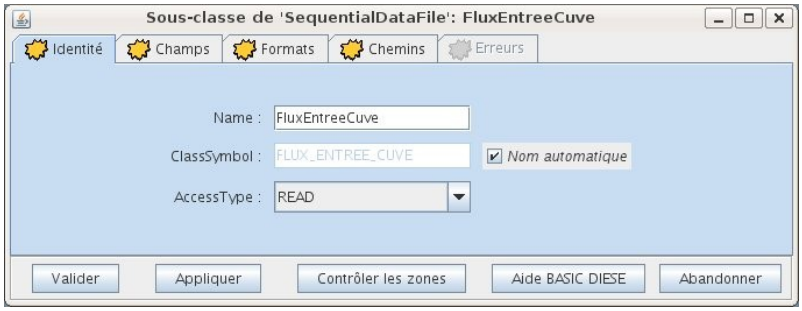
Cette stratégie est l'unique composant du « manager » du système.

Le manager est un des 3 composants du système de production, avec le système technique (l'atelier de remplissage) et le système opérant (structurellement : les ressources, et fonctionnellement : les opérations).

Le système de production est l'« entité simulée ». Elle joue le rôle de point d'encrage du « moteur de simulation » (encapsulé dans l'instance de la classe Simulation créée en préambule de la simulation) sur la « base de connaissances » structurelles, fonctionnelles et dynamiques sur le système étudié.

```
+ P fluxEntreeCuveLectureProcess lectureFluxEntree
   PAS = 5;  // 5 minutes
   NOM_SIMPLE_FICHIER "flux1.txt";
   INIT_PRCD_EVT    DATE_OCCUR = 0;
                    PRIORITE = 0;
   ;
;
```

```
// serie des quantites disponibles en entree de la cuve
//-----------------------------------------------------
0  0   10.
0  5   7.
0  10  8.
0  15  8.
...
23 40  8.
23 45  8.
23 50  10.
23 55  7.
```



3.2) Fonctionnement

3.2.1) Fonctions dans le système technique

Arrivée périodique de jus de fruit en amont du robinet d'alimentation.

Deux conceptions envisageables :

(1) les quantités sont générées dynamiquement

En chaque instant où une arrivée est supposée se produire dans la réalité, on lance le calcul de la quantité selon une loi de probabilité. La quantité lue ou générée devient la valeur courante d'un descripteur de la cuve, valeur surchargée lors de la prochaine lecture ou la prochaine génération.

1-1) Le calcul de la quantité est fait par la procédure d'exécution d'un processus ponctuel, lui-même géré par un événement autogénérable (à intervalle constant, selon la description de l'atelier).

1-2) Le calcul de la quantité est fait dans la procédure d'initialisation du processus continu d'alimentation de la cuve (voir ci-dessous). Il n'y a alors pas d'événement autogénérable d'arrivée de jus de fruit.

(2) les données sont des lignes d'un fichier de données (c'est la conception mise en oeuvre)

On lit une nouvelle ligne du fichier en l'instant où l'arrivée de jus de fruit est supposée se produire dans la réalité. Le rythme est fixé dans le fichier argument .str, lors de la spécification du processus en jeu : DATE_OCCUR = *0* et PAS = *5*. Ainsi, les arrivées sont en t=*0*, puis en t=*5*, puis en t=*15*, etc.

Cette conception mène à créer une classe de fichier et une classe de processus de lecture.

Le fichier, spécialisation d'une classe prédéfinie dans BASIC DIESE : SequentialDataFile, est spécifié (dans Solfege) par :

- ses champs : heure, minute et quantité
- le type de leurs valeurs : deux entiers et un flottant
- le format de lecture d'un enregistrement : « %d%d%f »
- le nombre de lignes d'entête : 2
- son emplacement physique : un répertoire, qui pourrait aussi être spécifié dans le processus de lecture. Le symbole IN_DIR renvoie à un chemin désigné dans le fichier argument .sim. Le nom du fichier (dans ce répertoire) est précisé dans le fichier .str.

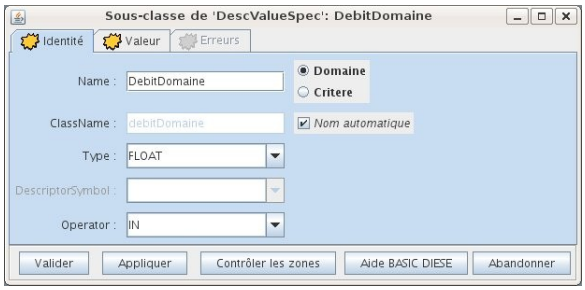
Solfege prend entièrement en charge le codage C++ de la classe de fichier.

```
void fluxEntreeCuveLectureProcess_initializeProcess_Body(ProcessMethod* pM)
{
  <type> pCP = (<type>)(pM->DescribedContinuousProcess());
  SequentialDataFile* pF = pCP->GetFile();
  //...
  int n;
  time_t timeDebFile = DateToAbsoluteTime(1, JAN, 2007, 0, 0, 0);
  time_t timeDebSim = pCurrentSim->AbsoluteBeginningTime();
  time_t timeDiff = timeDebSim - timeDebFile;
  int clockDiff = TimeIntervalToClockInterval(timeDiff);
  int n0 = clockDiff / pCP->Step();
  n = n0 + pF->GetNbTopCommentLines();
  pF->Skip(n);
};
```

```
void fluxEntreeCuveLectureProcess_postReadProcess_Body(ProcessMethod* pM)
{
  <type> pCP = (<type>)(pM->DescribedContinuousProcess());
  Cuve* pCuve = (Cuve*)pCP->ProcessedEntity();
  SequentialDataFile* pF = pCP->GetFile();
  float q      = pF->GetFloatFieldValue(QUANTITE);
  // ...
  pCuve->SetFloatConstValue(ENTREE_DISPONIBLE, q);
};
```

```
void fluxEntreeCuveLectureProcess_stopProcess_Body(ProcessMethod* pM)
{
  <type> pCP = (<type>)(pM->DescribedContinuousProcess());
  SequentialDataFile* pF = pCP->GetFile();
  StopAlimAuto();
  pF->Close();
  pCP->ToBeStopped(TRUE);
};
```

```
void alimentationAutomatiqueProcess_goOneStepForwardProcess_Body(ProcessMethod* pM)
{
  ContinuousProcess* pCP = pM->DescribedContinuousProcess();
  Cuve* pCuve = (Cuve*)pCP->ProcessedEntity();
  // ...
  Robinet* pRobinet = (Robinet*)pCuve->GetComponent(ROBINET_ALIM);
  float debitAlim = pRobinet->GetFloatConstValue(DEBIT);

  float transfertMaxEnUnPas = debitAlim * pCP->Pas();
  // ...
};
```



La lecture est opérée par une spécialisation d'un processus prédéfini dans BASIC DIESE : la lecture d'un fichier de données séquentielles. Le processus est géré par un événement, instance directe d'Event, créé par le parseur du fichier .str, lors de la spécification INIT_PROCEED_EVENT.
On surcharge les méthodes d'initialisation, de poursuite et d'arrêt du processus de lecture. Le code de l'initialisation est généré par Solfege pour sa partie 'ouverture du fichier'. La surcharge consiste à sauter un nombre de lignes tel qu'on se positionne sur la ligne du fichier correspondant à la date de début de la simulation (spécifiée dans le fichier .sim). Sans ce calcul spécifique, le positionnement n'aurait sauté qu'un nombre de lignes égal à la spécification 'SetNbTopCommentLines'.
Précisément, ce n'est pas la procédure de poursuite qu'on surcharge (elle est prédéfinie dans BASIC DIESE) mais ce qu'il se passe juste après la lecture physique du fichier : la méthode 'PostRead'. Son corps consiste, de manière générale, à 'transférer' les valeurs lues vers des descripteurs des entités du système. En l'occurrence, le descripteur 'EntreeDisponible' de la cuve.
Ce qu'il faut faire lorsque le processus reçoit une directive STOP est l'objet de la surcharge de la méthode d'arrêt (c'est notamment le cas quand la procédure de lecture rencontre la marque de fin de fichier). Ici, on arrête l'alimentation automatique. Le reste du code est généré par Solfege.

Processus automatique d'alimentation de la cuve.

On considère que le processus est continu. En d'autres termes, le transfert du jus en amont du robinet est progressif est peut ainsi être interrompu, laissant une certaine quantité non transférée.
Deux conceptions sont envisageables :

(1) un seul processus couvre temporellement toutes les arrivées en amont du robinet
Il est initié à la première et arrêté après le dernier transfert dans la cuve.
La procédure de poursuite teste si le robinet est ouvert, et effectue le transfert (voir plus loin) seulement si c'est le cas (il n'y a donc pas lieu d'expliciter une pré-condition), et si la valeur de 'EntreeDisponible' de la cuve n'est pas nulle.

(2) un processus est initié à chaque arrivée en amont du robinet (conception mise en oeuvre)
Il est arrêté dès que tout le jus arrivé a été transféré dans la cuve, ou bien si le robinet a été fermé. Il y a une pré-condition à l'initiation : le robinet doit être ouvert. S'il ne l'est pas, l'initiation du processus avorté est reportée à « l'étape » suivante.

Ce qui est commun aux deux conceptions :
- Le pas du processus est à ajuster (dans la fenêtre de création du processus, onglet 'Descripteurs') au niveau général de précision de la simulation. Les arrivées de jus de fruit sont espacées de 5 minutes : on suppose que tout le jus peut être transféré vers la cuve dans cet intervalle.
- La procédure de poursuite transfère une quantité de jus fonction du débit et du pas choisi. Si le pas est 1 minute et de débit 10 l/min, alors, la procédure transfère 10 l au plus. Si la quantité disponible est 5 l, l'alimentation est réalisée en un pas ; si la quantité disponible est 15 l, l'alimentation est réalisée en deux pas (10 l lors du premier pas, 5 l au second).
- l'ouverture/fermeture du robinet est provoquée (a) dans les opérations de mise en route et d'arrêt (voir plus loin), et (b) par un moniteur sur le niveau de la cuve (voir plus loin).
- Le débit est un descripteur constant d'un robinet. On l'a doté d'une spécification de domaine de valeurs (DebitDomaine) qui limite ses valeurs admissibles. Même s'il n'en a pas la vocation, un descripteur constant peut changer de valeur en cours de simulation.
- Le niveau de la cuve est un descripteur variable (VolumeCourant). D'abord pour traduire que cette grandeur a vocation à changer au cours du temps ; ensuite parce qu'un moniteur ne peut être placé que sur un descripteur variable.

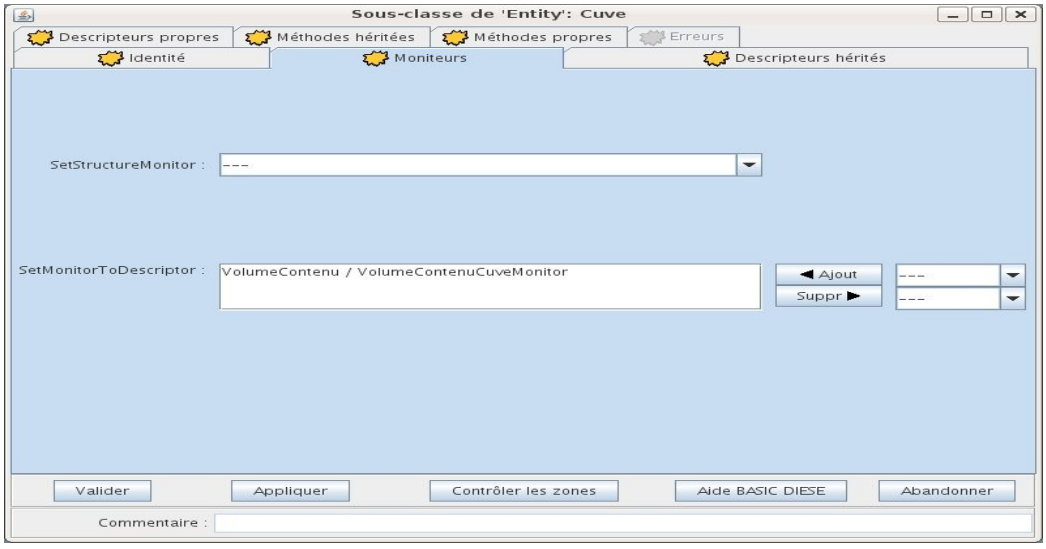
Dans la conception (2) :

– le moniteur en question réalise aussi l'arrêt du processus, s'il est en cours et qu'on monte au-dessus du seuil haut, et une nouvelle programmation de sa mise en oeuvre, si on descend sous le seuil médian et si l'agenda n'en contient pas déjà une.

```
{i1} = <pi><"CONVOYEUR"  "longueur">;  // valeur dans le .par

+ I atelierRemplissage latelier
   // ...
   + C convoyeur leConvoyeur
       // ...
       convoyeurPlateforme << <I><, uneMaxiBrique>;
       POUR i2 = (2) A ({i1})
           convoyeurPlateforme << I positionVide,; ;
       FIN POUR
   ;
   // ...
;
```

```
BasicEntity* alimentationConvoyeur_Body(EntityMethod* pM)
{
  Convoyeur* pE = (Convoyeur*)pM->DescribedEntity();
  Brique* pNouvelleBrique = NULL;
  <type> pGenBrq = (<type>)pE->GetEntityConstValue(CONVOYEUR_GENERATEUR);
  float probaPetite = pGenBrq->GetFloatConstValue(PROBA_PETITE_BRIQUE);
  int tirage = RandomInteger(100);
  if(((float)tirage)/100. < probaPetite)
    pNouvelleBrique = new MiniBrique();
  else
    pNouvelleBrique = new MaxiBrique();
  // ...
  return pNouvelleBrique;
};
```

```
void tassementConvoyeur_Body(EntityMethod* pM)
{
  int rang = pM->GetIntArgValue(RANG);
  Entity* pConvoyeur = pM->DescribedEntity();
  // ...
  <type>* descPlateforme = (<type>)(pConvoyeur->GetConstantDescriptor(CONVOYEUR_PLATEFORME));
  BasicEntity* pPV = descPlateforme->GetValueElement(rang-1);
  DeleteEntity(pPV);
  // ...
  PositionVide* pVide = new PositionVide();
  descPlateforme->SetValueElement(pVide, k-1);
};
```



- particularité des processus continus autogénérables, l'initialisation (re)met à 'false' la propriété du processus qui détermine s'il doit être arrêté (par le moteur) immédiatement. En effet, cette propriété a été mise à 'true' dans la procédure de poursuite dans le cas (général) où toute la quantité en entrée a été transférée, justement pour provoquer l'arrêt.

Initialisation du convoyeur.

Méthode attachée au convoyeur, invoquée lors de la mise en route de l'atelier (voir plus loin).
Le convoyeur a N positions, de rang *0* (sous le robinet) à *N-1* (près du générateur).
L'initialisation consiste à occuper toutes les positions par une brique. Cela met en jeu des alimentations (génération d'une brique) et des tassements (avancée vers le robinet).
Avant l'initialisation, un certain nombre de briques sont présentes du rang *0* au rang *k-1* et les positions à partir de *k* jusqu'à *N-1* sont vides. Autrement dit, si $k < 1$ il n'y a pas de briques, sinon il y en a *k*. L'initialisation réalise donc N-k alimentations. Dès qu'une alimentation s'est produite, les rouleaux du convoyeur décalent d'une position (tassement) la série de briques précédée par au moins une position vide. Exemples, pour N = 10 :
$k < 1$: 9 {alimentation + tassement} et 1 alimentation terminale
$k = 8$: 1 {alimentation + tassement} et 1 alimentation terminale
$k = 9$: 1 alimentation
$k = 10$: 0 alimentation

Alimentation du convoyeur (génération d'une brique de taille aléatoire).

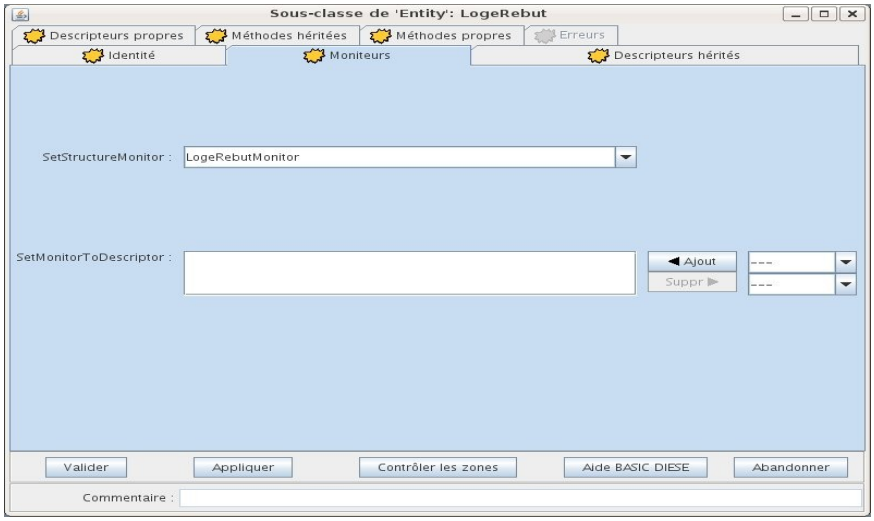
Méthode attachée au convoyeur, invoquée dans son initialisation et dans l'opération de constitution d'un lot de briques (voir plus loin le système opérant).
La brique générée est petite si le résultat d'un tirage aléatoire selon une loi uniforme entre 0 et 1 est inférieur à p. Grande sinon.
La brique est placée en dernière position (nécessairement vide) du convoyeur.
Si une des positions en remontant vers le robinet est vide, alors la série de briques partant de la nouvelle générée jusqu'à celle juste avant cette première position vide est décalée d'une position vers le robinet : c'est le tassement.

Tassement automatique de la file de briques.

Méthode attachée au convoyeur, dynamiquement associée à l'alimentation.
Elle possède un argument : le rang k à partir duquel les éléments sont déplacés d'une position vers le robinet. La position k-1 est nécessairement vide. On récupère la mémoire allouée à cet objet.
Les éléments de rang i à partir de k jusqu'à N-1 prennent le rang i-1.
L'élément de rang N-1 devient une position vide.

Atteinte de seuils sur le niveau de la cuve.

Un moniteur, mécanisme du moteur qui déclenche automatiquement l'exécution d'une procédure, est placé sur le descripteur modélisant le niveau variable
- seuil haut : la procédure du moniteur qui est lancée automatiquement avant le changement de valeur effectif réalise la fermeture du robinet d'alimentation (et, dans la conception (2) de l'alimentation, l'arrêt du processus). La valeur affectée est bornée par le volume de la cuve.
- seuil intermédiaire (alors que le niveau baisse) : le même moniteur (donc la même procédure) rouvre le robinet (et, dans la conception (2) de l'alimentation, la reprise du processus). La valeur affectée est bornée par zéro (vraisemblablement jamais en pratique).
- seuil bas : la condition d'ouverture de l'alimentation manuelle devient automatiquement satisfaite. La valeur affectée est bornée par zéro.




```
BasicEntity* logeRebut_whenAddElement_Body(MonitorMethod* pM)
{
  BasicEntity* loge = pM->GetEntityArgValue(CURRENTENTITYCONTAINER_ARGUMENT);
  DescriptedEntity* rebut = (DescriptedEntity*)loge->GetSuperSet();
  int lastProbaRaised = rebut->GetIntConstValue(DERNIERE_PROBA_AUGMENTEE);

  int nbPetites = rebut->GetElement(0)->GetNumberOfElements();
  // ...
  Entity* leConvoyeur = GetFirstEntity(CONVOYEUR);
  Entity* generateurDeBriques = leConvoyeur->GetEntityConstValue(CONVOYEUR_GENERATEUR);
  float probaPetite =
    generateurDeBriques->GetFloatConstValue(PROBA_PETITE_BRIQUE);
  // ...
  if((nbPetites >= gIntParam_REBUTPETITES_seuilMaxi_)
     && // ...
     (lastProbaRaised != 0)) {
    probaPetite /= gRealParam_REBUT_coeffReducProba_;
    if(probaPetite > 1.) probaPetite = 1.0;
    rebut->SetIntConstValue(DERNIERE_PROBA_AUGMENTEE, 0);
    // ...
  }
  else  // ...
};
```

```
bool iterationMagasinOuRebut_closingPredicate_Body(EntityMethod* pM)
{
  // ...
  Entity* pAtelier = GetFirstEntity(ATELIER_REMPLISSAGE);
  // ...
  Entity* pMagasin = pAtelier->GetComponent(MAGASIN_EXPEDITION);
  for(int k=0;k<2;k++) {
    Entity* pLoge = pMagasin->GetElement(k);
    int contenance = pLoge->GetIntConstValue(CONTENANCE);
    if(pLoge->GetNumberOfElements() == contenance) {
      return TRUE;
    }
  }
  // ...
  return FALSE;
};
```

Atteinte de seuils sur le contenu du rebut.

Moniteur sur la structure de chacune des deux loges du rebut.
La procédure modifie le descripteur du générateur de briques qui modélise (par une probabilité et son complément à 1) le rapport petites/grandes. L'élément candidat est ajouté au stock.
Comme la procédure est lancée (automatiquement) avant l'ajout effectif d'un élément, le changement est déclenché quand le nombre courant est, non pas supérieur, mais supérieur ou égal au seuil, situation qui résulte de l'ajout précédent. Bien entendu, on n'oublie pas de borner la probabilité à 1, quand on l'augmente.
Le codage du moniteur prend l'option de ne pas augmenter deux fois de suite la même probabilité (celle de générer une petite brique, celle de générer une grande brique).

Atteinte de seuils sur le contenu du stock.

On pourrait placer un moniteur sur la structure de chacune des deux loges du stock.
La procédure (lancée avant l'ajout effectif d'un élément) arrêterait la production si la loge est pleine. Le pack candidat ne serait pas ajouté au stock. L'arrêt de la production pourrait être provoqué en fermant l'itération (voir plus loin) et en rendant l'ouvrier indisponible (voir plus loin).
Beaucoup plus simplement, on conditionnera la fermeture du cycle de production au fait qu'une loge soit pleine.

```
// …
Event* pEvt = NULL;
pCurrentSim->GoHeadInAgenda();
while(pCurrentSim->IsCurrentInAgenda()) {
    pEvt = pCurrentSim->GetCurrentInAgenda();
    // ...
    pCurrentSim->GoNextInAgenda();
}
// …
```

```
// …
bool planned = pCurrentSim->InsertInAgendaEvents(pEvt);
if(! planned) {
    pEvt->DeleteProcesses();
    pEvt->DeleteInDepth();
}
// …
```

3.2.2) Commentaires sur les fonctions globales

1) Elles sont identifiées parce qu'elles sont invoquées à plusieurs endroits dans le code.

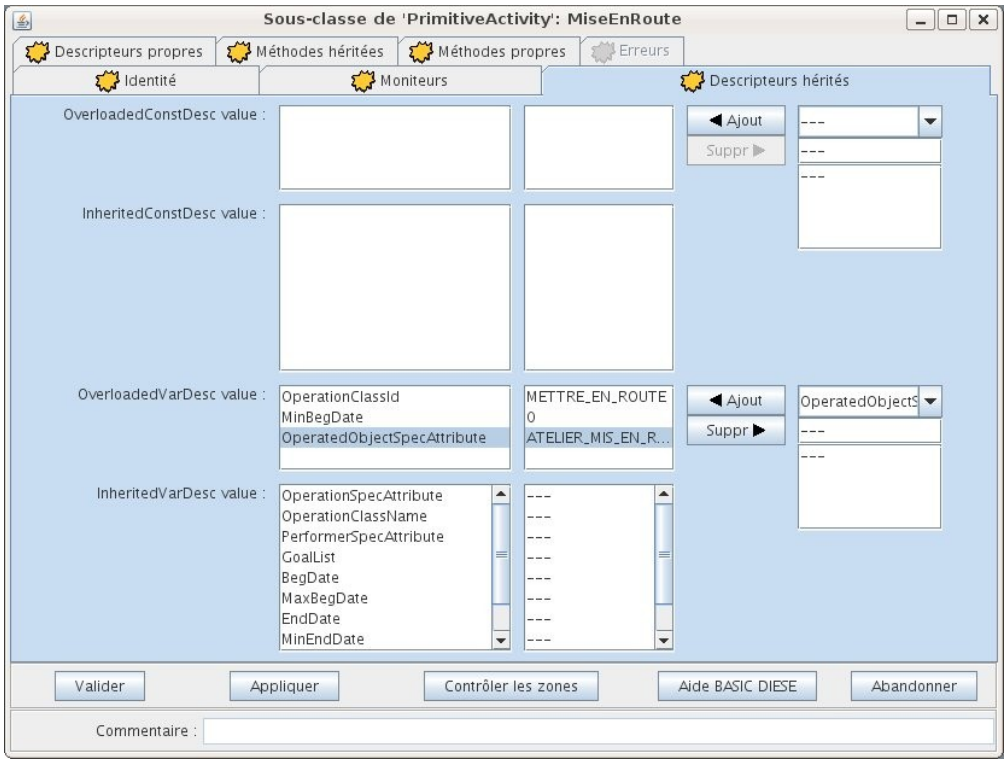
2) Alternativement à leur globalisation, elles pourraient aussi bien faire l'objet de méthodes attachées à des entités (en fait, la cuve). On peut en effet vérifier que le pointeur sur la cuve est connu dans les bouts de codes dans lesquels les fonctions sont invoquées.

3) Les lignes de code qui parcourent l'agenda peuvent être réutilisées telles quelles dans toutes les applications (voir ci-contre).

4) Les lignes de codes ci-contre répondent au souci de maîtriser les allocations/récupérations de mémoire, ici celles portant sur des événements candidats à entrer dans l'agenda, mais qu'on n'a finalement pas insérés. C'est par exemple le cas quand un événement candidat interviendrait après la date de fin de simulation, ou bien quand on a spécifié une probabilité d'occurrence inférieure à 1 (et que le tirage aléatoire est défavorable), ou encore quand un même événement porte sur un même objet à la même date.

On note au passage l'exploitation de la variable globale *pCurrentSim*, qui pointe sur l'unique instance allouée de la classe Simulation. On remarque ici qu'elle permet d'accéder à l'agenda d'événements, mais elle rend aussi et notamment le service Clock qui renvoie l'instant simulé courant.

```
// ********************************************************************
// le systeme technique
// ********************************************************************
+ I magasinExpedition leMagasin
   + E loge lesPetitsPacks
       contenance = 100;
   ;
   + E loge lesGrandsPacks
       contenance = 100;
   ;
;
+ I reserveCubis laReserveCubis
   POUR i3 = (1) A (10)
       + E cubitainer,;
   FIN POUR
;
```




```
//---------------------
// le plan
//---------------------
<I> <,iterationPlagesTravail>
   + E activityBefore sequencePlageTravail,
       <- E <I><, laMiseEnRoute>;
       <- E <I><, laProduction>;
       <- E <I><, lArretProduction>;
   ;
       readyToRun = 1;
;
```

```
//---------------------
// spécifications
//---------------------

+ SpE lesAteliers;

+ I operatedObjectSpecification lAtelierSpec,
   objectEntitySpecAttribute = <SpE><lesAteliers>;
   selectorFctId = 1; // ANYONE=1, MAX=2, ALL=3, DISJ=4, SETS=5
;

+ I arretProduction lArretProduction
   operatedObjectSpecAttribute = <I><, lAtelierSpec>;
;
```

3.2.3) Fonctions dans le système opérant

Mise en Route.

Faisabilité : l'ouvrier est (ou est redevenu) disponible, et a le temps de faire au moins un pack (voir le commentaire sur l'effet de la fonction « Arrêt » ci-dessous). Aucune loge du stock de packs n'est pleine. Le stock de cubitainers n'est pas vide. Cette condition est exprimée dans le prédicat d'ouverture de l'activité.
La disponibilité de l'ouvrier est régie par les événements d'arrivée/départ de l'ouvrier (*cf. infra*).
La configuration initiale de l'atelier (fichier argument .str) ne spécifie que la contenance des loges et n'ajoute pas d'éléments (packs ou briques) ; le constructeur d'une loge n'ajoute de lui-même aucun élément. Enfin, la configuration initiale de l'atelier installe un ensemble de 10 cubitainers.
Ainsi, au début d'une plage de travail, il n'y aura pas de (re)mise en route si le stock de cubis a été vidé lors de la plage précédente (la production de packs se sera alors arrêtée en cours de plage, *cf.* section <iterationMagasinOuRebut>).

Entité visée : un atelier. On l'a internalisé dans le constructeur, par une instanciation d'une spécification d'objet opéré (AtelierMisEnRoute).

Effet : initialisation du convoyeur (voir section « Dynamique du système technique ») ; ouverture du robinet d'alimentation (cuve), lancement de l'alimentation automatique.
Le convoyeur est initialisé en invoquant sa méthode dédiée. Ainsi, la mise en route n'est pas responsable du « comment c'est fait ».
Le robinet n'est effectivement ouvert que si ne niveau dans la cuve n'est pas trop haut.
Seulement si le robinet est effectivement ouvert, la fonction globale StartAlimAuto (voir plus haut un commentaire général) place dans l'agenda un événement qui initie et lance une alimentation automatique, synchronisée avec la prochaine arrivée de jus en amont de la cuve.

Type opération : vue comme ponctuelle (opération 'TimeGranulated' de vitesse = 100% en un pas).

Arrêt.

Faisabilité : pas de contrainte. Le fait que l'ouvrier va devenir indisponible, ou qu'une des deux loges du stock de packs est pleine, ou que le stock de cubitainers est vide, est capté par le prédicat de fermeture de l'itération de production de packs. Dès lors, l'arrêt, en tant qu'élément suivant l'itération dans la séquence, doit être opéré au prochain pas de temps.

Entité visée : un atelier. On l'a externalisé dans le fichier argument .str, par une instanciation de la classe prédéfinie OperatedObjectSpecification, qui retient 1 instance quelconque (mais il n'y en a qu'une !) de l'expansion de la spécification d'ensemble d'entités LesAteliers.

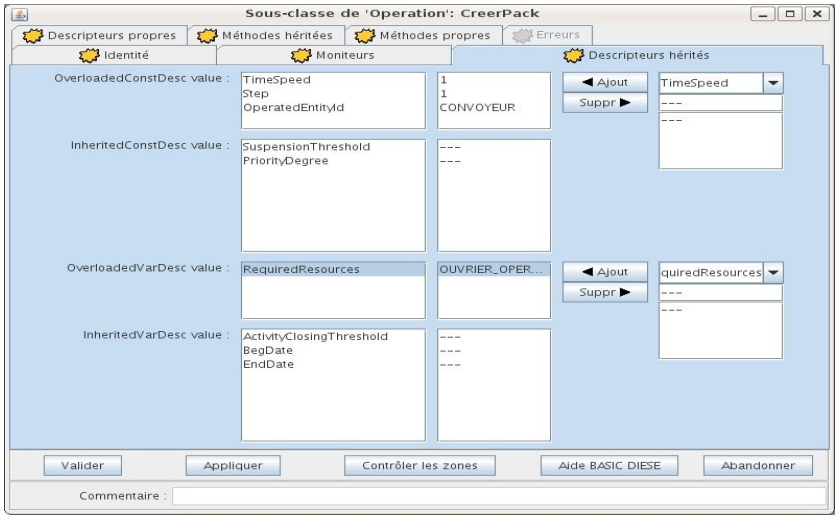
Effet : fermeture du robinet d'alimentation de la cuve, arrêt de l'alimentation automatique. Si l'arrêt intervient parce qu'on n'aurait plus le temps de terminer un pack si on le commençait, il faut empêcher une (re)mise en route immédiate : cela justifie la condition « l'ouvrier a le temps de faire au moins un pack » dans la faisabilité de la mise en route .
Attention ! Cette conception suppose que la gamme de fabrication d'un pack ne dure pas plus de x unités de temps (x = 10 dans le code du prédicat de fermeture). En effet, supposons qu'une gamme se termine en t=289, que la gamme suivante dure 12 unités de temps, alors celle-ci va se terminer en t=301. Elle se terminera virtuellement sans l'ouvrier, lequel est parti en t=300 (par l'événement de départ. Ceci viole la spécification.

Type opération : vue comme ponctuelle (opération 'TimeGranulated' de vitesse = 100% en un pas).

```
bool creerPack_stateTransitionProcedure_Body(EntityMethod* pM) {
//-----------------------------------------------------------------
//
//-----------------------------------------------------------------
  Entity* pConvoyeur = pM->GetEntityArgValue(OPERATION_TARGET);
  <type> descPlateforme = (<type>)(pConvoyeur->GetConstantDescriptor(CONVOYEUR_PLATEFORME));
  Entity* pNewPack = new Pack();
  // ...
  pFloatTab* positionsBriquesAPaqueter =
    pConvoyeur->GetFloatTabConstValue(POSITIONS_BRIQUES_A_PAQUETER);
  for(int k=2;k>0;k--) {
    int index = (int)(positionsBriquesAPaqueter->get_element(k));
    BasicEntity* pMostDistantBrique = descPlateforme->GetValueElement(index);
    pNewPack->AddElement(pMostDistantBrique);
    PositionVide* pPositionVide = new PositionVide();
    descPlateforme->SetValueElement(pPositionVide, index);
    // ...
  }
  // ...
};
```

```
//---------------------
// spécifications
//---------------------
+ I operatedObjectSpecification leConvoyeurSpec,
   preExpandedList << <I><, leConvoyeur>;
;

+ I creationPack uneCreationPack
   operatedObjectSpecAttribute = <I><, leConvoyeurSpec>;
;
```

```
bool creerPack_stateTransitionProcedure_Body(EntityMethod* pM) {
//-----------------------------------------------------------------
//
//-----------------------------------------------------------------
  // ...
  Entity* pConvoyeur = pM->GetEntityArgValue(OPERATION_TARGET);
  // ...
  <type> descPlateforme = (<type>)(pConvoyeur->GetConstantDescriptor(CONVOYEUR_PLATEFORME));
  pFloatTab* positionsBriquesAPaqueter =
    pConvoyeur->GetFloatTabConstValue(POSITIONS_BRIQUES_A_PAQUETER);
  // ...
  for(int k=2;k>0;k--) {
    int index = (int)(positionsBriquesAPaqueter->get_element(k));
    // ...
    PositionVide* pPositionVide = new PositionVide();
    descPlateforme->SetValueElement(pPositionVide, index);
    // ...
  }
  // ...
};
```



Assemblage d'un pack.

Faisabilité : « il existe 2 autres briques … de même taille … ». Cette condition est exprimée dans le prédicat d'ouverture de la séquence d'activités de fabrication (SequenceMiseEnMagasin). Noter que le manager ne fait pas que regarder « s'il existe 2 autres briques … de même taille … » : il écrit les positions des briques en question sur un « post it » qu'il colle sur le convoyeur. C'est l'objet du descripteur PositionsBriquesAPaqueter du convoyeur, exploité dans le code de l'effet de l'opération CreerPack.
Noter que, dans le code du prédicat d'ouverture de la séquence, le développeur a choisi de faire référence au descripteur prédéfini PreExpandedList de la spécification d'objets opérés de sa première (sous-)activité. Cela impose à l'utilisateur de spécifier l'objet opéré (dans le fichier argument .str) par ce descripteur, et non par une spécification d'ensemble d'entités complétée par un opérateur tel que *anyone*, etc.

Entité visée : un convoyeur. Dans le fichier argument .str, on vient de le voir, c'est une instanciation de la classe prédéfinie OperatedObjectSpecification, pour laquelle on value directement la liste (preExpandedList) résultant en principe de l'expansion d'une spécification d'ensemble d'entités.

Effet : création d'un pack vide ; ajout de la brique de tête dans le pack ; placement au rang 0 ; répéter deux fois {repérage du rang *k* de la plus lointaine des briques mémorisées sur le « post it » ; création d'un pack ; transfert de la brique dans le pack ; invocation du tassement de la file à partir de *k+1* ; alimentation du convoyeur}. Si on commençait par la plus proche, le tassement modifierait le rang de la suivante à mettre dans le pack, et le « post it » ne pointerait alors plus sur la brique repérée. Noter aussi que, par convention avec la méthode TassementConvoyeur du convoyeur, on remplace les briques enlevées par des objets virtuels de classe PositionVide.

Type opération : vue comme ponctuelle (opération 'TimeGranulated' de vitesse = 100% en un pas).

Ressources : un ouvrier. Spécification comme ressource propre de l'opération (par opposition à une spécification comme « performer » de l'activité). Noter que, telle qu'est programmée la base, cette spécification sera toujours satisfaite, puisque l'activité (et donc l'opération) ne sera mise en œuvre que si un ouvrier est arrivé sur l'atelier (prédicat d'ouverture de la séquence). On peut provoquer une non satisfaction en demandant, non pas un seul ouvrier, mais par exemple deux : c'est le descripteur SelectorFctArg de la spécification OuvrierOperant.

```
float remplirBriques_setUserFloatQuantitySpeed_Body(EntityMethod* pM)
{
  Operation* pOt = (Operation*)(pM->DescribedEntity());
  float result;
  BasicEntity* pAtelier = GetFirstEntity(ATELIER_REMPLISSAGE);
  BasicEntity* pCuve = pAtelier->GetComponent(CUVE);
  DescriptedEntity* pRobinetVidange =
    (DescriptedEntity*)pCuve->GetComponent(ROBINET_VIDANGE);
  float debit = pRobinetVidange->GetFloatConstValue(DEBIT);
  int stepLength = ClockIntervalToTimeInterval(pOt->GetStep()) / 60;
  result = debit * (float)stepLength;
  return result;
};
```

```
//---------------------
// spécifications
//---------------------
+ SpE lePackDeTete;

+ I operatedObjectSpecification lePackSpec,
   objectEntitySpecAttribute = <SpE><lePackDeTete>;
   selectorFctId = 1; // ANYONE=1
;
//---------------------
// activités primitives
//---------------------
+ I remplissagePack unRemplissagePack
   operatedObjectSpecAttribute =
       + I operatedObjectSpecification,
           preExpandedList << <I><, laCuve>;
       ;
   ;
   destinationSpecAttribute =
       <I><, lePackSpec>;
   transferInformationAttribute =
       + I infoTransfertCuvePack info1;
   ;
;
```

```
float infoTransfertCuvePack_setUserTransferedFloatQuantity_Body(EntityMethod* pM)
{
  float result;
  EntitySpec* pES = App_NewEntitySpecFromString("lePackDeTete");
  pES->Expand();
  pEntityTab* wrappedPack = pES->GetExpansion();
  BasicEntity* lePack = wrappedPack->get_element(0); // nécessairement un seul
  pEntityTab lesBriques = lePack->GetElementTab();
  DescriptedEntity* uneBrique = (DescriptedEntity*)lesBriques[0];
  result = uneBrique->GetFloatConstValue(VOLUME) * 3.; // 3 briques par pack
  return result;
};
```

```
pEntityTab* lePackDeTete_entityListInstantiator_Body(EntitySpecMethod* pM)
{
  EntitySpec* es = pM->DescribedEntitySpec();
  pEntityTab* result = es->GetExpansion();
  result->erase();
  UserClassId classId = pM->GetIntArgValue(CLASSID_ARGUMENT);  // PACK
  Entity* leConvoyeur = (Entity*)GetFirstEntity(CONVOYEUR);
  <type> descPlateforme = (<type>)(leConvoyeur->GetConstantDescriptor(CONVOYEUR_PLATEFORME));
  BasicEntity* pObj = descPlateforme->GetValueElement(0);
  if(pObj->IsInstanceOf(classId))
    result->push_back(pObj);
  return result;
};
```

Remplissage des briques d'un pack.

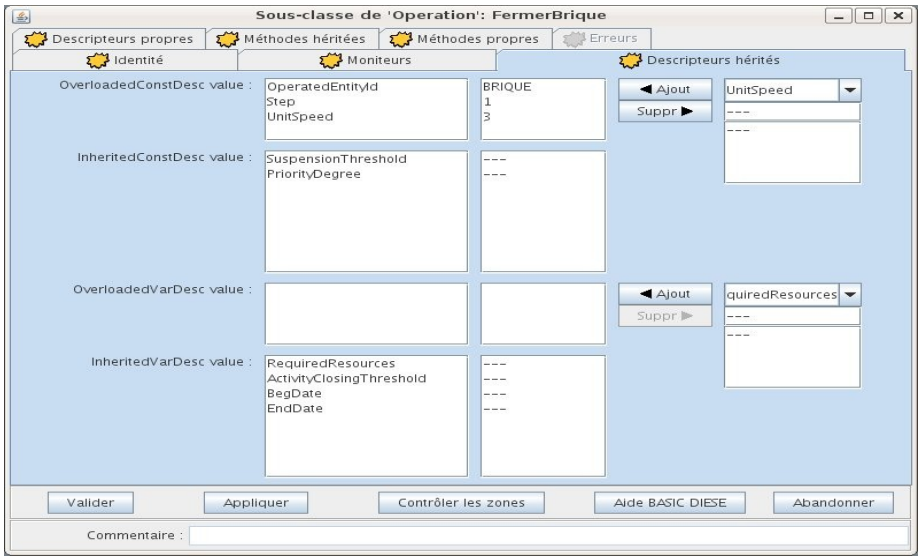
Faisabilité : le niveau de la cuve n'est pas inférieur au volume total des briques. Cette condition est normalement assurée par le moniteur sur le niveau de la cuve, qui provoque un remplissage quand on passe au-dessous du seuil bas.

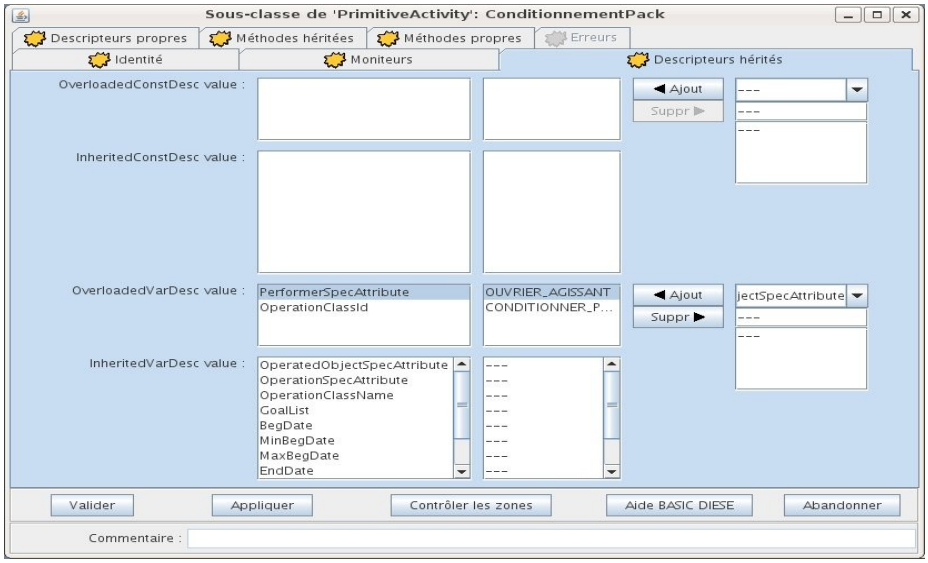
Effet : il est assuré par la méthode StateTransitionProcedure de l'opération prédéfinie de transfert : le niveau de la cuve est diminué du volume total des briques du pack, qui devient plein.

Type opération : à effet progressif (opération 'FloatQuantityGranulated') .
La vitesse est fonction du pas de l'opération et du débit du robinet de vidange. Si le pas est 1 minute et de débit 5 l/min, alors, la vitesse est 5. Si le pas est 30 secondes et de débit 5 l/min, alors, la vitesse est 2,5 (vitesse = débit * pas).
L'activité RemplissagePack est une activité de transfert, qui a de ce fait une source (la cuve, qu'on peut désigner explicitement dans le fichier argument .str : il n'y en a qu'une), une destination (un pack : non désignable *a priori*, mais par une spécification d'objet opéré en position de destination) et enfin une information de transfert.
L'information de transfert (classe InfoTransfertCuvePack) est dotée d'une définition fonctionnelle de la quantité à transférer, puisque celle-ci dépend du volume des briques (petite ou grande) qu'on a assemblées dans le pack en cours de fabrication. La quantité effectivement transférée sera, soit la valeur calculée, soit le contenu effectif de la cuve s'il est inférieur. Il est donc possible qu'un pack soit incomplètement rempli.
La spécification de la destination, c'est d'abord une spécification d'ensemble d'entités qui, par définition, construit une liste de packs. Cette liste veut contenir le pack de tête et n'a donc qu'un seul élément. C'est ensuite la valeur du descripteur SelectorFctId : ici *anyone* … qui de toutes façons n'a pas le choix. Enfin SelectorFctArg est *1* par défaut.
Noter qu'une activité de transfert exige que la source et la destination aient toutes les deux comme descripteur celui dont le symbole de classe est la valeur du descripteur TransferQuantityId de l'information de transfert (ici VolumeContenu).
Noter encore, sur la condition de faisabilité, qu'installer dans le plan une activité de transfert aurait pu nous « assurer le coup », puisque celle-ci regarde, par construction, si la « source » est suffisamment approvisionnée. Il se trouve aussi que ce test n'est opéré que si la source est une ressource à capacité, ce qui ne correspond pas à la conception adoptée. On rappelle donc qu'il est possible qu'un pack soit incomplètement rempli s'il n'y a pas assez de jus dans la cuve et si le seuil bas … est trop bas. L'alimentation manuelle ne peut intervenir ici pour compléter la source, puisque, par son MaxBegStatePredicate, elle a été débrayée (pour cette itération) juste après qu'elle ait eu son unique (IsOneShot=1) chance d'intervenir. Ceci peut être changé en étendant la fenêtre d'ouverture de l'alimentation manuelle (variable locale *longueurFenetreOuverture*).

Entité visée : on l'a vu, c'est une cuve pour la source, et un pack pour la destination. Ces informations sont externalisées dans le fichier argument .str. La cuve est désignée explicitement dans la « preExpandedList » d'une instance directe d'OperatedObjectSpecification. Le pack est désigné par l'opérateur *anyone* sur le résultat de l'expansion d'une spécification d'ensemble d'entités (LePackDeTete), laquelle est codée pour renvoyer le pack situé sous le robinet.

```
// ********************************************************************
// le systeme opérant
// ********************************************************************
+ I ouvrier lOperateurConvoyeur
   availabilityStatus = 0;
   power = 0.25;
;

//---------------------
// spécifications
//---------------------

+ SpE lePackDeTete;

+ I operatedObjectSpecification lePackSpec,
   objectEntitySpecAttribute = <SpE><lePackDeTete>;
   selectorFctId = 1; // ANYONE=1
;
//---------------------
// activités primitives
//---------------------
+ I fermetureBriquesPack desFermeturesBriques
   operatedObjectSpecAttribute = <I><, lePackSpec>;
;

+ I conditionnementPack unConditionnementPack
   operatedObjectSpecAttribute = <I><, lePackSpec>;
/* performer spécifié dans le constructeur */
;
```





Fermeture des briques.

Faisabilité : aucune contrainte.

Effet : réalisé par la méthode StateTransitionProcedure de l'opération FermerBrique : l'état de la brique (descripteur EtatOuverture) passe de *ouvert* à *ferm*é.

Type opération : à effet progressif (opération 'UnitGranulated'). Le moteur de simulation enchaîne, par construction, autant d'invocations de la StateTransitionProcedure qu'il en faut pour traiter toutes les « units » de type 'brique' présentes comme éléments dans les entités spécifiées par la spécification d'objets opérés). de l'activité. De cette spécification (c'est l'instance lePackSpec dans le fichier argument .str), on sait qu'elle 'renvoie' la liste du seul pack de tête : de ce fait, les invocations de la StateTransitionProcedure vont fermer tour à tour les éléments (briques) du pack de tête, ce qui est bien l'effet recherché.
La vitesse est fixée à *3* dans le constructeur de l'opération : les 3 briques du pack sont donc fermées en un seul un pas, lequel dure une seule unité de temps.

Entité visée : on l'a vu, l'opération porte sur une brique, et l'objet opéré par l'activité est le pack de tête.

Conditionnement du pack.

Faisabilité : aucune contrainte.

Entité visée : un pack, spécifié dans le fichier argument .str par l'instance lePackSpec.

Effet : réalisé par la méthode StateTransitionProcedure de l'opération ConditionnerPack. L'objet opéré est le pack de tête (par spécification dans le fichier argument .str) : on ne fait qu'afficher son nom sur la trace de la simulation.

Ressources : un ouvrier, placé en position de « performer » de l'activité, par la spécification de performer OuvrierAgissant. Puisque celle-ci est attachée dans le constructeur de l'activité, l'utilisateur ne peut pas jouer sur le type d'ouvrier (classe Ouvrier, valeur du descripteur EntitySpecReferenceClassId), ni sur son nombre (1, valeur de SelectorFctArg).

Type opération : à effet progressif (opération 'TimeGranulated') . L'opération semble, en regardant son constructeur, définie comme ponctuelle : la vitesse est *1* (= 100% en un pas). Cependant, l'ouvrier étant placé en performer de l'activité (et non en ressource propre de l'opération), sa puissance (descripteur hérité Power) multiplie la vitesse. Si la puissance déclarée dans le fichier argument .str est 0,25, la vitesse est 0,25 et l'opération va durer 4 pas de temps. Si la puissance est 0,50, l'opération va durer 2 pas de temps.

```
bool stockerPack_stateTransitionProcedure_Body(EntityMethod* pM) {
//---------------------------------------------------------------------
//
//---------------------------------------------------------------------
  Entity* pConvoyeur = pM->GetEntityArgValue(OPERATION_TARGET);
  // ...
  <type> descPlateforme = (<type>)(pConvoyeur->GetConstantDescriptor(CONVOYEUR_PLATEFORME));
  BasicEntity* pPack = descPlateforme->GetValueElement(0);
  BasicEntity* pAtelier = pConvoyeur->GetSuperEntity();
  DescriptedEntity* pMagasin = (DescriptedEntity*)pAtelier->GetComponent(MAGASIN_EXPEDITION);
  DescriptedEntity* pUneBrique = (DescriptedEntity*)pPack->GetElement();
  int index = 0;
  if(pUneBrique->IsInstanceOf(MAXI_BRIQUE))
    index = 1;
  BasicEntity* pLoge = pMagasin->GetElement(index);
  // ...
  float production;
  if(index == 0) {
    production = pMagasin->GetFloatVarValue(PRODUCTION_MINI);
    production += 3. * pUneBrique->GetFloatConstValue(VOLUME);
    pMagasin->SetFloatVarValue(PRODUCTION_MINI, production);
  }
  else {
    // ...
  }
  // ...
};
```

```
bool miseAuRebut_openingPredicate_Body(EntityMethod* pM)
{
  // ...
  Entity* pOuvrier = GetFirstEntity(OUVRIER);
  result = (bool)pOuvrier->GetIntConstValue(PRET_A_PAQUETER);
  return ! result;
};
```

```
//---------------------
// spécifications
//---------------------
+ I operatedObjectSpecification leRebutSpec,
   preExpandedList << <I><, leRebut>;
;
//---------------------
// activités primitives
//---------------------
+ I miseAuRebut uneMiseAuRebut
   operatedObjectSpecAttribute = <I><, leRebutSpec>;
;
```

```
bool mettreAuRebut_stateTransitionProcedure_Body(EntityMethod* pM)
{
  Entity* pRebut = pM->GetEntityArgValue(OPERATION_TARGET);
  // ...
  BasicEntity* pAtelier = pRebut->GetSuperEntity();
  Entity* pConvoyeur = pAtelier->GetComponent(CONVOYEUR);
  <type> descPlateforme = (<type>)(pConvoyeur->GetConstantDescriptor(CONVOYEUR_PLATEFORME));
  BasicEntity* pBrique = descPlateforme->GetValueElement(0);
  int indexLoge = 0;
  if(pBrique->IsInstanceOf(MAXI_BRIQUE)) indexLoge = 1;
  BasicEntity* pLoge = pRebut->GetElement(indexLoge);
  // ...
  PositionVide* pPositionVide = new PositionVide();
  descPlateforme->SetValueElement(pPositionVide, 0);
  // ...
};
```

Stockage d'un pack.

Faisabilité : aucune contrainte.

Entité visée : un convoyeur (même mode de spécification que pour la création du pack). Une loge de magasin est aussi touchée, sans la connaître au moment de la spécification. On trouve donc plus pratique de désigner le convoyeur comme entité opérée, et de chercher la loge touchée dans la procédure de transition d'état.

Effet : le pack de tête, remplacé par une position vide, devient élément de la loge du magasin correspondant à la taille des briques. On tasse à partir du rang 1 et on alimente d'une nouvelle brique. Deux descripteurs du magasin (Production[Maxi Mini]) permettent de capter l'évolution de la production des deux types de briques au cours du temps. Ainsi, on pourra visualiser graphiquement ces évolutions dans l'interface utilisateur MI_Diese.

Ressources : un ouvrier, placé en ressource propre de l'opération. Même mode de spécification et même remarque que pour l'opération d'assemblage d'un pack.

Type opération : vue comme ponctuelle (opération 'TimeGranulated' de vitesse = 100% en un pas).

Mise au rebut.

Faisabilité : « il n'existe pas 2 autres briques … ». Codé dans le prédicat d'ouverture de l'activité MiseAuRebut, en exploitant la valeur calculée dans le prédicat d'ouverture de la séquence d'activités de fabrication, et mise au frigo dans le descripteur PretAPaqueter de l'ouvrier.
Lors de l'interprétation de l'opérateur de disjonction (*or*) le moteur de simulation construit dynamiquement un ensemble de jeux d'activités primitives alternatifs. S'il n'y avait pas de conditions d'ouverture sur les activités en jeu, cela se réduirait ici à deux jeux d'une seule activité : {(assemblage) *vs.* (mise au rebut)}. A cette étape, un jeu de préférences (basé sur la taille des briques sur la convoyeur : « y en a-t-il 3 identiques ? ») devrait choisir le jeu d'activités à privilégier, l'autre étant annulé. On a plutôt choisi d'exprimer cette préférence par des conditions d'ouverture, complètement contradictoires, pour les activités mise en magasin et mise au rebut. Ainsi, si la condition d'une des deux activités est satisfaite, celle de l'autre ne l'est pas. Une activité va donc démarrer et, par construction de l'opérateur *or*, l'autre sera automatiquement annulée.

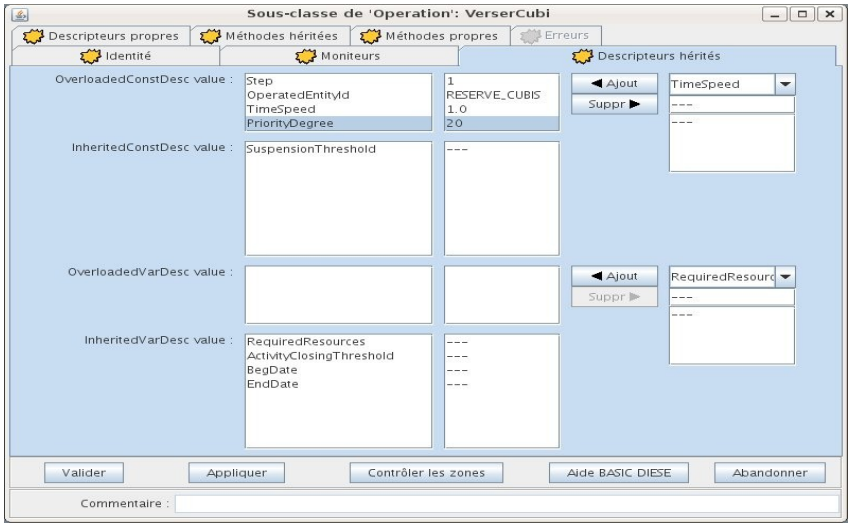
Entité visée : un rebut. C'est une des loges du rebut qui est touchée, sans la connaître au moment de la spécification. On désigne cependant ici le rebut comme entité opérée, et on cherche la loge alimentée, ainsi que la brique à jeter, dans la procédure de transition d'état. Le rebut est désigné dans le fichier argument .str : c'est une instanciation de la classe prédéfinie OperatedObjectSpecification, pour laquelle on value directement la liste (preExpandedList) résultant en principe de l'expansion d'une spécification d'ensemble d'entités.

Effet : la brique au rang 0 est enlevée du convoyeur et ajoutée dans la loge du rebut correspondant à sa taille ; tassement à partir du rang 1 ; alimentation du convoyeur. On n'oublie pas de remplacer la brique mise au rebut par une position vide (le tassement récupère la mémoire allouée à l'objet de tête : il ne faut pas que ce soit la brique !).

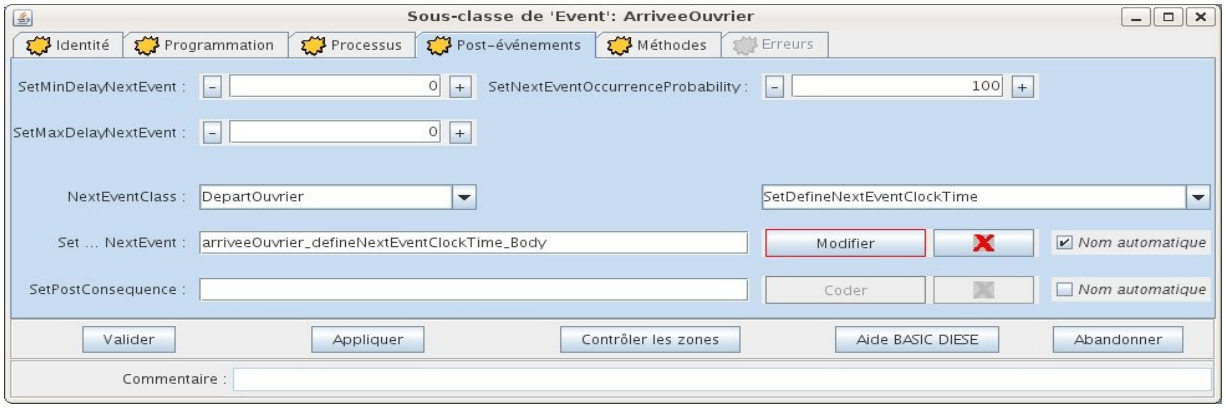
Ressources : aucune.

Type opération : vue comme ponctuelle (opération 'TimeGranulated' de vitesse = 100% en un pas).

```
//---------------------
// activités primitives
//---------------------
+ I alimentationManuelle uneAlimManuelle
   operatedObjectSpecAttribute =
       + I operatedObjectSpecification,
           preExpandedList << <I><, laReserveCubis>;
       ;
   ;
;
```




```
{i2} = <pi><"POSTE" "heureDebutAM"> - <pi><"SIMULATION" "heureDebut">;

+ V arriveeOuvrier debutPoste
   DATE_OCCUR = {i2}*60;
   I resourceMobilizationProcess = <I><ouvrier lOperateurConvoyeur>;
;
```




```
int arriveeOuvrier_defineNextEventClockTime_Body(EventMethod* pM)
{
  int departClockTime;
  Event* pArriveeEvent = pM->GetEventArgValue(PREVIOUS_EVENT_ARGUMENT);
  int arriveeClockTime = pArriveeEvent->OccurrenceClockTime();
  int duree = gIntParam_POSTE_duree_;
  departClockTime = arriveeClockTime + duree;
  // ...
};
```

```
// System parameters
// ----------------------------------
   // ...
   "SIMULATION" "heureDebut" <- 6 "";
   "POSTE"      "duree"      <- 240 "";
   "POSTE"      "heureDebutAM"   <- 8 "";
   // ...
```

Alimentation manuelle de la cuve.

Faisabilité : le niveau de la cuve est en dessous du seuil bas et le stock de cubitainer n'est pas vide. La seconde condition est assurée vraie parce que l'itération de production s'arrête dès que le stock de cubis est vide (voir plus loin). Quant à la première, on la code dans le prédicat d'ouverture de l'activité d'alimentation manuelle. Une alternative mécaniquement valide serait la condition de faisabilité de l'opération : c'est moins approprié, parce l'alimentation peut techniquement s'opérer même si le seuil n'est pas très bas ; ne pas faire d'alimentation est un choix de pilotage à transcrire dans la stratégie, donc dans les activités.

Entité visée : un peu arbitrairement : la réserve de cubitainers. Une alternative valide serait la cuve. La réserve est désignée explicitement dans la « preExpandedList » d'une instance directe d'OperatedObjectSpecification. On remarque que cette instance est créée dans l'expression qui affecte une valeur au descripteur OperatedObjectSpecAttribute, alors qu'on l'avait isolée, par exemple et de manière arbitraire, pour la mise au rebut.

Effet : le premier cubitainer est enlevé du stock ; le niveau de la cuve est augmenté du volume du cubitainer. On pourrait récupérer, dans la procédure de transition d'état, la mémoire allouée au cubi.

Ressources : aucune.

Type opération : vue comme ponctuelle (opération 'TimeGranulated' de vitesse = 100% en un pas). Noter la priorité de l'opération (*20*), supérieure à celles (*99*, attribuées par défaut) des différentes activités primitives de la fabrication d'un pack, ou de la mise au rebut de la brique de tête. Ainsi, à une date donnée, l'alimentation manuelle interviendra avant toute autre.

Arrivées et départs de l'ouvrier.

Il ne s'agit plus, là, d'éléments de la stratégie de pilotage. Plus d'activités ni d'opérations donc, mais des événements exogènes auxquels le pilotage doit s'adapter.
On profite de l'existence de processus prédéfinis de mobilisation et d'immobilisation de ressource, dont on écrit des spécialisations. Une mobilisation (ici l'arrivée de l'ouvrier) génère, par construction, une immobilisation. Une immobilisation (ici le départ de l'ouvrier) génère, par construction, une mobilisation. On peut ainsi entretenir une alternance d'arrivées et de départs.
Il y a donc deux classes d'événements, ArriveeOuvrier et DepartOuvrier, chacun ayant le rôle de « NextEvent » pour l'autre.
La méthode prédéfinie (au niveau de la classe Event de DIESE) DefineNextEventClockTime n'a plus qu'à dater le prochain départ (avec la durée de la plage de travail, qui est un paramètre du système, lu dans le fichier argument .par)) ou la prochaine arrivée (deux paramètres dont les valeurs peuvent être par exemple 14h et 8h le lendemain).
La ressource sur laquelle portent les processus, c'est l'ouvrier, spécifié dans le fichier argument .str, Soit explicitement (ligne « I resourceMobilizationProcess = <I><ouvrier,> ; »), soit implicitement (le moteur considèrera la première ressource déclarée, ici la seule, l'ouvrier). On aurait aussi pu créer une spécialisation de ResourceMobilizationProcess, préciser ses propriétes (notamment l'entité opérée) et l'attacher (dans le constructeur ou dans le fichier argument .str) à l'événement de mobilisation. Le moteur de simulation se charge de dire à l'immobilisation que c'est le même ouvrier qui est touché.

```
//----------------------
// activité agrégées
//----------------------
+ I activityOptional alimManuelleIfNeeded
   <- E <I><, uneAlimManuelle>;
   isOneShot = 1;
;
```

```
bool alimentationManuelle_maxBegStatePredicate_Body(EntityMethod* pM)
{
  Entity* pAlim = pM->DescribedEntity();
  bool result = FALSE
  ActivityIteration* pCycleProduction = ((Activity*)pAlim)->IsIterated();
  <type> pConjunction = (<type>)pCycleProduction->GetElement();
  int instantOuverture = pConjunction->GetIntVarValue(BEG_DATE);
  int longueurFenetreOuverture = 0;
  //int longueurFenetreOuverture = 4;
  if(pCurrentSim->Clock() > instantOuverture + longueurFenetreOuverture) {
    result = TRUE;
  }
  return result;
};
```

Remarque générale sur les activités agrégées (opérateurs)

Un développeur d'application peut coder un opérateur sur ses activités primitives, dont il ressent le besoin et qui ne serait pas prédéfini dans la couche CONTROL DIESE. Mais en règle (très) générale les activités agrégées seront des instances directes ou indirectes des opérateurs prédéfinis. On examine dans la suite celles introduites dans notre application : l'optionalité, le choix (disjonction), l'ensemble non ordonné (conjonction), la séquence et l'itération.
Instance directe signifie qu'on ne crée pas de spécialisation de l'opérateur prédéfini : par exemple on crée une instance d'ActivityOptional. Instance indirecte signifie on va créer puis instancier, par exemple, IterationMagasinOuRebut, spécialisation propre à notre application d'ActivityIteration.

On introduit une spécialisation propre pour une des deux raisons suivantes :

- on souhaite « internaliser » une propriété de l'opérateur : par exemple le délai entre la fermeture d'une activité et l'ouverture de la suivante dans une séquence. On pourrait valuer le descripteur CloseToOpenDelays d'une instance directe de ActivityBefore dans le fichier argument .str, mais on préfère ne pas donner cette possibilité à l'utilisateur. La valuation sera donc codée dans le constructeur d'une spécialisation d'ActivityBefore.
- le comportement prédéfini de l'opérateur est insuffisant : par exemple, la procédure du moteur qui stoppe l'enchaînement des activités dans une itération en fonction des descripteurs (par exemple) MinEndDate et/ou MaxNumberOfReplications. Le comportement spécifique requis sera codé dans la méthode ClosingPredicate d'une spécialisation d'ActivityIteration propre à l'application.

Optionalité de l'alimentation manuelle

On place l'activité d'alimentation manuelle comme unique élément d'une activité 'option' (classe prédéfinie ActivityOptional). On décide d'externaliser la valeur du descripteur IsOneShot. On pourra donc observer la différence entre les valeurs 1 et 0 lors de deux simulations successives.
Avec la valeur 0, le moteur de simulation donne à l'alimentation la possibilité de s'ouvrir tant que la fin de la fenêtre d'ouverture n'est pas dépassée (c'est le comportement standard, celui qui laisse tout son pouvoir au concept de fenêtre d'ouverture). On a codé pour l'alimentation un prédicat MaxBegStatePredicate (sans le complémenter par une valeur du descripteur MaxBegDate). Dès que ce prédicat est satisfait, la fenêtre d'ouverture est dite dépassée, et l'activité 'option' est fermée. Plus précisément, on a codé que ce prédicat serait vrai passé un certain délai (valeur de la variable locale *longueurFenetreOuverture*) après l'ouverture. Si le délai est *0*, et que l'ouverture a eu lieu en t=x[4], alors la fermeture sera « prononcée » en t=$x+1$. De ce fait, la prochaine création d'un pack ou la prochaine mise au rebut aura lieu en t=$x+2$. C'est parce qu'on peut considérer que l' de temps $x+1$ est « perdue » qu'on a exploité l'autre valeur du descripteur IsOneShot.
Avec la valeur 1, le moteur de simulation ne donne qu'une seule chance à l'activité optionnelle (l'alimentation) de s'ouvrir, dès lors que les conditions d'ouverture de l'activité 'option' sont satisfaites. Concrètement, cela est testé dans l'unité de temps qui suit celle au cours de laquelle un pack a été mis en magasin ou une brique au rebut. Si le test de la condition d'ouverture est négatif, alors l'éventualité d'une alimentation manuelle ne sera examinée qu'après le cycle de production suivant. La spécification de fin de fenêtre d'ouverture (MaxBegStatePredicate) est ignorée. Dans l'exemple pris dans l'alinéa précédent, si l'ouverture a eu lieu en t=x, alors la fermeture sera « prononcée » en t=x. De ce fait, la prochaine création d'un pack ou la prochaine mise au rebut aura lieu en t=$x+1$, et cette unité de temps ne sera pas « perdue ».

[4] On rappelle que dans l'expression 't = x', x désigne un point quelconque dans l'intervalle de temps [x $x+1$[, avec l'unité spécifiée dans le fichier .sim. En d'autres termes, les instants (réels) dans cet intervalle ne sont pas distinguables et sont tous désignés par x.

```
//---------------------
// activités agrégées
//---------------------
+ I iterationMagasinOuRebut laProduction
   + E activityConjunction unCycleDeProduction,
       + E activityDisjunction magasinOuRebut,
               <- E <I><, uneMiseEnMagasin>;
               //
               <- E <I><, uneMiseAuRebut>;
       ;
       // ...
   ;
;
```

```
//---------------------
// activités agrégées
//---------------------
+ I sequenceMiseEnMagasin uneMiseEnMagasin
   <- E <I><, uneCreationPack>;
   <- E <I><, unRemplissagePack>;
   <- E <I><, desFermeturesBriques>;
   <- E <I><, unConditionnementPack>;
   <- E <I><, unStockagePack>;
;
```

```
bool sequenceMiseEnMagasin_openingPredicate_Body(EntityMethod* pM)
{
  BasicEntity* pBefore = pM->DescribedEntity();
  PrimitiveActivity* pCreation = (PrimitiveActivity*)pBefore->GetElement(0);
  Entity* pOOSpec = pCreation->GetEntityVarValue(OPERATED_OBJECT_SPEC_ATTRIBUTE);
  Entity* pConvoyeur = (Entity*)(pOOSpec->GetEntityTabVarValue(PRE_EXPANDED_LIST)-
>get_element(0));
  // ...
  <type> descPlateforme = (<type>)(pConvoyeur->GetConstantDescriptor(CONVOYEUR_PLATEFORME));
  BasicEntity* pFirstBrique = descPlateforme->GetValueElement(0);
  UserClassId briqueAPaqueterId = pFirstBrique->ClassSymbol();
  int nbSlots = pConvoyeur->GetIntConstValue(CONVOYEUR_LONGUEUR);
  pFloatTab* positionsBriquesAPaqueter = new pFloatTab();
  positionsBriquesAPaqueter->push_back((float)0);
  int nbBriquesAPaqueter = 1;
  for(int k=1;k<nbSlots;k++) {
    if(descPlateforme->GetValueElement(k)->ClassSymbol() == briqueAPaqueterId) {
      positionsBriquesAPaqueter->push_back((float)k);
      nbBriquesAPaqueter++;
    }
    if(nbBriquesAPaqueter == 3) {
      result = TRUE;
      break;
    }
  }
  pConvoyeur->SetFloatTabConstValue(POSITIONS_BRIQUES_A_PAQUETER,
                   positionsBriquesAPaqueter);
  // ...
}
```

Choix entre mise en magasin et mise au rebut

Les deux activités sont éléments d'une instance directe de ActivityDisjunction, déclarée dans le fichier argument .str.
On a vu dans les commentaires sur les deux activités primitives qu'une et une seule des deux peut être ouverte dès lors que la disjonction peut l'être aussi. Ceci n'est pas requis par le moteur de simulation ni la conception de la disjonction : si aucun élément d'une disjonction n'est ouvrable alors que celle-ci l'est, le test d'ouverture est renouvelé tant qu'on n'est pas sorti de la fenêtre d'ouverture de tous les éléments ; et dès qu'un élément est ouvert, les autres sont automatiquement annulés, s'ils ne sont pas déjà fermés.
On pourrait penser à une alternative consistant à remplacer la disjonction : or(a, b), par une conjonction d'activités optionnelles : and(opt(a), opt(b)), avec la justification suivante : comme un des prédicats d'ouverture de *a* et *b* serait vrai et l'autre faux, a ou *b* serait mise et oeuvre ; l'autre ne le serait pas, sans que ça soit un échec du plan. Il se trouve que ceci ne peut fonctionner que si *a* et *b* sont munies d'un MaxBegStatePredicate, ce qui réduit l'intérêt de l'alternative. En effet, si *a* est ouverte et mise en œuvre, *b* devient NoLongerOpenable quand son prédicat MaxBegStatePredicate devient vrai. Alors la conjonction est fermée et on passe à l'itération suivante. Sans prédicat pour imposer une limite, *b* ne serait jamais fermée, ni la conjonction. Et on n'itèrerait donc pas.

La gamme de fabrication d'un pack

Les activités d'assemblage, remplissage, fermeture, conditionnement et stockage sont les éléments, ajoutés dans cet ordre, d'une instance de la classe SequenceMiseEnMagasin. Cette spécialisation d'ActivityBefore est caractérisée par son prédicat d'ouverture, justifié et commenté avec l'activité primitive d'assemblage d'un pack.
Il existe un autre opérateur de mise en séquence : ActivityMeeting. Aurait-il pu être utilisé à la place d'ActivityBefore ? En principe : oui, si et seulement s'il n'y a aucune contrainte à l'ouverture immédiate d'un élément après la fermeture du précédent. Il se trouve que c'est le cas dans notre configuration. En effet, la seule contrainte envisageable est l'indisponibilité de l'ouvrier. Or, par notre construction de l'alternance mobilisation/immobilisation l'ouvrier est assuré disponible durant toute la plage de travail.

Conjonction entre fabrication d'un pack et éventuelle alimentation manuelle

On souhaite exprimer que chaque cycle de production comprend aussi (éventuellement) une alimentation manuelle, mais sans préciser si celle-ci doit avoir lieu avant, pendant ou après l'assemblage d'un pack ou la mise au rebut d'une brique. L'opérateur approprié est la conjonction. On crée donc, dans le fichier argument .str, une instance directe d'ActivityConjunction, avec deux éléments (la disjonction et l'activité 'option').
Pour qu'on puisse observer une alimentation « pendant » la séquence (*e.g.* entre remplissage et fermeture), il faut qu'on ait spécifié une valeur de la variable locale *longueurFenetreOuverture* (dans le MaxBegStatePredicate, voir plus haut) suffisamment grande pour que, dans le cas où l'alimentation manuelle ne disposerait pas de toutes ses ressources en début de cycle, le moteur de simulation « patiente » pour s'apercevoir que les ressources sont devenues disponibles après le remplissage. Bien entendu il ne faut pas que le MaxBegStatePredicate ait été désactivé par la valeur *1* du descripteur IsOneShot). Ceci est théorique, puisqu'on a vu que l'alimentation manuelle, dans la version courante de l'application, n'est nullement contrainte par ses ressources.
Enfin, on aurait pu contraindre l'alimentation à intervenir seule en début ou en fin de séquence (ActivityBefore à la place de la conjonction), ou bien « au milieu » (ActivityInclusion). Pour qu'elle intervienne en même temps que l'assemblage ou la mise au rebut (resp. le stockage ou la mise au rebut), on aurait utilisé une ActivityCostarting (resp. ActivityCoending), et on aurait fait en sorte que les ressources éventuellement requises par les deux activités soient disponibles en même temps.

```
//---------------------
// activités agrégées
//---------------------
+ I iterationMagasinOuRebut laProduction
   + E activityConjunction unCycleDeProduction,
       // ...
   ;
;
```

```
bool OuvrierPartBientot(int delai) {
  bool result = FALSE;
  int now = pCurrentSim->Clock();
  pCurrentSim->GoHeadInAgenda();
  while(pCurrentSim->IsCurrentInAgenda()) {
    Event* pEvt = pCurrentSim->GetCurrentInAgenda();
    if(! strcmp(pEvt->ClassName(), "departOuvrier")) {
      int dateDepart = pEvt->OccurrenceClockTime();
      if((dateDepart - now) < delai) {
   result = TRUE;
      }
    }
    pCurrentSim->GoNextInAgenda();
  }
  return result;
}
```

```
bool iterationMagasinOuRebut_closingPredicate_Body(EntityMethod* pM)
{
  // ...
  if(OuvrierPartBientot(10))
    return TRUE;
  // ...
};
```

```
bool miseEnRoute_openingPredicate_Body(EntityMethod* pM)
{
  // ...
  Entity* pOuvrier = GetFirstEntity(OUVRIER);
  if(pOuvrier->GetIntVarValue(AVAILABILITY_STATUS) != 1) {
    return FALSE;
  }
  if(OuvrierPartBientot(10)) {
    return FALSE;
  }
  // ...
};
```

La répétition de la production d'un pack (ou de la mise au rebut)

De manière générale, pour exécuter plusieurs fois une activité (dite 'itérée') en ne la spécifiant qu'une seule fois, on la place comme (seul) élément d'une itération (ActivityIteration). Le moteur de simulation, après avoir fermé l'activité itérée, la remet en position d'attente d'une nouvelle ouverture, conditionnellement (i) aux nombres minimum et maximum d'itérations autorisées, (ii) aux conditions de fermeture de l'itération elle-même (fenêtre et/ou prédicat, classiquement) et (iii) auxs contraintes induites par les activités du plan dont l'itération est élément direct ou indirect.
Notre itération démarrera dès que l'activité de mise en route sera fermée : notre problème est plutôt « quand et comment l'arrêter ? ». Selon l'alinéa précédent, on peut jouer soit sur la spécification de l'itération soit sur celle d'une activité qui lui est connectée par un opérateur.
En suivant la seconde option, on pourrait penser provoquer l'arrêt de l'itération en spécifiant une date d'ouverture pour l'activité d'arrêt, qui la suit dans une séquence. Fausse piste, parce que dans un before(*a*, *b*) l'OpeningPredicate de *b* n'est testé que quand *a* est candidate à la fermeture. Or notre itération ne le sera jamais car liée à aucune autre activité (*e.g.* un ActivityCoending) qui pourrait demander sa fermeture. On suit donc la première piste. Et là, plusieurs « sous-pistes ».
Contraindre le nombre d'itérations par un intervalle [min max] n'est pas approprié. D'une part, les opérations de mise en magasin et de mise au rebut ne sont pas d'égales durées. D'autre part, on risque de laisser l'ouvrier inoccupé dans l'atelier ou de lui faire louper l'heure de la sortie.
Spécifier une fenêtre de fermeture datée fait perdre de la souplesse à la gestion de l'atelier : on rappelle que la fin de la plage de travail est provoquée par un événement externe au système qui est l'heure de la pause, définie indépendamment.
Spécialiser la méthode prédéfinie UpdateReactivatedSon ? Cette méthode est invoquée une fois que le moteur a remis l'activité itérée en attente d'ouverture, et on peut y modifier, typiquement sa fenêtre d'ouverture. L'astuce serait de donner une fenêtre déjà dépassée. N'y pensons plus pour deux raisons. D'abord, c'est un peu complexe et pas très naturel. Mais surtout, pour qu'UpdateReactivatedSon soit invoquée, il faut que la précédente itération soit fermée, c'est-à-dire que toutes les activités sous-jacentes soient fermées. Dans le cas où une de ces activités (non optionnelles) auraient ses conditions de mise en œuvre définitivement non satisfaites (départ de l'ouvrier), elle ne pourrait *a fortiori* pas être fermée, empêchant l'itération de se fermer.
Reste l'exploitation du ClosingPredicate, lieu naturel d'expression fonctionnelle d'une condition de fermeture, testé tant que l'itération est ouverte et par conséquent à chaque remise à jour de la situation du plan (événement UpdateSituationEvent, voir le fichier argument .str) pendant la mise en œuvre de l'activité itérée. Le moteur de simulation est programmé de telle sorte que, si le ClosingPredicate devient vrai pendant, par exemple, le conditionnement du pack, alors l'activité itérée continue à être mise en œuvre (jusqu'au stockage compris) puis l'itération est fermée (et l'activité itérée non remise en attente, bien entendu). On n'est cependant pas à l'abri d'une incapacité à fermer ce qui sera la dernière activité itérée, rencontrée dans l'alinéa précédent. On est donc amené à anticiper les situations qui empêchent de fermer l'activité itérée. En d'autres termes on va fermer l'itération juste avant qu'on n'ait plus le temps de terminer proprement un cycle 'magasin ou rebut'. Un peut avoir une estimation plus ou moins fine du temps nécessaire pour réaliser un cycle. Fondée sur le cas le plus défavorable (mise en magasin d'un pack de grandes briques), elle peut aussi tenir compte de la 'puissance' de l'ouvrier, qui module, on l'a vu, la durée du conditionnement. Dans notre code, l'estimation, grossière, est de 10 unités de temps. On peut s'exercer en remplaçant cette valeur par la valeur d'un paramètre du système. Le ClosingPredicate renvoie donc vrai, dès qu'on est à moins de 10 UT de la date d'occurrence du prochain (et d'ailleurs seul, par construction) événement d'immobilisation de l'ouvrier, analysé comme seule cause possible de l' « empêchement » d'une activité.
Quand l'itération sur la production des packs est fermée un peu avant le départ effectif de l'ouvrier, l'opération d' « Arrêt » (de l'atelier) est exécutée, ce qui provoque la mise en attente de la plage de travail suivante. Pour que celle-ci ne soit pas rouverte immédiatement, le prédicat d'ouverture de la mise en route s'assure que l'ouvrier ne part pas « bientôt ».

```
//---------------------
// finalement, le plan
//---------------------
<I> <,iterationPlagesTravail>
   + E activityBefore sequencePlageTravail,
       <- E <I><, laMiseEnRoute>;
       //
       <- E <I><, laProduction>;
       //
       <- E <I><, lArretProduction>;
   ;
       readyToRun = 1;
;
```

```
//initialisation de la simulation de PackMan (fichier .sim)
//TRACE PARSING;
INITIALISATION SIMULATION
   // ...
   DATE_DEBUT 1 JAN 07 <pi><"SIMULATION" "heureDebut"> 0 0;
   DATE_FIN   2 JAN 07;
   // ...
;
```

```
main (int argc, char* argv[])
{
   // ...
   ParseDirectiveFile(argv[5]);
   //
   // Observation situation des activites du plan en fin de simulation
   //
   BasicEntity* pArb = GetFirstBasicEntity(ACTIVITIES_RESOURCES_BLOCK);
   Activity* pRootAct = (Activity*)(pArb->GetComponent(ACTIVITY));
   pRootAct->DisplaySituation();
   DeleteEntityInstances();
   // ...
   DeleteEntityClasses();
   delete pCurrentSim;
   // ...
}
```

```
// Ce fichier contient les directives de simulation.

// ...

RUN;

// ...

DELETE ENTITYSPEC;
DELETE MONITOR;
DELETE FILE;
DELETE PARAMETRE;
//
// Inhibation lignes ci-dessous : pour DisplaySituation dans main.cc
//
//DELETE ENTITE INSTANCE;
// ...
//DELETE ENTITE CLASSE;
//DELETE SIMULATION;

// ...
```

La structure d'une plage de travail, entre arrivée et départ de l'ouvrier

C'est une séquence (ActivityBefore) { mise en route, itération de la production, arrêt } sans contrainte spécifique : l'ouverture, les transitions entre éléments et la fermeture sont provoquées par les spécifications sur les éléments, déjà commentées dans les sections précédentes.

La répétition des plages de travail

C'est une itération (ActivityIteration), avec la séquence ci-dessus comme élément, et sans contrainte spécifique.
Lorsque le plan est instancié, il est « dormant ». Puisque, par définition, il n'est l'objet d'aucun opérateur, sa mise en attente d'ouverture doit être provoquée d'une autre manière : c'est le rôle de la valeur *1* affectée au descripteur prédéfini ReadyToRun. Un moniteur prédéfini saisi cette affectation pour réaliser la mise en attente d'ouverture.
Il n'y a pas de pré-condition, portant sur l'activité itérée, à l'ouverture d'une itération. Comme d'autre part cette itération, en tant que racine du plan, n'est contrainte par aucun autre opérateur, son ouverture est immédiate en t=*0*. Il faudra cependant attendre l'arrivée de l'ouvrier pour l'ouverture de la 'mise en route' et donc de la première plage de travail.
L'enchaînement des plages de travail est réglée naturellement : lorsqu'une plage est fermée, par départ de l'ouvrier, lequel provoque la mise en œuvre de l'activité d'arrêt, alors le moteur de simulation remet une plage de travail en attente d'ouverture. Celle-ci interviendra dès que l'ouvrier reviendra, provoquant l'ouverture de la mise en route, donc de la nouvelle plage.
Enfin, quant à la fermeture de l'itération sur les plages de travail, elle ne peut être provoquée que par ses spécifications propres, toujours parce que l'itération, racine du plan, n'est contrainte par aucun autre opérateur. Or aucune spécification propre n'affecte l'itération, disons par choix du modélisateur. La conséquence est que la simulation s'arrêtera, à la date précisée dans le fichier argument .sim), sans que le plan ne soit fermé ! On peut le vérifier en adressant le message DisplaySituation à l'itération dans le programme principal, après la fin de la simulation. Plus précisément après l'appel du parseur de fichier de directives de simulation (ParserDirectiveFile), mais en ayant pris garde de débrayer la récupération de la mémoire allouée aux entités (donc aux activités) dans le fichier argument .dir) (celui qui est interprété par ParserDirectiveFile).

```
//initialisation de la simulation de PackMan (fichier .sim)
//TRACE PARSING;
INITIALISATION SIMULATION
   // ...
   DATE_DEBUT 1 JAN 07 <pi><"SIMULATION" "heureDebut"> 0 0;
   DATE_FIN   2 JAN 07;
   // ...
;
```

```
//initialisation de la simulation de PackMan (fichier .sim)
//TRACE PARSING;
INITIALISATION SIMULATION
   UNITE_TPS MINUTE; //DAY HOUR SECOND
   NB_UNITE_TPS 1;
   // ...
;
```

```
// ************************************************************************
// la dynamique du système
// ************************************************************************
+ P fluxEntreeCuveLectureProcess lectureFluxEntree
   // …
   INIT_PRCD_EVT        DATE_OCCUR = 0;
                        // …
   ;
;
```

```
bool mettreEnRoute_stateTransitionProcedure_Body(EntityMethod* pM)
{
  Entity* pAtelier = pM->GetEntityArgValue(OPERATION_TARGET);
  // ...
  Entity* pCuve = pAtelier->GetComponent(CUVE);
  float niveauCourant = pCuve->GetFloatVarValue(VOLUME_CONTENU);
  float niveauMaxi = pCuve->GetFloatConstValue(VOLUME_MAXI_CONTENU);
  if(niveauCourant < niveauMaxi) {
    // ...
    StartAlimAuto();
  }
  else {
    // ...
  }
  // ...
};
```

```
bool mettreEnRoute_stateTransitionProcedure_Body(EntityMethod* pM)
{
  Entity* pAtelier = pM->GetEntityArgValue(OPERATION_TARGET);
  // ...
  Entity* pConvoyeur = pAtelier->GetComponent(CONVOYEUR);
  pConvoyeur->ExecVoidMethod(INITIALISATION_CONVOYEUR);
  // ...
};
```

3.3) La dynamique du système

3.3.1) Dynamique du système technique

Le début de la période de simulation est spécifié par le DATE_DEBUT du fichier argument .sim. C'est la valeur 0 de l'horloge qui marque le temps simulé. Le temps simulé est celui que traverse le système simulé, par opposition au temps réel, celui au cours duquel se déroule la simulation sur une machine. L'horloge progresse d'une valeur à une autre, obligatoirement supérieure. La plus petite progression possible est l'unité de temps simulé. Cette unité est spécifiée dans le fichier des paramètres de la simulation (.sim), par un certain nombre d'une certaine unité de temps naturelle (ici *1* minute). Toutes les valeurs spécifiées ou affichées de l'horloge se réfèrent à l'unité de temps simulé. Ainsi, la valeur *240* du paramètre d'identifiants « POSTE » et « duree » doit être interprétée comme *240* minutes ou *4* heures, etc.

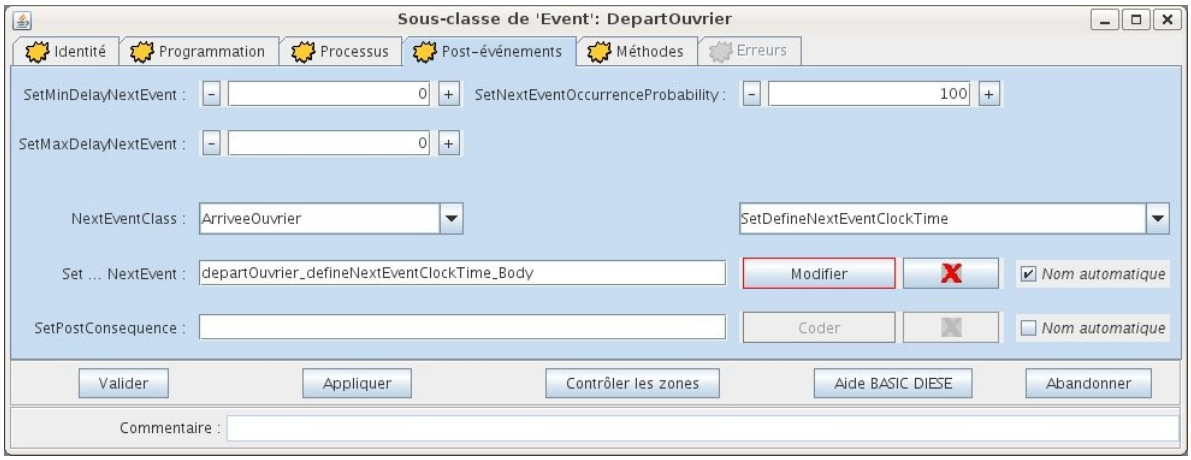
L'arrivée de jus de fruit

L'arrivée de jus de fruit en amont du robinet d'alimentation (fluxEntreeCuveLectureProcess) débute au début de la période de simulation (DATE_OCCUR = *0* dans le .str,). Elle est stoppée par l'épuisement de la « source » : fin du fichier (dans la conception adoptée) ou arrêt de la génération dynamique. Supposée indépendante du fonctionnement du système simulé, cette arrivée est programmée dans l'agenda en dehors de la gestion de l'atelier (c'est-à-dire les procédures liées aux activités et opérations).

L'alimentation automatique

L'alimentation automatique est contrôlée par un événement « unique » en t=*0* (conception 1) ou bien un événement programmé (pour intervenir immédiatement) à la suite de chaque arrivée de jus de fruit (dans la conception 2 adoptée). Cet événement initie et lance le transfert dans la cuve du jus qui vient d'arriver. Il est programmé dans l'agenda une première fois, puis autogénéré.
L'insertion de l'événement est réalisée par un appel à la fonction globale StartAlimAuto dans la procédure de transition d'état de la mise en route, c'est-à-dire au début de chaque plage de travail.
L'auto-génération est gérée par la méthode spécifiée dans le 'SetGenerateNextEvent' de Solfege. Elle fait appel à un paramètre du système dont la valeur doit être égale à l'intervalle de temps entre deux arrivées de jus de fruit, c'est-à-dire entre les instants correspondant à deux lignes successives dans le fichier de données séquentielles. L'auto-génération est arrêtée par un appel à la fonction globale StopAlimAuto dans la procédure de transition d'état de l'opération d'arrêt, c'est-à-dire à la fin de chaque plage de travail. En réalité, l'arrêt s'opère en ôtant l'événement de l'agenda : lorsque la fonction d'auto-génération est invoquée, l'ouvrier est toujours disponible et elle renvoie *vrai*.
L'alimentation peut aussi être arrêtée au cours d'un transfert de jus dans la cuve, par le moniteur sur le volume de la cuve, au moment de la fermeture du robinet d'alimentation provoquée par le passage en dessus du seuil haut. L'événement est régénéré par le moteur de simulation (et programmé au même moment que la prochaine arrivée de jus de fruit), mais le processus aura vraisemblablement sa pré-condition non satisfaite (le robinet doit être ouvert). L'alimentation sera reprise quand le même moniteur aura rouvert le robinet après que le niveau sera redescendu sous le seuil moyen (rendant la pré-condition à nouveau satisfaite). Noter que ce n'est pas l'appel de StartAlimAuto qui est déterminant en cette circonstance, puisque l'événement est déjà dans l'agenda, « en position d'attente ».

```
// ----------------------------------
// System parameters
// ----------------------------------
   // ...
   "SIMULATION" "heureDebut" <- 6 "";
   "POSTE"      "duree"      <- 240 "";
   "POSTE"      "heureDebutAM"   <- 8 "";
   "POSTE"      "heureDebutPM"   <- 14 "";
   // ...
```




```
int departOuvrier_defineNextEventClockTime_Body(EventMethod* pM)
{
  int arriveeClockTime;
  Event* pDepartEvent = pM->GetEventArgValue(PREVIOUS_EVENT_ARGUMENT);
  int departClockTime = pDepartEvent->OccurrenceClockTime();
  Date departEventDate = ClockToDate(departClockTime);
  int heureDebutAM = gIntParam_POSTE_heureDebutAM_;
  int heureDebutPM = gIntParam_POSTE_heureDebutPM_;
  if(departEventDate.hour < heureDebutPM) {
    arriveeClockTime =
      DateToClock(departEventDate.mday, departEventDate.month, departEventDate.year,
                  heureDebutPM, 0, 0);
  }
  else {
    int dureeJour = TimeIntervalToClockInterval(24*60*60);
    arriveeClockTime =
      DateToClock(departEventDate.mday, departEventDate.month, departEventDate.year,
                  heureDebutAM, 0, 0)
      +dureeJour;
  }
  return arriveeClockTime;
};
```

Le convoyeur

Le convoyeur est créé avant le démarrage de la simulation avec N positions (voir l'initialisation du convoyeur dans la section « Fonctions dans le système technique ») puis initialisé au début de la simulation. Cette initialisation est un des effets de la mise en route du système au début de chaque plage de travail. A partir de la deuxième plage, elle part de la situation en fin de plage précédente, c'est-à-dire normalement sans positions vides.
Après le démarrage de la simulation, les alimentations successives du convoyeur sont provoquées (exclusivement) par les opérations de création d'un pack et de mise au rebut d'une brique.
L'effet de l'alimentation est modifié par un moniteur posé sur les loges du rebut : la probabilité de générer une petite (resp. grande) brique est augmentée s'il y a trop de petites (resp. grandes) briques au rebut : ainsi augmente la probabilité de constituer des packs de petites (resp. grandes) briques.

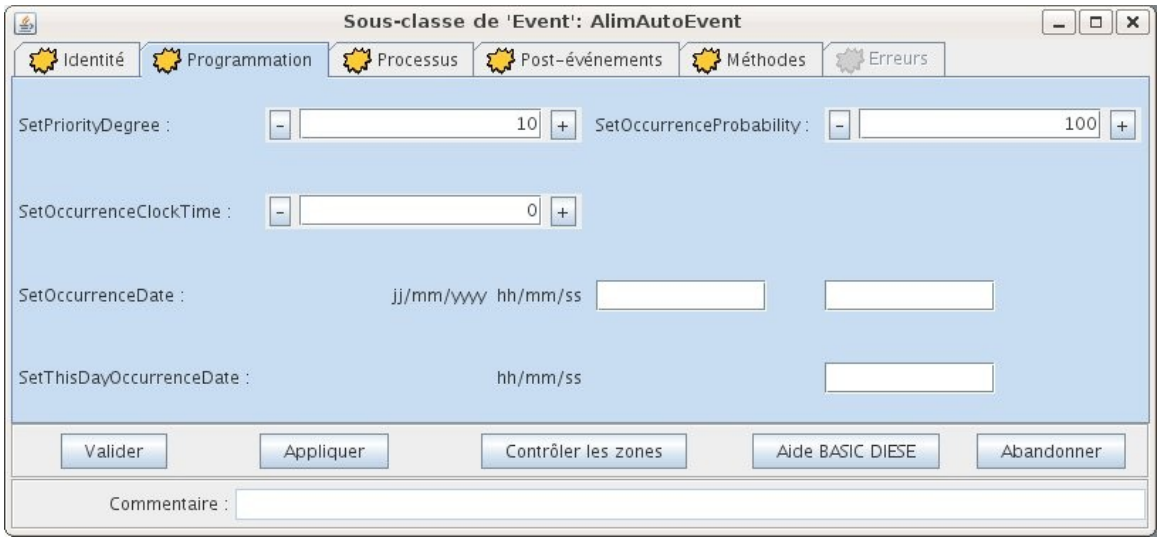
3.3.2) Dynamique du système opérant

La simulation est opérée sur une période supposée être organisée en plages de travail, au nombre de deux par jour. Les périodes de disponibilité (travail) et d'indisponibilité (pause) de l'ouvrier sont réglées par un paramètre du système qui fixe la durée de la plage de travail, et par deux autres paramètres qui spécifient l'heure de démarrage de chaque plage.
Pour provoquer l'enchainement de l'alternance « arrivée/départ », on a donné … :

- … à l'événement d'arrivée un caractère autogénérable : la prochaine occurrence est un départ programmé à l'horaire de démarrage de la plage qui démarre augmenté de la durée de la plage.
- … à l'événement de départ un caractère autogénérable : la prochaine occurrence est une arrivée programmée normativement (par exemple, à 14h le jour même ou à 8h le lendemain.

Le premier événement (une arrivée) est spécifié dans le fichier argument .str.
On note l'exploitation de certains services de gestion du temps simulé, prédéfinis dans DIESE : DateToClock, ClockToDate (traduction entre la valeur de l'horloge du temps simulé et la date calendaire simulée), TimeIntervalToClockInterval, qui traduit un nombre de secondes en un nombre d'unités de temps simulé.

```
// ------------------------------------
// on programme l'interprétation du plan
// ------------------------------------
+ V "updateSituationEvent" maj
    DATE_OCCUR = 0;
    MIN_DELAY_NEXT = 1;
    MAX_DELAY_NEXT = 1;
;
```



3.3.3) Dynamique du système décisionnel

L'examen du plan

L'examen du plan d'activités débute en t=*0*, comme le spécifie 'DATE_OCCUR=*0*' de l'événement prédéfini UpdateSituationEvent, dans le fichier argument .str.
L'itération sur les plages de travail est immédiatement ouverte, mais l'activité itérée (la première plage) reste en attente d'ouverture, ainsi que son premier élément (la mise en route). Toutes les autres activités (agrégées et primitives) sont « dormantes ». La mise en route sera ouverte dès que l'ouvrier sera arrivé.
L'examen du plan est renouvelé selon le rythme spécifié par MIN_DELAY_NEXT et MAX_DELAY_NEXT dans le fichier argument .str. Le délai avant le nouvel examen est tiré aléatoirement (loi uniforme) dans l'intervalle fermé entre les deux valeurs. Un délai constant est donc spécifié par deux valeurs égales, et il n'y a pas de tirage aléatoire dans ce cas.
Par construction du moteur de simulation, l'examen du plan s'arrête quand toutes les activités sont fermées ou annulées, ou bien lorsqu'on atteint la date de fin de la simulation (paramètre de la simulation, fichier argument .sim). On a expliqué plus haut (section 'La répétition des plages de travail') pourquoi la fin de la simulation interviendra toujours avant que toutes les activités ne soient fermées.

La mise en œuvre des opérations

Par conception du moteur de simulation du pilotage, un examen du plan, supposé instantané, est suivi, mais dans le même instant, par l'établissement, lui aussi supposé instantané, d'un ensemble de jeux d'instructions à mettre en œuvre.
Lorsqu'il n'y a qu'un seul jeu, les opérations sous-jacentes sont mises en œuvre ensemble, chacune avec les ressources qu'elle requiert, prouvées disponibles par le moteur. Dans notre atelier, les jeux sont toujours réduits à une seule activité, sauf lorsqu'une alimentation manuelle est nécessaire, auquel cas elle est associée à une création de pack, dans un jeu provoqué par l'opérateur de conjonction. Il se trouve que leurs mises en œuvre ne sont pas concomitantes, à la fois parce que la priorité de l'alimentation est plus forte et qu'elle est ponctuelle. Si les deux activités étaient non-ponctuelles, alors on verrait le degré de progression de leurs opérations augmenter « en parallèle ».
Quant à la situation où plusieurs jeux d'activités sont candidats à exécution, elle est exclue de notre application, puisque notre disjonction est exclusive, et qu'il n'y a jamais plusieurs manières alternatives d'allouer les ressources.
Trois priorités attribuées par défaut sont surchargées dans l'application : celle de l'opération d'alimentation manuelle (VerserCubi), on vient de l'évoquer, et celles, liées, des événements gérant l'alimentation automatique et l'arrivée de jus de fruit. Les valeurs respectives de 10 et 0 contraignent l'alimentation à suivre l'arrivée de jus.

```
// System parameters
// ----------------------------------
   // ...
   “RANDOM_GENERATOR” “seed”    <- 15833 “”; // 0 “”;
   "CONVOYEUR"  "probaPetite"    <- 0.50 "";
   // ...
```

```
INITIALISATION SIMULATION
    // ...
   INITRAND <pi><”RANDOM_GENERATOR” “seed”>;
   //IN_DIR "/home/durand/APPLIS_DIESE/KBS/cdiese/PackMan_SD/in/";
   IN_DIR "../in/";
   //OUT_DIR "/home/durand/APPLIS_DIESE/KBS/cdiese/PackMan_SD/out/";
   OUT_DIR "../out/";
;
```

```
// SAVE DESCRIPTOR
// -------------------------------
SAVE DESCRIPTOR "magasinExpedition" END  "production2.txt"     APPEND CLOCK
       "productionMaxi" 0 "productionMini" 0;
//SAVE DESCRIPTOR "cuve"         ALL "niveauCuve2_jhm.txt" NEW DATE_JHM
//      "volumeContenu"  0;
SAVE DESCRIPTOR "cuve"          ALL  "niveauCuve2_clock.txt"  NEW CLOCK
       "volumeContenu"  0;
```

4) Utilisation du simulateur

On souhaite analyser l'évolution du niveau de la cuve et prédire le volume de jus mis en briques.

Notre modèle ne contient qu'un seul élément aléatoire : la composition de la série de briques, qui dépend de deux choses :
- les probabilités (complémentaires à 1) de tirage d'une petite et d'une grande : c'est le paramètre d'identifiants « CONVOYEUR » et « probaPetite » (fichier des paramètres du système .par).
- la valeur donnée à la « graine » du générateur de nombres aléatoires, dans le fichier des paramètres de la simulation (fichier .sim). Si le paramètre du système (fichier .par) d'identifiants « RANDOM_GENERATOR » et « seed » a la valeur *0*, la série des nombres générés sera différente à chaque simulation (parce que le moteur remplace *0* par le numéro du 'processus système' que constitue l'exécution de la simulation). Sinon, elle sera toujours la même, ainsi que la série de briques sur le convoyeur. La seconde option est utile pour comparer des situations dans un contexte déterministe. Noter au passage qu'une initialisation du générateur faite, non dans le fichier .sim mais plus tard dans le fichier des directives (fichier .dir) est possible mais ne provoque pas la même dynamique. En effet, la série de briques générées dans la lecture du fichier .str n'est pas la même selon que la graine du générateur de nombres a été établie explicitement (dans le .sim, c'est-à-dire antérieurement) ou bien par défaut (si l'initialisation explicite est reportée dans le .dir).

Un autre élément aléatoire de la description informelle de l'atelier a été rendu déterministe : les quantités de jus de fruit arrivant en amont de la cuve. Les valeurs sont consignées dans un fichier de données présenté avec la fonction d'arrivée périodique de jus de fruit (section 3.2). On a notamment vu que le répertoire dans lequel le fichier est recherché est celui spécifié dans le fichier des paramètres de la simulation (.sim) sous l'étiquette 'IN_DIR'.

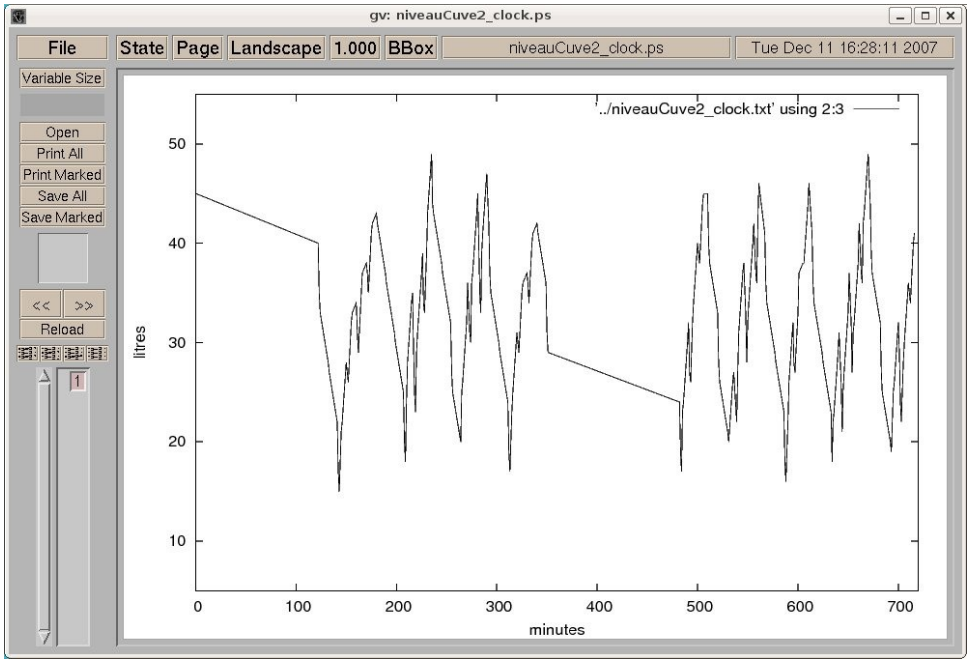
Le fichier des spécifications de sortie (.osp) demande une trace sélective d'exécution. Le moteur de simulation impose que le répertoire dans lequel les fichiers sont créés soit celui spécifié dans le fichier des paramètres de la simulation sous l'étiquette 'OUT_DIR'. Les requêtes portent sur l'évolution du niveau de la cuve (on peut choisir le format de date) et sur la valeur finale de la production. Pour celle-ci, on va accumuler les valeurs calculées par les exécutions successives.

Les fichiers de la structure du système et des directives de simulation ont été commentés avec les choix de conception du simulateur.

On va utiliser deux jeux de fichiers en arguments du simulateur. Ils ne diffèrent que sur un seul point (dans le fichier .par) : le volume des petites et des grandes briques, 1 et 2 litres, respectivement, pour un des jeux et 2 et 4 litres, respectivement, pour l'autre jeu. Tous les fichiers d'un jeu portent un nom de préfixe identique, qui identifie le jeu. Les deux préfixes sont sim1 et sim2. Un jeu est aussi dénommé simulation, notamment dans l'utilisation en mode interactif.

Dans le mode interactif, on utilise l'interface MI_Diese pour préparer les jeux de fichiers et pour lancer les exécutions. Le mode 'commande' consiste, pour l'utilisateur, à taper dans une invite de commandes de son système d'exploitation une ligne interprétable par le processeur de commandes, toujours composée du nom du programme exécutable suivi par les arguments. Les fichiers requis par l'exécution doivent avoir été préparés à l'aide d'éditeurs disponibles dans le système.

Les programmes exécutés en mode 'commande' et interactif sont respectivement `main` et `mainIhm`. Ils sont générés par des items dédiés du menu « Génération' de Solfege.




```
#magasinExpedition  clock    product product
leMagasin           1080     300     150
```

```
void App_ParseMainArguments (int argc, char* argv[])
{
  if (argc < 6) {
    printf("Usage : main system_parameters simulation_parameters
                        output_specifications structure directives");
    printf("        [-m(emory)] [-v(erbose)] [-t(trace)] [-f(trace)]\n");
    exit(0);
  };
  for (int i=6;i<argc;i++) {
    if (!strcmp(argv[i], "-v")) gTraceActionMode  = TRACE_ACTION_ON;
    if (!strcmp(argv[i], "-m")) gTraceNewDelMode  = TRACE_NEWDEL_ON;
    if (!strcmp(argv[i], "-t")) gMyTrace          = TRUE;
    if (!strcmp(argv[i], "-f")) gMyVerif          = TRUE;
  };
}
```

```
bool conditionnerPack_stateTransitionProcedure_Body(EntityMethod* pM)
{
  //...
  Entity* pPack = pM->GetEntityArgValue(OPERATION_TARGET);
  if(gMyTrace) printf("\nt = %d conditionnement '%s'",
                      pCurrentSim->Clock(), pPack->InstanceName());
  //...
};
```

```
void alimentationAutomatiqueProcess_initializeProcess_Body(ProcessMethod* pM)
{
  // ...
  if(etatRobinet == FERME) {
    printf("\n!!! t = %d Le robinet d'alimentation est ferme, de maniere inattendue !!!",
                    pCurrentSim->Clock());
    exit(0);
  }
  else
    if(gMyVerif) printf("\n!!! t = %d Le robinet d'alimentation est bien ouvert, comme
attendu !!!",                         pCurrentSim->Clock());
};
```

4.1) Mode commande

Les fichiers arguments sont dans `in`, répertoire frère du répertoire de travail. La séquence suivante :

```
cd ~/APPLIS_DIESE/KBS/cdiese/PackMan_SD/exec
main ../in/sim2.par ../in/sim2.sim ../in/sim2.osp ../in/sim2.str ../in/sim2.dir
```

génère une dynamique du volume et une production finale de jus qui peuvent être celles représentées ci-contre avec un script gnuplot, à partir des fichiers de sortie produits. Noter que les libellés des descripteurs sont tronqués à 7 caractères.

La commande d'exécution peut comporter des arguments additionnels, au-delà des 5 noms des fichiers d'entrée. Par exemple :

```
main ../in/sim2.par    …      ../in/sim2.dir –t
```

Cette possibilité est régie par une fonction globale prédéfinie, App_ParseMainArguments, dont le développeur peut surcharger le corps. Dans notre application, on y a prévu un message d'aide pour le cas où la commande serait mal formée (pas ou pas assez d'arguments). Et on a prévu de donner une valeur à quelques variables globales : deux sont prédéfinies dans DIESE (*gTraceActionMode* et *gTraceNewDelMode*), deux sont définies dans l'application (*gMyTrace* et *gMyVerif*). Si on place des arguments tels que '`-v`' ou '`-t`' après les 5 arguments-fichiers, les variables correspondantes passent de leur valeur *faux* (attribuée par défaut) à la valeur *vrai*. Le moteur de simulation comporte des instructions conditionnées aux variables prédéfinies (voir la documentation de DIESE), et le code de l'application, lui, réagit aux variables spécifiques. Il s'agit d'instructions qui mettent en oeuvre deux niveaux de trace, activés par la présence des arguments '-t' et '-f' respectivement. On en donne deux exemples ci-contre.

Noter enfin qu'on peut lancer en mode 'commande' l'exécutable `mainIhm`, celui qui est compilé pour être lancé, normalement, par l'interface graphique Mi_Diese (voir la section suivante). Cette procédure exceptionnelle n'est motivée que par un besoin de déboguage dans une session MI_Diese.

```
mainIhm ../in/sim2.par    …      ../in/sim2.dir
```

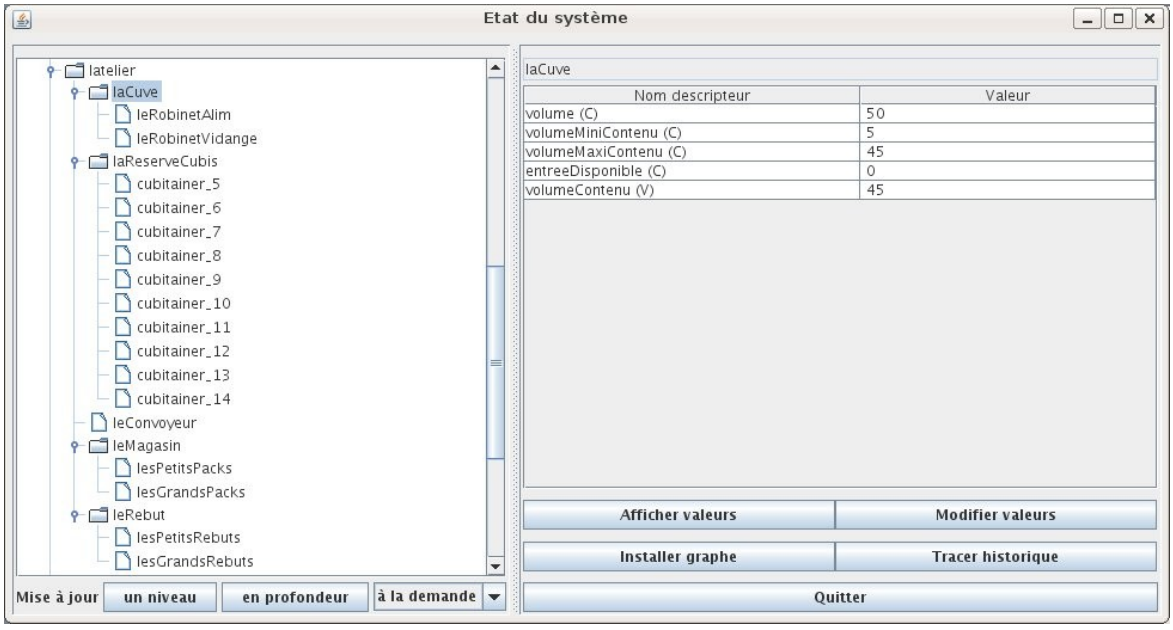
Le code du simulateur se met en attente de messages (ceux qui lui sont normalement transmis en réaction de la manipulation de l'interface). Le message qui permet de lancer la simulation est le suivant :

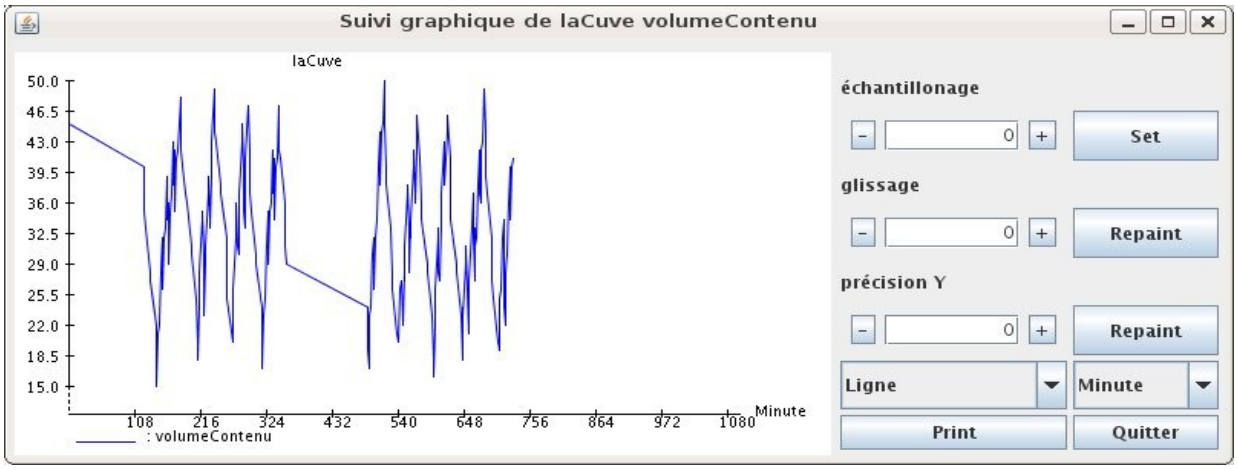
```
EXEC CONTINU 0
```

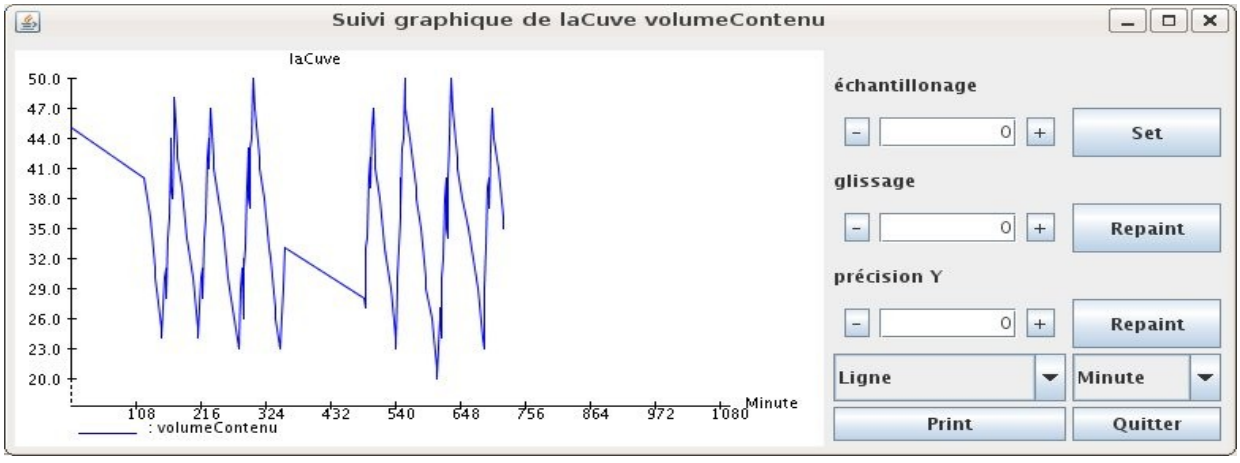
Celui sui permet de clore l'exécution en fin de simulation et de revenir en mode 'commande' est le suivant :

```
STOP
```

```
INITIALISATION SIMULATION
    // ...
   IN_DIR "/home/durand/APPLIS_DIESE/KBS/cdiese/PackMan_SD/in/";
   //IN_DIR "../in/";
   OUT_DIR "/home/durand/APPLIS_DIESE/KBS/cdiese/PackMan_SD/out/";
   //OUT_DIR "../out/";
;
```







4.2) Mode interactif : MI_Diese

On utilise les mêmes fichiers arguments que dans le mode 'commande'. Se rappeler que l'interface peut être lancée à partir de n'importe quel répertoire. En d'autres termes, la séquence de lancement pourra être :

```
cd ~/LIB_DIESE/DIESE.4.4/midiese
java –jar midiese.jar ~/APPLIS_DIESE/SIM/cdiese/PackMan_SD/sim2.s1 &
```

ou bien par exemple :

```
cd ~/APPLIS_DIESE/KBS/cdiese/PackMan_SD/exec
java –jar ~/LIB_DIESE/DIESE.4.4/midiese/midiese.jar ~/APPLIS_DIESE/SIM/… &
```

ou encore (en ne précisant pas la simulation à charger) :

```
java –jar midiese.jar &
```

En conséquence, on doit penser à ajuster le chemin des répertoires dans les fichiers arguments. Dans le fichier des paramètres de la simulation (.sim), les chemins relatifs `../in` et `../out` ne correspondent plus forcément à l'emplacement réel des fichiers.

Il existe plusieurs modes d'utilisation interactive : le lancement immédiat ou différé sur une simulation isolée ou lancement différé sur un lot de simulations. Se reporter à la documentation de MI_Diese pour lancer les exécutions.

Les lancements différés et immédiats sur simulations isolées doivent produire les mêmes résultats (aux aspects stochastiques près, notamment ceux liés à l'initialisation du générateur de nombres aléatoires à partir du numéro du processus système).

Quant aux simulations par lots, on rappelle qu'il en existe de deux types : les plans de simulations (lot 'Plan') et les lots fondés sur une gamme de valeurs pour des paramètres du système (lot 'Diff').

Un lot 'Plan' est un enchaînement de simulations isolées, chacune ayant ses propres fichiers arguments, et chacune répétées un certain nombre de fois. Un lot typique consiste à lancer une seule et même simulation un très grand nombre de fois (pour étude statistique dans un contexte aléatoire).

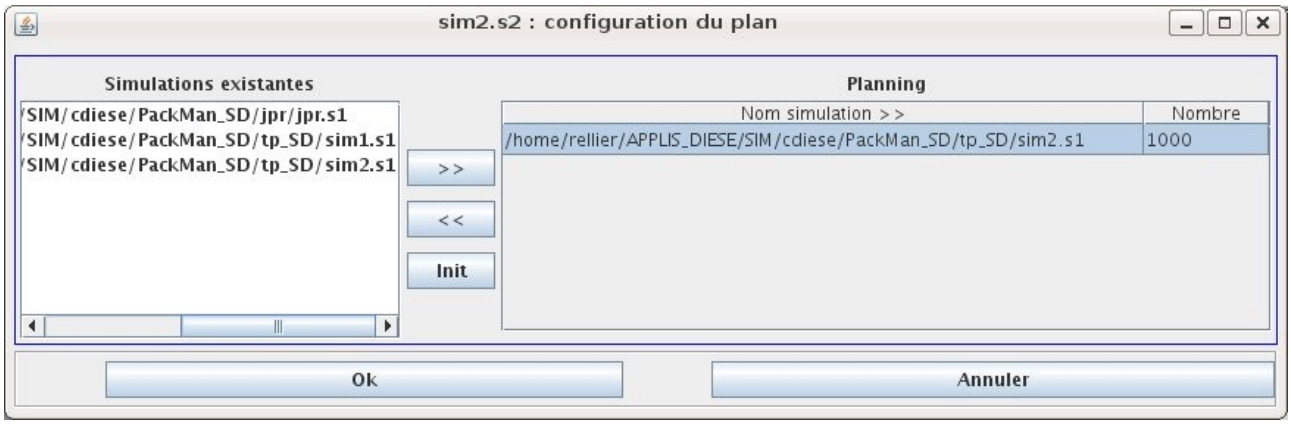
Un lot 'Diff' (appelé aussi « delta ») est construit pour observer le comportement du système avec des ensembles de paramètres différents (c'est-à-dire des simulations différentes) chacun dérivant du précédent en changeant la valeur d'un ou de plusieurs paramètres du système.

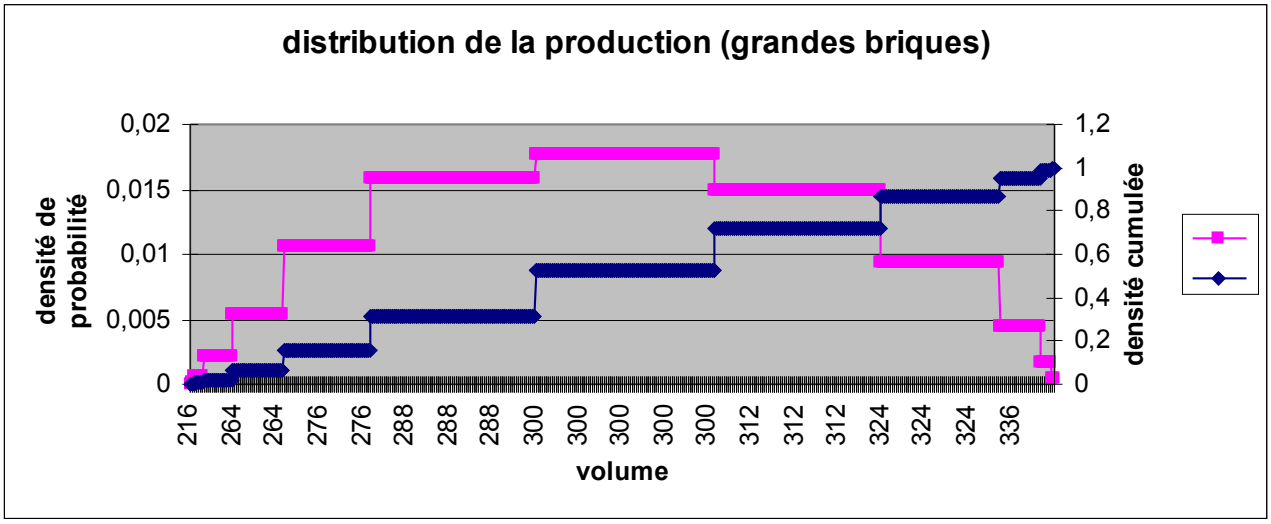
4.2.1) Lancement immédiat d'une simulation isolée

Après l'initialisation de la simulation (bouton 'Init'), on peut obtenir la visualisation ci-contre de l'état du système, avec sur la partie droite les valeurs des descripteurs de la cuve. On a demandé sur cette fenêtre (bouton 'Installer graphe', après sélection de la ligne 'volumeContenu') les suivis graphiques du niveau de la cuve, reproduits ci-contre pour deux simulations différentes, l'une avec la simulation sim2 (briques de 2 et 4 litres, en haut) et l'autre avec la simulation sim1 (briques de 1 et 2 litres, en bas).

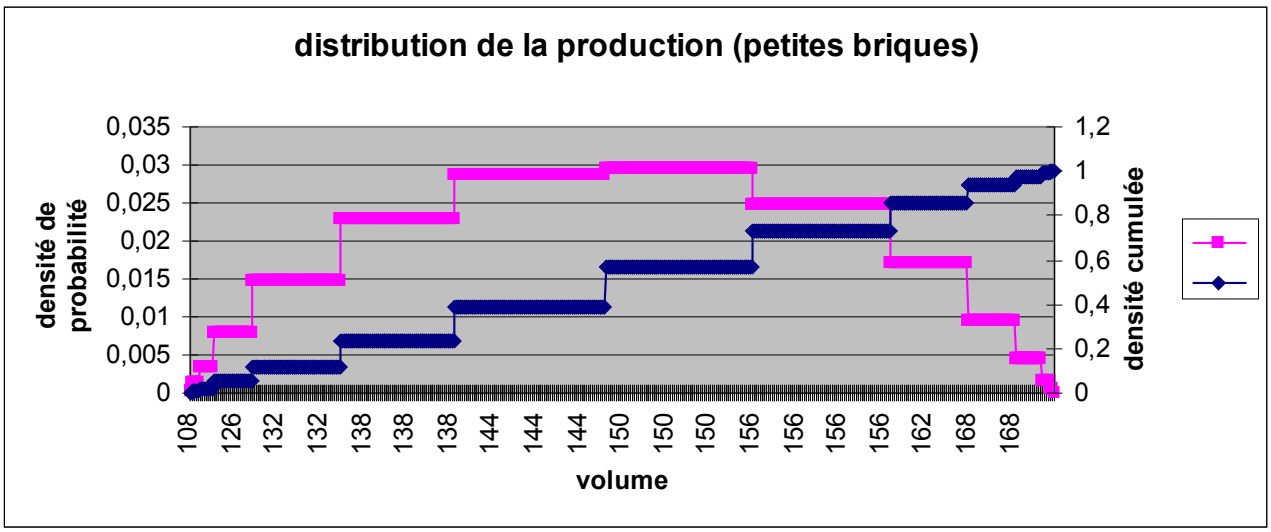
La dynamique est bien entendu la même qu'en mode 'commande'. On peut le vérifier d'une part sur le fichier de sortie relatif à la production finale (voir le fichier .osp), et d'autre part sur les fichiers de trace obtenus avec les deux exécutables `main` et `mainIhm` lancés en mode 'commande' avec l'argument additionnel '`-t`'.

On observe cependant quelques différences entre les tracés obtenus par gnuplot (après `main` en mode 'commande') et la fonction graphique de MI_Diese : par exemple, la valeur de volume *50* atteinte en la minute *510* n'est pas rapportée par gnuplot. C'est parce qu'au même instant, mais cependant « après », un remplissage de brique ramène le niveau à 45, et que, pour un instant donné, le moteur ne sauvegarde sur fichier que la dernière valeur affectée. La fonction graphique de MI_Diese, elle, « capte » tous les changements de valeurs.





moyenne : 298,716
écarts-type 22,5955603



moyenne : 147,816
écarts-type 13,3646603

4.2.2) Lancement différé d'un lot de simulation de type 'Plan'

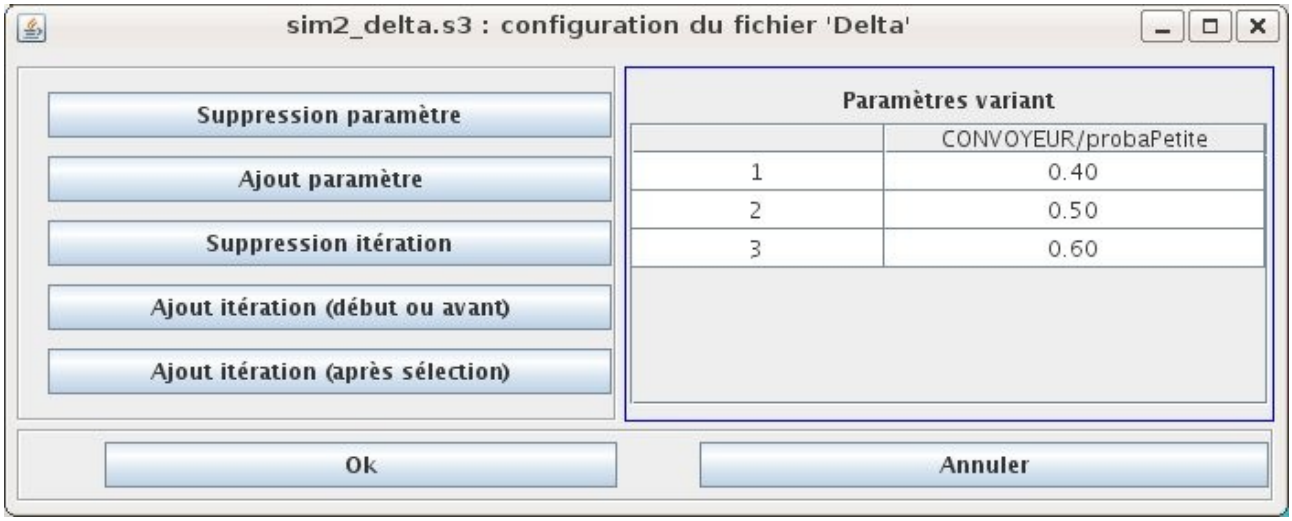
On a constitué un plan de simulation qui consiste à exécuter 1000 fois la simulation sim2. On veut en effet connaître la distribution de la production résultant du caractère aléatoire de la taille des briques qui arrivent sur le convoyeur.
La requête d'exécution différée du menu 'Run' (item 'Lancement différé') a fait connaître au système d'exploitation qu'il devra exécuter une certaine tâche à l'heure spécifiée. Cette tâche a été définie par MI_Diese dans un petit programme qui consiste à enchaîner 1000 fois la simulation sim2 en mode 'commande' (la commande étant bien entendu lancée par le système d'exploitation et non l'utilisateur). Ce petit programme s'auto-détruit en fin de tâche.

On s'est assuré que le simulateur possédait bien son caractère stochastique, en donnant la valeur *0* au paramètre d'identifiants « RANDOM_GENERATOR » et « seed » (fichier .par).

Les fichiers produits sont les suivants :

- dans le répertoire des simulations, un fichier de trace des exécutions, dont le nom est composé du préfixe *trace*, puis du nom de la simulation (ici *sim2*, puis enfin de l'heure de lancement de la tâche. Ce fichier contient ce qui est envoyé à l'écran quand on travaille en mode 'commande' (par exemple la durée de la simulation).
- dans le répertoire des fichiers de sortie (voir le fichier des paramètres de la simulation .sim) les fichiers demandés dans le fichier des spécifications de sortie (.osp). On se rappelle que pour la production de jus de fruit, on a demandé l'accumulation des valeurs calculées par les exécutions successives.

Pour visualiser les distributions des volumes produits, on a choisi un graphique sous le tableur Excel (Microsoft ®). On a donc importé dans une feuille du tableur le fichier production2.txt (après lui avoir ôté les lignes d'entête pour ne laisser subsister que les valeurs calculées), puis exploité la fonction graphique du tableur. On a obtenu les graphiques ci-contre. On remarque que les valeurs de production sont dans un ensemble discret restreint : ceci est dû, d'une part à la durée courte de la simulation (un jour seulement) et d'autre part au caractère discret de la production (on produit le jus des petites briques 6 litres par 6 litres et celui des grandes 12 litres par 12 litres). Dans chaque graphique, il y a 1000 points *x*.[*y1 y2*], un pour chaque simulation. La largeur des segments horizontaux correspond au nombre de simulations qui on résulté en la valeur *x* du volume de la production.




```
#magasinExpedition      clock   product product
leMagasin               1080    348     114

#magasinExpedition      clock   product product
leMagasin               1080    300     150

#magasinExpedition      clock   product product
leMagasin               1080    216     192
```

4.2.3) Lancement différé d'un lot de simulation de type 'Diff'

On souhaite ici connaître l'effet, sur la production totale de jus de fruit, d'une variation de la probabilité de générer une petite brique. On a donc créé un lot de simulation qui va enchaîner 3 exécutions du simulateur avec des valeurs différentes du paramètre du système dont les identifiants sont « CONVOYEUR » et « probaPetite », respectivement *0.40*, *0.50* et *0.60*.
La requête d'exécution différée du menu 'Run' (item 'Lancement différé') a fait connaître au système d'exploitation qu'il devra exécuter une certaine tâche à l'heure spécifiée. Cette tâche a été définie par MI_Diese dans un petit programme qui consiste à enchaîner 3 simulations sim2 en mode 'commande' avec le même jeu de fichiers arguments, à l'exception du fichier des paramètres du système. Lors de chacune des 3 commandes, c'est une copie du fichier .par de base qui est exploitée, seulement modifiée sur la valeur du paramètre faisant l'objet de la variation. (les commandes sont lancées par le système d'exploitation et non l'utilisateur). Ce petit programme s'auto-détruit en fin de tâche.

On s'est assuré que le simulateur avait un caractère déterministe, en donnant une valeur différente de *0* au paramètre d'identifiants « RANDOM_GENERATOR » et « seed » (fichier .par).

Les fichiers produits sont les suivants :

- dans le répertoire des simulations, un fichier de trace des exécutions, dont le nom est composé du préfixe *trace*, puis du nom de la simulation (ici *sim2_copyDelta*, puis enfin de l'heure de lancement de la tâche. Ce fichier contient ce qui est envoyé à l'écran quand on travaille en mode 'commande' (par exemple la durée de la simulation).
- dans le répertoire des fichiers de sortie (voir le fichier des paramètres de la simulation .sim) les fichiers demandés dans le fichier des spécifications de sortie (.osp). On se rappelle que pour la production de jus de fruit, on a demandé l'accumulation des valeurs calculées par les exécutions successives. On y retrouve donc le résultat, visualisé ci-contre, des 3 exécutions correspondant aux 3 valeurs du paramètre variant. La seconde exécution (valeur *0.50* pour la probabilité de générer une petite Brique) produit bien le même résultat que la simulation isolée avec la même valeur de ce paramètre (et à graine du générateur de nombres aléatoires identique).

Index

Annexe 1 : spécifications structurelles

```
############
#  Entity  #
############

Entity AtelierRemplissage {
      ClassSymbol               : ATELIER_REMPLISSAGE
      ClassName                 : "atelierRemplissage"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : O_ENTITY
      AddComponentClassId       : CUVE
      AddComponentClassId       : RESERVE_CUBIS
      AddComponentClassId       : CONVOYEUR
      AddComponentClassId       : MAGASIN_EXPEDITION
      AddComponentClassId       : REBUT

}
#=============================================
Entity Brique {
      ClassSymbol               : BRIQUE
      ClassName                 : "brique"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : ELEMENT_DE_CONVOYEUR

      AddConstantDescriptor     : Volume   ---
      AddVariableDescriptor     : VolumeContenu   ---
      AddVariableDescriptor     : EtatOuverture   ---
}
#=============================================
Entity Convoyeur {
      ClassSymbol               : CONVOYEUR
      ClassName                 : "convoyeur"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : O_ENTITY

      AddConstantDescriptor     : ConvoyeurLongueur   ---
      AddConstantDescriptor     : ConvoyeurGenerateur   ---
      AddConstantDescriptor     : ConvoyeurPlateforme   ---
      AddConstantDescriptor     : PositionsBriquesAPaqueter   ---
      AddMethod                 : InitialisationConvoyeur
      AddMethod                 : AlimentationConvoyeur
      AddMethod                 : TassementConvoyeur
}
#=============================================
Entity Cubitainer {
      ClassSymbol               : CUBITAINER
      ClassName                 : "cubitainer"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : O_ENTITY

      AddConstantDescriptor     : Volume   ---
}
#=============================================
Entity Cuve {
      ClassSymbol               : CUVE
      ClassName                 : "cuve"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : O_ENTITY
      SetMonitorToDescriptor    : VolumeContenu     VolumeContenuCuveMonitor
      AddComponentClassId       : ROBINET_ALIM
      AddComponentClassId       : ROBINET_VIDANGE

      AddConstantDescriptor     : Volume   ---
      AddConstantDescriptor     : VolumeMiniContenu   ---
      AddConstantDescriptor     : VolumeMaxiContenu   ---
      AddConstantDescriptor     : EntreeDisponible   ---
      AddVariableDescriptor     : VolumeContenu   ---
}
#=============================================
Entity ElementDeConvoyeur {
      ClassSymbol               : ELEMENT_DE_CONVOYEUR
      ClassName                 : "elementDeConvoyeur"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : O_ENTITY

}
#=============================================
Entity GenerateurDeBriques {
```

```
      ClassSymbol               : GENERATEUR_DE_BRIQUES
      ClassName                 : "generateurDeBriques"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : O_ENTITY

      AddConstantDescriptor     : ProbaPetiteBrique   ---
}
#=============================================
Entity Loge {
      ClassSymbol               : LOGE
      ClassName                 : "loge"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : O_ENTITY
      ElementClassId            : ELEMENT_DE_CONVOYEUR

      AddConstantDescriptor     : Contenance   ---
}
#=============================================
Entity LogeRebut {
      ClassSymbol               : LOGE_REBUT
      ClassName                 : "logeRebut"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : LOGE
      SetStructureMonitor       : LogeRebutMonitor

}
#=============================================
Entity Magasin {
      ClassSymbol               : MAGASIN
      ClassName                 : "magasin"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : O_ENTITY
      ElementClassId            : LOGE

}
#=============================================
Entity MagasinExpedition {
      ClassSymbol               : MAGASIN_EXPEDITION
      ClassName                 : "magasinExpedition"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : MAGASIN

      AddVariableDescriptor     : ProductionMini   ---
      AddVariableDescriptor     : ProductionMaxi   ---
}
#=============================================
Entity MaxiBrique {
      ClassSymbol               : MAXI_BRIQUE
      ClassName                 : "maxiBrique"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : BRIQUE

}
#=============================================
Entity MiniBrique {
      ClassSymbol               : MINI_BRIQUE
      ClassName                 : "miniBrique"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : BRIQUE

}
#=============================================
Entity Pack {
      ClassSymbol               : PACK
      ClassName                 : "pack"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : ELEMENT_DE_CONVOYEUR
      ElementClassId            : BRIQUE
      SetMonitorToDescriptor    : VolumeContenu     VolumeContenuPackMonitor

      AddVariableDescriptor     : VolumeContenu   ---
}
#=============================================
Entity PositionVide {
      ClassSymbol               : POSITION_VIDE
      ClassName                 : "positionVide"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : ELEMENT_DE_CONVOYEUR

}
#=============================================
Entity Rebut {
```

```
      ClassSymbol               : REBUT
      ClassName                 : "rebut"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : MAGASIN
      ElementClassId            : LOGE_REBUT

      AddConstantDescriptor     : DerniereProbaAugmentee   -1
}
#============================================
Entity ReserveCubis {
      ClassSymbol               : RESERVE_CUBIS
      ClassName                 : "reserveCubis"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : O_ENTITY
      ElementClassId            : CUBITAINER

}
#============================================
Entity Robinet {
      ClassSymbol               : ROBINET
      ClassName                 : "robinet"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : O_ENTITY

      AddConstantDescriptor     : Debit   ---
      AddVariableDescriptor     : EtatOuverture   ---
}
#============================================
Entity RobinetAlim {
      ClassSymbol               : ROBINET_ALIM
      ClassName                 : "robinetAlim"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : ROBINET

}
#============================================
Entity RobinetVidange {
      ClassSymbol               : ROBINET_VIDANGE
      ClassName                 : "robinetVidange"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : ROBINET

}
#============================================

##############
#  Activity  #
##############

PrimitiveActivity AlimentationManuelle {
      ClassSymbol               : ALIMENTATION_MANUELLE
      ClassName                 : "alimentationManuelle"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : PRIMITIVE_ACTIVITY

      OverloadedVarDesc value   : OperationClassId   VERSER_CUBI
      OverloadedMethod body     : OpeningPredicate   alimentationManuelle_openingPredicate_Body
      OverloadedMethod body     : MaxBegStatePredicate   alimentationManuelle_maxBegStatePredicate_Body
}
#============================================
PrimitiveActivity ArretProduction {
      ClassSymbol               : ARRET_PRODUCTION
      ClassName                 : "arretProduction"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : PRIMITIVE_ACTIVITY

      OverloadedVarDesc value   : OperationClassId   ARRETER_PRODUCTION
}
#============================================
PrimitiveActivity ConditionnementPack {
      ClassSymbol               : CONDITIONNEMENT_PACK
      ClassName                 : "conditionnementPack"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : PRIMITIVE_ACTIVITY

      OverloadedVarDesc value   : PerformerSpecAttribute   OUVRIER_AGISSANT
      OverloadedVarDesc value   : OperationClassId   CONDITIONNER_PACK
}
#============================================
PrimitiveActivity CreationPack {
      ClassSymbol               : CREATION_PACK
      ClassName                 : "creationPack"
      signatureConstructeur     : "()"
```

```
      instanceList              : "yes"
      SetParentClassId          : PRIMITIVE_ACTIVITY

      OverloadedVarDesc value   : OperationClassId   CREER_PACK
}
#============================================
PrimitiveActivity FermetureBriquesPack {
      ClassSymbol               : FERMETURE_BRIQUES_PACK
      ClassName                 : "fermetureBriquesPack"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : PRIMITIVE_ACTIVITY

      OverloadedVarDesc value   : OperationClassId   FERMER_BRIQUE
}
#============================================
TransferInformation InfoTransfertCuvePack {
      ClassSymbol               : INFO_TRANSFERT_CUVE_PACK
      ClassName                 : "infoTransfertCuvePack"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : TRANSFER_INFORMATION

      OverloadedConstDesc value : TransferQuantityId   VOLUME_CONTENU
      OverloadedMethod body     : SetUserTransferedFloatQuantity   infoTransfertCuvePack_setUserTransfered-
FloatQuantity_Body
}
#============================================
NonPrimitiveActivity IterationMagasinOuRebut {
      ClassSymbol               : ITERATION_MAGASIN_OU_REBUT
      ClassName                 : "iterationMagasinOuRebut"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : ACTIVITY_ITERATION

      OverloadedMethod body     : ClosingPredicate   iterationMagasinOuRebut_closingPredicate_Body
}
#============================================
PrimitiveActivity MiseAuRebut {
      ClassSymbol               : MISE_AU_REBUT
      ClassName                 : "miseAuRebut"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : PRIMITIVE_ACTIVITY

      OverloadedVarDesc value   : OperationClassId   METTRE_AU_REBUT
      OverloadedMethod body     : OpeningPredicate   miseAuRebut_openingPredicate_Body
}
#============================================
PrimitiveActivity MiseEnRoute {
      ClassSymbol               : MISE_EN_ROUTE
      ClassName                 : "miseEnRoute"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : PRIMITIVE_ACTIVITY

      OverloadedVarDesc value   : OperationClassId   METTRE_EN_ROUTE
      OverloadedVarDesc value   : MinBegDate   0
      OverloadedVarDesc value   : OperatedObjectSpecAttribute   ATELIER_MIS_EN_ROUTE
      OverloadedMethod body     : OpeningPredicate   miseEnRoute_openingPredicate_Body
}
#============================================
PrimitiveActivity RemplissagePack {
      ClassSymbol               : REMPLISSAGE_PACK
      ClassName                 : "remplissagePack"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : TRANSFER_ACTIVITY

      OverloadedVarDesc value   : OperationClassId   REMPLIR_BRIQUES
}
#============================================
NonPrimitiveActivity SequenceMiseEnMagasin {
      ClassSymbol               : SEQUENCE_MISE_EN_MAGASIN
      ClassName                 : "sequenceMiseEnMagasin"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : ACTIVITY_BEFORE

      OverloadedMethod body     : OpeningPredicate   sequenceMiseEnMagasin_openingPredicate_Body
}
#============================================
PrimitiveActivity StockagePack {
      ClassSymbol               : STOCKAGE_PACK
      ClassName                 : "stockagePack"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : PRIMITIVE_ACTIVITY

      OverloadedVarDesc value   : OperationClassId   STOCKER_PACK
}
```

```
#============================================

###############
#  Operation  #
###############

Operation ArreterProduction {
      ClassSymbol               : ARRETER_PRODUCTION
      ClassName                 : "arreterProduction"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : TIME_GRANULATED_OPERATION

      OverloadedConstDesc value : OperatedEntityId   ATELIER_REMPLISSAGE
      OverloadedConstDesc value : Step   1
      OverloadedConstDesc value : TimeSpeed   1
      OverloadedMethod body     : StateTransitionProcedure   arreterProduction_stateTransitionProcedure_Body
}
#============================================
Operation ConditionnerPack {
      ClassSymbol               : CONDITIONNER_PACK
      ClassName                 : "conditionnerPack"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : TIME_GRANULATED_OPERATION

      OverloadedConstDesc value : TimeSpeed   1
      OverloadedConstDesc value : Step   1
      OverloadedConstDesc value : OperatedEntityId   PACK
      OverloadedMethod body     : StateTransitionProcedure   conditionnerPack_stateTransitionProcedure_Body
}
#============================================
Operation CreerPack {
      ClassSymbol               : CREER_PACK
      ClassName                 : "creerPack"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : TIME_GRANULATED_OPERATION

      OverloadedConstDesc value : TimeSpeed   1
      OverloadedConstDesc value : Step   1
      OverloadedConstDesc value : OperatedEntityId   CONVOYEUR
      OverloadedVarDesc value   : RequiredResources   OUVRIER_OPERANT
      OverloadedMethod body     : StateTransitionProcedure   creerPack_stateTransitionProcedure_Body
}
#============================================
Operation FermerBrique {
      ClassSymbol               : FERMER_BRIQUE
      ClassName                 : "fermerBrique"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : UNIT_GRANULATED_OPERATION

      OverloadedConstDesc value : OperatedEntityId   BRIQUE
      OverloadedConstDesc value : Step   1
      OverloadedConstDesc value : UnitSpeed   3
      OverloadedMethod body     : StateTransitionProcedure   fermerBrique_stateTransitionProcedure_Body
}
#============================================
Operation MettreAuRebut {
      ClassSymbol               : METTRE_AU_REBUT
      ClassName                 : "mettreAuRebut"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : TIME_GRANULATED_OPERATION

      OverloadedConstDesc value : TimeSpeed   1
      OverloadedConstDesc value : Step   1
      OverloadedConstDesc value : OperatedEntityId   REBUT
      OverloadedMethod body     : StateTransitionProcedure   mettreAuRebut_stateTransitionProcedure_Body
}
#============================================
Operation MettreEnRoute {
      ClassSymbol               : METTRE_EN_ROUTE
      ClassName                 : "mettreEnRoute"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : TIME_GRANULATED_OPERATION

      OverloadedConstDesc value : TimeSpeed   1
      OverloadedConstDesc value : OperatedEntityId   ATELIER_REMPLISSAGE
      OverloadedConstDesc value : Step   1
      OverloadedMethod body     : StateTransitionProcedure   mettreEnRoute_stateTransitionProcedure_Body
}
#============================================
Operation RemplirBriques {
      ClassSymbol               : REMPLIR_BRIQUES
      ClassName                 : "remplirBriques"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : FLOAT_QUANTITY_GRANULATED_TRANSFER_OPERATION
```

```
      OverloadedConstDesc value : Step   1
      OverloadedConstDesc value : OperatedEntityId   CUVE
      OverloadedMethod body     : SetUserFloatQuantitySpeed   remplirBriques_setUserFloatQuantitySpeed_Body
}
#============================================
Operation StockerPack {
      ClassSymbol               : STOCKER_PACK
      ClassName                 : "stockerPack"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : TIME_GRANULATED_OPERATION

      OverloadedConstDesc value : TimeSpeed   1
      OverloadedConstDesc value : Step   1
      OverloadedConstDesc value : OperatedEntityId   CONVOYEUR
      OverloadedVarDesc value   : RequiredResources   OUVRIER_OPERANT
      OverloadedMethod body     : StateTransitionProcedure   stockerPack_stateTransitionProcedure_Body
}
#============================================
Operation VerserCubi {
      ClassSymbol               : VERSER_CUBI
      ClassName                 : "verserCubi"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : TIME_GRANULATED_OPERATION

      OverloadedConstDesc value : Step   1
      OverloadedConstDesc value : OperatedEntityId   RESERVE_CUBIS
      OverloadedConstDesc value : TimeSpeed   1.0
      OverloadedConstDesc value : PriorityDegree   20
      OverloadedMethod body     : StateTransitionProcedure   verserCubi_stateTransitionProcedure_Body
}
#============================================

##############
#  Resource  #
##############

SingleResource Ouvrier {
      ClassSymbol               : OUVRIER
      ClassName                 : "ouvrier"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : SINGLE_WORKER

      AddConstantDescriptor     : PretAPaqueter   ---
}
#============================================

############
#  System  #
############


###################
#  Specification  #
###################

OperatedObjectSpec AtelierMisEnRoute {
      ClassSymbol               : ATELIER_MIS_EN_ROUTE
      ClassName                 : "atelierMisEnRoute"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : OPERATED_OBJECT_SPECIFICATION

      OverloadedConstDesc value : SelectorFctId   ANYONE
      OverloadedConstDesc value : ObjectEntitySpecAttribute   lesAteliers
}
#============================================
PerformerSpec OuvrierAgissant {
      ClassSymbol               : OUVRIER_AGISSANT
      ClassName                 : "ouvrierAgissant"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : PERFORMER_SPECIFICATION

      OverloadedConstDesc value : SelectorFctId   ANYONE
      OverloadedConstDesc value : SelectorFctArg   1
      OverloadedVarDesc value   : EntitySpecReferenceClassId   OUVRIER
}
#============================================
OperationResourceSpec OuvrierOperant {
      ClassSymbol               : OUVRIER_OPERANT
      ClassName                 : "ouvrierOperant"
      signatureConstructeur     : "()"
      instanceList              : "yes"
      SetParentClassId          : OPERATION_RESOURCE_SPECIFICATION

      OverloadedConstDesc value : SelectorFctId   ANYONE
      OverloadedConstDesc value : SelectorFctArg   1
```

```
      OverloadedVarDesc value   : EntitySpecReferenceClassId   OUVRIER
}
#============================================

##############
#  Strategy  #
##############


###############
#  Condition  #
###############


################
#  Descriptor  #
################

Descriptor Contenance {
      ClassSymbol               : CONTENANCE
      ClassName                 : contenance
      ClassComment              : un nombre maximum d'elements d'un conteneur
      SubClass                  : CONSTANT
      Type                      : INT
      AddSubDomain              : [ 0 ; 32767 ]
}
#============================================
Descriptor ConvoyeurGenerateur {
      ClassSymbol               : CONVOYEUR_GENERATEUR
      ClassName                 : convoyeurGenerateur
      ClassComment              : pointeur sur le generateur de brique composant du convoyeur
      SubClass                  : CONSTANT
      Type                      : P_ENTITY
      SetDefaultValue           : GENERATEUR_DE_BRIQUES
      AddSubDomain              : GENERATEUR_DE_BRIQUES
}
#============================================
Descriptor ConvoyeurLongueur {
      ClassSymbol               : CONVOYEUR_LONGUEUR
      ClassName                 : convoyeurLongueur
      ClassComment              : nombre maximal de lots de briques alignes sur le convoyeur
      SubClass                  : CONSTANT
      Type                      : INT
      SetDefaultValue           : 10
}
#============================================
Descriptor ConvoyeurPlateforme {
      ClassSymbol               : CONVOYEUR_PLATEFORME
      ClassName                 : convoyeurPlateforme
      ClassComment              : les positions du convoyeur sont les elements d'un tableau d'entites
      SubClass                  : CONSTANT
      Type                      : ENTITY_TAB
      AddSubDomain              : ELEMENT_DE_CONVOYEUR
}
#============================================
Descriptor Debit {
      ClassSymbol               : DEBIT
      ClassName                 : debit
      ClassUnity                : litres/minute
      SubClass                  : CONSTANT
      Type                      : FLOAT
      DescValueSpec             : DebitDomaine
}
#============================================
Descriptor DerniereProbaAugmentee {
      ClassSymbol               : DERNIERE_PROBA_AUGMENTEE
      ClassName                 : derniereProbaAugmentee
      ClassComment              : 0 si on a touché probaPetite ; 1 si on a touché probaPetite ; -1 à l'init.
      SubClass                  : CONSTANT
      Type                      : INT
}
#============================================
Descriptor EntreeDisponible {
      ClassSymbol               : ENTREE_DISPONIBLE
      ClassName                 : entreeDisponible
      ClassComment              : quantite disponible en entree de cuve, introduite seulement si robinet alim ou-
vert
      ClassUnity                : litre
      SubClass                  : CONSTANT
      Type                      : FLOAT
}
#============================================
Descriptor EtatOuverture {
      ClassSymbol               : ETAT_OUVERTURE
      ClassName                 : etatOuverture
      ClassComment              : OUVERT=0 ou FERME=1 (enumeration)
      SubClass                  : VARIABLE
      Type                      : INT
      SetDefaultValue           : FERME
      AddSubDomain              : [ 0 ; 1 ]
}
```

```
#============================================
Descriptor PositionsBriquesAPaqueter {
      ClassSymbol               : POSITIONS_BRIQUES_A_PAQUETER
      ClassName                 : positionsBriquesAPaqueter
      ClassComment              : FloatTab en attendant IntTab !
      SubClass                  : CONSTANT
      Type                      : FLOAT_TAB
}
#============================================
Descriptor PretAPaqueter {
      ClassSymbol               : PRET_A_PAQUETER
      ClassName                 : pretAPaqueter
      ClassComment              : 1 si l'ouvrier a consate qu'un pack peut etre cree, 0 sinon
      SubClass                  : CONSTANT
      Type                      : INT
}
#============================================
Descriptor ProbaPetiteBrique {
      ClassSymbol               : PROBA_PETITE_BRIQUE
      ClassName                 : probaPetiteBrique
      ClassComment              : lors de la generation d'une brique, probalilite qu'elle soit petite
      SubClass                  : CONSTANT
      Type                      : FLOAT
      SetDefaultValue           : 0.5
      SetToDefaultValue         : TRUE
      AddSubDomain              : [ 0. ; 1. ]
}
#============================================
Descriptor ProductionMaxi {
      ClassSymbol               : PRODUCTION_MAXI
      ClassName                 : productionMaxi
      ClassComment              : le total de litres emmagasines dans la loge des maxi briques
      ClassUnity                : litres
      SubClass                  : VARIABLE
      Type                      : FLOAT
}
#============================================
Descriptor ProductionMini {
      ClassSymbol               : PRODUCTION_MINI
      ClassName                 : productionMini
      ClassComment              : le total de litres emmagasines dans la loge des mini briques
      ClassUnity                : litres
      SubClass                  : VARIABLE
      Type                      : FLOAT
}
#============================================
Descriptor Volume {
      ClassSymbol               : VOLUME
      ClassName                 : volume
      ClassUnity                : litre
      SubClass                  : CONSTANT
      Type                      : FLOAT
      SetDefaultValue           : 0.
      SetToDefaultValue         : TRUE
      AddSubDomain              : [ 0.001 ; 1000. ]
}
#============================================
Descriptor VolumeContenu {
      ClassSymbol               : VOLUME_CONTENU
      ClassName                 : volumeContenu
      ClassComment              : volume du contenu
      ClassUnity                : litre
      SubClass                  : VARIABLE
      Type                      : FLOAT
      SetDefaultValue           : 0.
      SetToDefaultValue         : TRUE
}
#============================================
Descriptor VolumeMaxiContenu {
      ClassSymbol               : VOLUME_MAXI_CONTENU
      ClassName                 : volumeMaxiContenu
      ClassComment              : seuil superieur pour le volume du contenu
      SubClass                  : CONSTANT
      Type                      : FLOAT
}
#============================================
Descriptor VolumeMiniContenu {
      ClassSymbol               : VOLUME_MINI_CONTENU
      ClassName                 : volumeMiniContenu
      ClassComment              : seuil inferieur pour le volume du contenu
      ClassUnity                : litre
      SubClass                  : CONSTANT
      Type                      : FLOAT
}
#============================================

#############
#  Monitor  #
#############

StructureMonitor LogeRebutMonitor {
```

```
      ClassName                 : logeRebutMonitor
      AssignAddElementMethod    : logeRebut_whenAddElement_Body
}
#============================================
DescValueMonitor VolumeContenuCuveMonitor {
      ClassName                 : volumeContenuCuveMonitor
      type                      : FLOAT
      AssignWhenSetMethod       : volumeContenuCuve_whenSetFloat_Body
}
#============================================
DescValueMonitor VolumeContenuPackMonitor {
      ClassName                 : volumeContenuPackMonitor
      type                      : FLOAT
      AssignWhenSetMethod       : volumeContenuPack_whenSetFloat_Body
}
#============================================

##############
#  Argument  #
##############

Argument Rang {
      ClassName                 : rang
      ClassSymbol               : RANG
      Type                      : INT
}
#============================================

#############
#  Method  #
#############

Method AlimentationConvoyeur {
      ClassSymbol               : ALIMENTATION_CONVOYEUR
      ClassName                 : alimentationConvoyeur
      ObjectClass               : O_ENTITY
      AssignBody                : alimentationConvoyeur_Body
      ClassShortComment         : renvoie la brique generee et placee en derniere position
      Type                      : P_ENTITY
      pEntityType               : Brique*
}
#============================================
Method InitialisationConvoyeur {
      ClassSymbol               : INITIALISATION_CONVOYEUR
      ClassName                 : initialisationConvoyeur
      ObjectClass               : O_ENTITY
      AssignBody                : initialisationConvoyeur_Body
      ClassShortComment         : remplace les positions vides pres du generateur par des briques
      Type                      : VOID
}
#============================================
Method TassementConvoyeur {
      ClassSymbol               : TASSEMENT_CONVOYEUR
      ClassName                 : tassementConvoyeur
      ObjectClass               : O_ENTITY
      AssignBody                : tassementConvoyeur_Body
      ClassShortComment         : le rang des entites a partir du rang <arg> est dimunue de 1
      Type                      : VOID
      AddArgument               : Rang
}
#============================================

##################
#  DescValueSpec  #
##################

DescValueSpec DebitDomaine {
      ClassName                 : debitDomaine
      Type                      : FLOAT
      Category                  : DOMAIN
      Operator                  : IN
      AddSubDomain              : [ 1. ; 10. ]
}
#============================================

################
#  EntitySpec  #
################

EntitySpec LePackDeTete {
      ClassName                 : lePackDeTete
      UserClassId               : PACK
      QuantificatorId           : NFIRST
      Mode                      : RETURN_INSTANCES
      Effectif                  : 1
      AssignEntityListInstantiator : lePackDeTete_entityListInstantiator_Body
}
#============================================
EntitySpec LesAteliers {
      ClassName                 : lesAteliers
      UserClassId               : ATELIER_REMPLISSAGE
```

```
      QuantificatorId           : NFIRST
      Mode                      : RETURN_INSTANCES
      Effectif                  : 1
      AssignSelectorPredicate   : lesAteliers_selectorPredicate_Body
}
#============================================

#############
#  Process  #
#############

ContinuousProcess AlimentationAutomatiqueProcess {
      ClassSymbol               : ALIMENTATION_AUTOMATIQUE_PROCESS
      ClassName                 : alimentationAutomatiqueProcess
      ParentClassId             : ContinuousProcess
      ProcessedEntityClassId    : CUVE
      FirstEntity               : true
      SetPrecondition           : alimentationAutomatiqueProcess_processPrecondition_Body
      SetPostponementDelay      : alimentationAutomatiqueProcess_processPostponementDelay_Body
      PostponementDelay         : 1
      Step                      : 1
      PostInitialisationEvent   : 0   0
      SetInitialize             : alimentationAutomatiqueProcess_initializeProcess_Body
      SetGoOneStepForward       : alimentationAutomatiqueProcess_goOneStepForwardProcess_Body
}
#============================================
ReadSequentialDataFileProcess FluxEntreeCuveLectureProcess {
      ClassSymbol               : FLUX_ENTREE_CUVE_LECTURE_PROCESS
      ClassName                 : fluxEntreeCuveLectureProcess
      ParentClassId             : ReadSequentialDataFileProcess
      ProcessedEntityClassId    : CUVE
      FirstEntity               : true
      PostponementDelay         : 1
      Step                      : 1
      PostInitialisationEvent   : 0   0
      SetInitialize             : fluxEntreeCuveLectureProcess_initializeProcess_Body
      SetPostRead               : fluxEntreeCuveLectureProcess_postReadProcess_Body
      FileClassId               : FLUX_ENTREE_CUVE
      SetStop                   : fluxEntreeCuveLectureProcess_stopProcess_Body
}
#============================================

############
#  Event  #
############

Event AlimAutoEvent {
      ClassName                 : alimAutoEvent
      ParentClass               : Event
      SetPriorityDegree         : 10
      SetOccurrenceProbability  : 100
      SetOccurrenceClockTime    : 0
      NextEventClass            : Event
      SetMinDelayNextEvent      : 0
      SetMaxDelayNextEvent      : 0
      SetNextEventOccurrenceProbability : 100
      SetGenerateNextEvent      : alimAutoEvent_generateNextEvent_Body
      AddProcessAction          : [ alimentationAutomatiqueProcess     / Init ]
      AddProcessAction          : [ alimentationAutomatiqueProcess     / Proceed ]
}
#============================================
Event ArriveeOuvrier {
      ClassName                 : arriveeOuvrier
      ParentClass               : ResourceMobilizationEvent
      SetPriorityDegree         : 99
      SetOccurrenceProbability  : 100
      SetOccurrenceClockTime    : 0
      NextEventClass            : DepartOuvrier
      SetMinDelayNextEvent      : 0
      SetMaxDelayNextEvent      : 0
      SetNextEventOccurrenceProbability : 100
      SetDefineNextEventClockTime : arriveeOuvrier_defineNextEventClockTime_Body
}
#============================================
Event DepartOuvrier {
      ClassName                 : departOuvrier
      ParentClass               : ResourceImmobilizationEvent
      SetPriorityDegree         : 99
      SetOccurrenceProbability  : 100
      SetOccurrenceClockTime    : 0
      NextEventClass            : ArriveeOuvrier
      SetMinDelayNextEvent      : 0
      SetMaxDelayNextEvent      : 0
      SetNextEventOccurrenceProbability : 100
      SetDefineNextEventClockTime : departOuvrier_defineNextEventClockTime_Body
      ProcessedEntityClassId    : O_ENTITY
}
#============================================

########################
#  SequentialDataFile  #
```

```
#######################

SequentialDataFile FluxEntreeCuve {
      ClassSymbol               : FLUX_ENTREE_CUVE
      AccessType                : READ
      AssignPhysicalPath        : IN_DIR
      AssignKeyFormat           : "%d%d"
      AssignDataFormat          : "%f"
      SetNbTopCommentLines      : 2

      AddField                  : H   INT
      AddField                  : MIN      INT
      AddField                  : QUANTITE     FLOAT
}
#============================================
```

Enumérations et variables globales

```
::::::::::::::
global.enum
::::::::::::::
enum ValeursEtatOuverture 0 FERME OUVERT



::::::::::::::
global.var
::::::::::::::
int gIntParam_ATELIER_rythmeAlimAuto_ = 0
int gMyTrace = 0
int gMyVerif = 0
float gRealParam_REBUT_coeffReducProba_ = 0.
int gIntParam_REBUTGRANDES_seuilMaxi_ = 0
int gIntParam_REBUTPETITES_seuilMaxi_ = 0
int gIntParam_CONVOYEUR_longueur_ = 0
int gIntParam_POSTE_duree_ = 0
float gRealParam_MAXIBRIQUE_Volume_ = 0.
float gRealParam_MINIBRIQUE_Volume_ = 0.
float gRealParam_CUBITAINER_Volume_ = 0.
int gIntParam_POSTE_heureDebutAM_ = 0
int gIntParam_POSTE_heureDebutPM_ = 0
```

Annexe 2 : spécifications fonctionnelles

```
#######################################
#  méthodes prédéfinies de : Activity  #
#######################################

bool alimentationManuelle_maxBegStatePredicate_Body(EntityMethod* pM) {
//---------------------------------------------------------------------
// on ne donne qu'une UT a l'alimentation manuelle pour s'operer ou pas
//---------------------------------------------------------------------
  Entity* pAlim = pM->DescribedEntity();
  bool result = FALSE;
  //
  // en l'instant d'ouverture de la conjonction 'magasinOuRebut' ET 'alim',
  // si le niveau dans la cuve est bas, l'alim. est ouverte,
  // si le niveau est encore haut, l'alim. n'est pas ouverte et elle sera
  //    immediatement abandonnee, et il faudra attendre la fermeture de
  //    'magasinOuRebut' pour ouvrir eventuellement une alim.
  //
  ActivityIteration* pCycleProduction = ((Activity*)pAlim)->IsIterated();
  DescriptedEntity* pConjunction =
     (DescriptedEntity*)pCycleProduction->GetElement();
  int instantOuverture = pConjunction->GetIntVarValue(BEG_DATE);
  //
  // le test
  //
  int longueurFenetreOuverture = 0;
  //int longueurFenetreOuverture = 4;
  if(pCurrentSim->Clock() > instantOuverture + longueurFenetreOuverture) {
    result = TRUE;
    if(gMyVerif) printf("\nt = %d alim. manuelle n'est pas mise en oeuvre", pCurrentSim->Clock());
  }
  return result;
};
bool alimentationManuelle_openingPredicate_Body(EntityMethod* pM) {
//-------------------------------------------------------------------
// vrai si le seuil inferieur de la cuve est atteint
//-------------------------------------------------------------------
  bool result = FALSE;
  //
  // acces a la cuve
  //
  BasicEntity* pAtelier = GetFirstEntity(ATELIER_REMPLISSAGE);
  DescriptedEntity* pCuve = (DescriptedEntity*)pAtelier->GetComponent(CUVE);
  //
  // test / seuil
  //
  float volumeCourant = pCuve->GetFloatVarValue(VOLUME_CONTENU);
  float seuilBas =  pCuve->GetFloatConstValue(VOLUME_MINI_CONTENU);
  if(volumeCourant < seuilBas)
    result = TRUE;

  if(gMyVerif && (! result))  printf("\nt = %d alimentationManuelle non requise : volumeCourant = %5.3f seuilBas =
%5.3f", pCurrentSim->Clock(), volumeCourant, seuilBas);
  return result;
};
float infoTransfertCuvePack_setUserTransferedFloatQuantity_Body(EntityMethod* pM) {
//-----------------------------------------------------------------------------
// Retourne la quantite totale a transferer par l'activite en jeu.
// (surcharge une eventuelle valeur du descripteur TransferQuantityFloatValue).
//-----------------------------------------------------------------------------
  float result;
  //
  // on recupere le pack de tete par la spec.d'ensemble d'entites
  // (autre option : exploitation des services sur la structure de l'atelier)
  //
  EntitySpec* pES = App_NewEntitySpecFromString("lePackDeTete");
  pES->Expand();
  pEntityTab* wrappedPack = pES->GetExpansion();
  BasicEntity* lePack = wrappedPack->get_element(0);
  //
  // du pack a la brique
  //
  pEntityTab lesBriques = lePack->GetElementTab();
  DescriptedEntity* uneBrique = (DescriptedEntity*)lesBriques[0];
  //
  // 3 briques dans le pack
  //
  result = uneBrique->GetFloatConstValue(VOLUME) * 3.;

  return result;
};
bool iterationMagasinOuRebut_closingPredicate_Body(EntityMethod* pM) {
//
// vrai ssi    - ouvrier va devenir indisponible
```

```
//          OU - une loge du stock de packs est pleine
//          OU - le stock de cubitainers est vide
//
  Operation* pOt = (Operation*)(pM->DescribedEntity());
  Entity* pAtelier = GetFirstEntity(ATELIER_REMPLISSAGE);
  //
  // l'ouvrier part dans moins de 10 UT (minutes)
  //
  if(OuvrierPartBientot(10))
    return TRUE;
  //
  // chaque loge
  //
  Entity* pMagasin = pAtelier->GetComponent(MAGASIN_EXPEDITION);
  for(int k=0;k<2;k++) {
    Entity* pLoge = pMagasin->GetElement(k);
    int contenance = pLoge->GetIntConstValue(CONTENANCE);
    if(pLoge->GetNumberOfElements() == contenance) {
      return TRUE;
    }
  }
  //
  // le stock de cubis
  //
  Entity* pStockCubis = pAtelier->GetComponent(RESERVE_CUBIS);
  if(pStockCubis->GetNumberOfElements() == 0) {
    return TRUE;
  }
  //
  // rien ne justifie d'arreter
  //
  return FALSE;
};
bool miseAuRebut_openingPredicate_Body(EntityMethod* pM) {
//-----------------------------------------------------------
// Doit renvoyer vrai pour que l'activite puisse etre ouverte.
//-----------------------------------------------------------
  Entity* pE = pM->DescribedEntity();
  bool result;
  //
  // le predicat d'ouverture de la sequence de mise en magasin
  // a deja fait le calcul : on prend l'oppose
  //
  Entity* pOuvrier = GetFirstEntity(OUVRIER);
  result = (bool)pOuvrier->GetIntConstValue(PRET_A_PAQUETER);
  return ! result;
};
bool miseEnRoute_openingPredicate_Body(EntityMethod* pM) {
//-----------------------------------------------------------
// vrai ssi - ouvrier disponible et ne part pas bientot
//          - aucune loge du stock n'est pleine
//          - le stock de cubitainers n'est pas vide
//-----------------------------------------------------------
  Operation* pOt = (Operation*)(pM->DescribedEntity());
  Entity* pAtelier = GetFirstEntity(ATELIER_REMPLISSAGE);
  //
  // l'ouvrier
  //
  Entity* pOuvrier = GetFirstEntity(OUVRIER);
  if(pOuvrier->GetIntVarValue(AVAILABILITY_STATUS) != 1) {
    if(gMyVerif) printf("\nechec openingPredicate miseEnRoute : '%s' indisponible", pOuvrier->InstanceName());
    return FALSE;
  }
  if(OuvrierPartBientot(10)) {
    if(gMyVerif) printf("\nechec openingPredicate miseEnRoute : '%s' part bientot", pOuvrier->InstanceName());
    return FALSE;
  }
  //
  // chaque loge
  //
  Entity* pMagasin = pAtelier->GetComponent(MAGASIN_EXPEDITION);
  for(int k=0;k<2;k++) {
    Entity* pLoge = pMagasin->GetElement(k);
    int contenance = pLoge->GetIntConstValue(CONTENANCE);
    int n = pLoge->GetNumberOfElements();
    if(n == contenance) {
      if(gMyVerif) printf("\nechec openingPredicate miseEnRoute : '%s' pleine (%d packs)", pLoge->InstanceName(),
n);
      return FALSE;
    }
  }
  //
  // le stock de cubis
  //
  Entity* pStockCubis = pAtelier->GetComponent(RESERVE_CUBIS);
  if(pStockCubis->GetNumberOfElements() == 0) {
    if(gMyVerif) printf("\nechec openingPredicate miseEnRoute : '%s' vide", pStockCubis->InstanceName());
    return FALSE;
  }
  //
  // tout est OK
```

```
  //
  return TRUE;
};
bool sequenceMiseEnMagasin_openingPredicate_Body(EntityMethod* pM) {
//-----------------------------------------------------------
// Renvoie vrai ssi 2 autres briques identiques a celle de rang 0
//-----------------------------------------------------------
  BasicEntity* pBefore = pM->DescribedEntity();
  PrimitiveActivity* pCreation = (PrimitiveActivity*)pBefore->GetElement(0);

  Entity* pOOSpec = pCreation->GetEntityVarValue(OPERATED_OBJECT_SPEC_ATTRIBUTE);
  Entity* pConvoyeur = (Entity*)(pOOSpec->GetEntityTabVarValue(PRE_EXPANDED_LIST)->get_element(0));
  bool result = FALSE;

  EntityTabConstantDescriptor* descPlateforme =
    (EntityTabConstantDescriptor*)(pConvoyeur->GetConstantDescriptor(CONVOYEUR_PLATEFORME));
  BasicEntity* pFirstBrique = descPlateforme->GetValueElement(0);
  UserClassId briqueAPaqueterId = pFirstBrique->ClassSymbol();
  int nbSlots = pConvoyeur->GetIntConstValue(CONVOYEUR_LONGUEUR);
  pFloatTab* positionsBriquesAPaqueter = new pFloatTab();
  positionsBriquesAPaqueter->push_back((float)0);
  int nbBriquesAPaqueter = 1;
  for(int k=1;k<nbSlots;k++) {
    BasicEntity* pBrique = descPlateforme->GetValueElement(k);
    if(pBrique->ClassSymbol() == briqueAPaqueterId) {
      positionsBriquesAPaqueter->push_back((float)k);
      nbBriquesAPaqueter++;
    }
    if(nbBriquesAPaqueter == 3) {
      result = TRUE;
      break;
    }
  }
  pConvoyeur->SetFloatTabConstValue(POSITIONS_BRIQUES_A_PAQUETER,
                      positionsBriquesAPaqueter);

  if(gMyVerif) {
    printf("\npositions de 3 memes briques : ");
    positionsBriquesAPaqueter->display();
    if(result) printf(" -> magasin"); else printf(" -> rebut");
  }

  //
  // le predicat d'ouverture de la mise au rebut exploite ce resultat
  //
  Entity* pOuvrier = GetFirstEntity(OUVRIER);
  pOuvrier->SetIntConstValue(PRET_A_PAQUETER, result);

  return result;
}

########################################
#  méthodes prédéfinies de : Operation  #
########################################

bool arreterProduction_stateTransitionProcedure_Body(EntityMethod* pM) {
//
// L'entite operee est l'atelier.
// - on ferme le robinet d'alimentation
// - on detruit le processus d'alimentation automatique
//
  Operation* pOt = (Operation*)(pM->DescribedEntity());
  Entity* pAtelier = pM->GetEntityArgValue(OPERATION_TARGET);
  //
  // le robinet d'alim de la cuve
  //
  Entity* pCuve = pAtelier->GetComponent(CUVE);
  Entity* pRobinetAlim = pCuve->GetComponent(ROBINET_ALIM);
  pRobinetAlim->SetIntVarValue(ETAT_OUVERTURE, FERME);
  if(gMyTrace) printf("\n!!! t = %d  arret production -> robinet %s !!!", pCurrentSim->Clock(),
EnumEtatToChar(FERME));
  //
  // le processus d'alimentation
  //
  StopAlimAuto();

  if(gMyVerif) {
    printf("\nContenu du convoyeur a l'arret de production : {");
    Entity* pConvoyeur = (Entity*)pAtelier->GetComponent(CONVOYEUR);
    EntityTabConstantDescriptor* descPlateforme =
      (EntityTabConstantDescriptor*)(pConvoyeur->GetConstantDescriptor(CONVOYEUR_PLATEFORME));
    pEntityTab* etatConvoyeur = descPlateforme->GetEntityTabValue();
    int nbPositions = etatConvoyeur->size();
    for(int k=0;k<nbPositions;k++) {
      BasicEntity* pPosition = etatConvoyeur->get_element(k);
      if(pPosition->IsInstanceOf(MAXI_BRIQUE))
     printf("G ");
      else
     printf("p ");
    }
    printf("}");
```

```
  }

  return TRUE;
};
bool conditionnerPack_stateTransitionProcedure_Body(EntityMethod* pM) {
//-----------------------------------------------------------------------
// pas d'effet explicite sur l'etat,
// le seul "effet" est la mobilisation de ressources sur une duree
//-----------------------------------------------------------------------
  //Operation* pOt = (Operation*)(pM->DescribedEntity());
  bool result = TRUE;
  //
  // Captage du pack au rang 0 pour message de trace
  //
  Entity* pPack = pM->GetEntityArgValue(OPERATION_TARGET);
  if(gMyTrace) printf("\nt = %d conditionnement '%s'", pCurrentSim->Clock(), pPack->InstanceName());
  return result;
};
bool creerPack_stateTransitionProcedure_Body(EntityMethod* pM) {
//-------------------------------------------------------------
//
//-------------------------------------------------------------
  Operation* pOt = (Operation*)(pM->DescribedEntity());
  Entity* pConvoyeur = pM->GetEntityArgValue(OPERATION_TARGET);
  bool result = TRUE;
  //
  // Creation du pack avec la premiere brique et remplacement au rang 0
  //
  EntityTabConstantDescriptor* descPlateforme =
    (EntityTabConstantDescriptor*)(pConvoyeur->GetConstantDescriptor(CONVOYEUR_PLATEFORME));
//descPlateforme->GetEntityTabValue()->display_names();
  Entity* pNewPack = new Pack();
  pNewPack->AddElement(descPlateforme->GetValueElement(0));
  descPlateforme->SetValueElement(pNewPack, 0);
  //
  // Ajout dans le pack des deux autres briques, avec tassement
  //
  pFloatTab* positionsBriquesAPaqueter =
    pConvoyeur->GetFloatTabConstValue(POSITIONS_BRIQUES_A_PAQUETER);
  if(gMyVerif) positionsBriquesAPaqueter->display();
  for(int k=2;k>0;k--) {
    int index = (int)(positionsBriquesAPaqueter->get_element(k));
    BasicEntity* pMostDistantBrique = descPlateforme->GetValueElement(index);
    if(gMyVerif) printf("\npMostDistantBrique[%d] '%s' :", k, pMostDistantBrique->InstanceName());
    // ajout dans le pack
    pNewPack->AddElement(pMostDistantBrique);
    // remplacement par position vide
    PositionVide* pPositionVide = new PositionVide();
    descPlateforme->SetValueElement(pPositionVide, index);
    // tassement
    pConvoyeur->SetIntArgValue(TASSEMENT_CONVOYEUR,
                    RANG, index+1);//= premiere brique a decaler
    pConvoyeur->ExecVoidMethod(TASSEMENT_CONVOYEUR);
    // alimentation
    pConvoyeur->ExecVoidMethod(ALIMENTATION_CONVOYEUR);
  }

  if(gMyTrace) printf("\nt = %d creation '%s' :", pCurrentSim->Clock(), pNewPack->InstanceName());

  if(gMyVerif) {
    pNewPack->DisplayElementNames();
    descPlateforme->GetEntityTabValue()->display_names();
  }
  return result;
};
bool fermerBrique_stateTransitionProcedure_Body(EntityMethod* pM) {
//-----------------------------------------------------------------
// modif descripteur EtatOuverture, OUVERT dans le constructeur Brique
//-----------------------------------------------------------------
  Operation* pOt = (Operation*)(pM->DescribedEntity());
  Entity* pBrique = pM->GetEntityArgValue(OPERATION_TARGET);
  bool result = TRUE;
  pBrique->SetIntVarValue(ETAT_OUVERTURE, FERME);

  BasicEntity* pPack = pBrique->GetSuperSet();
  if(gMyTrace) printf("\nt = %d fermeture '%s' pour '%s'", pCurrentSim->Clock(), pBrique->InstanceName(), pPack-
>InstanceName());
  return result;
};
bool mettreAuRebut_stateTransitionProcedure_Body(EntityMethod* pM) {
//-----------------------------------------------------------------
// Mise au rebut de la brique de tete
// ssi il n'y en a pas 2 autres identiques sur la plateforme.
// Alternative d'une disjonction dont l'autre est inversement conditionnee
// -> pas de feasibilityCondition explicite pour mettreAuRebut
//-----------------------------------------------------------------
  Operation* pOt = (Operation*)(pM->DescribedEntity());
  Entity* pRebut = pM->GetEntityArgValue(OPERATION_TARGET);
  bool result = TRUE;
  //
  // la brique
```

```
  //
  BasicEntity* pAtelier = pRebut->GetSuperEntity();
  Entity* pConvoyeur = pAtelier->GetComponent(CONVOYEUR);
  EntityTabConstantDescriptor* descPlateforme =
    (EntityTabConstantDescriptor*)(pConvoyeur->GetConstantDescriptor(CONVOYEUR_PLATEFORME));
//descPlateforme->GetEntityTabValue()->display_names();
  BasicEntity* pBrique = descPlateforme->GetValueElement(0);
  int indexLoge = 0;
  if(pBrique->IsInstanceOf(MAXI_BRIQUE))
    indexLoge = 1;
  //
  // la loge du rebut
  //
  BasicEntity* pLoge = pRebut->GetElement(indexLoge);
  //
  // placement de la brique dans la loge
  //
  pLoge->AddElement(pBrique);
  if(gMyTrace) printf("\nt = %d mise au rebut '%s' dans '%s'",
                      pCurrentSim->Clock(),
                      pBrique->InstanceName(),
                      pLoge->InstanceName());
  // ... et remplacement par position vide
  PositionVide* pPositionVide = new PositionVide();
  descPlateforme->SetValueElement(pPositionVide, 0);
  //
  // tassement et alimentation
  //
  pConvoyeur->SetIntArgValue(TASSEMENT_CONVOYEUR,
                  RANG, 1);//= premiere brique a decaler
  pConvoyeur->ExecVoidMethod(TASSEMENT_CONVOYEUR);
  pConvoyeur->ExecVoidMethod(ALIMENTATION_CONVOYEUR);

  return result;
};
bool mettreEnRoute_stateTransitionProcedure_Body(EntityMethod* pM) {
//
// L'entite operee est l'atelier.
// - on initialise le convoyeur
// - on ouvre le robinet d'alimentation
// - on construit le processus d'alimentation automatique
//
  Operation* pOt = (Operation*)(pM->DescribedEntity());
  Entity* pAtelier = pM->GetEntityArgValue(OPERATION_TARGET);
  //
  // le convoyeur
  //
  Entity* pConvoyeur = pAtelier->GetComponent(CONVOYEUR);
  pConvoyeur->ExecVoidMethod(INITIALISATION_CONVOYEUR);
  //
  // le robinet d'alim de la cuve
  //
  Entity* pCuve = pAtelier->GetComponent(CUVE);
  float niveauCourant = pCuve->GetFloatVarValue(VOLUME_CONTENU);
  float niveauMaxi = pCuve->GetFloatConstValue(VOLUME_MAXI_CONTENU);
  if(niveauCourant < niveauMaxi) {
    Entity* pRobinetAlim = pCuve->GetComponent(ROBINET_ALIM);
    pRobinetAlim->SetIntVarValue(ETAT_OUVERTURE, OUVERT);
    if(gMyTrace) printf("\n!!! t = %d mise en route -> robinet %s !!!", pCurrentSim->Clock(), EnumEtatToChar(OU-
VERT));
    //
    // le processus d'alimentation
    //
    StartAlimAuto();
  }
  else {
    if(gMyTrace) printf("\n!!! t = %d mise en route -> robinet reste %s !!!", pCurrentSim->Clock(), EnumEtatTo-
Char(FERME));
  }

  return TRUE;
};
float remplirBriques_setUserFloatQuantitySpeed_Body(EntityMethod* pM) {
//
// Retourne la vitesse de l'operation.
// (surcharge une eventuelle valeur du descripteur FloatQuantitySpeed).
//
  Operation* pOt = (Operation*)(pM->DescribedEntity());
  float result;
  //
  // le debit (l/min) du robinet de vidange
  //
  BasicEntity* pAtelier = GetFirstEntity(ATELIER_REMPLISSAGE);
  BasicEntity* pCuve = pAtelier->GetComponent(CUVE);
  DescriptedEntity* pRobinetVidange =
    (DescriptedEntity*)pCuve->GetComponent(ROBINET_VIDANGE);
  float debit = pRobinetVidange->GetFloatConstValue(DEBIT);
  //
  // duree (min) du pas de l'OT
  //
  int stepLength = ClockIntervalToTimeInterval(pOt->GetStep()) / 60;
```

```
  //
  // vitesse (l) = debit (l/min) * duree (min)
  //
  result = debit * (float)stepLength;

  return result;
};
bool stockerPack_stateTransitionProcedure_Body(EntityMethod* pM) {
//-----------------------------------------------------------------
//
//-----------------------------------------------------------------
  Operation* pOt = (Operation*)(pM->DescribedEntity());
  Entity* pConvoyeur = pM->GetEntityArgValue(OPERATION_TARGET);
  bool result = TRUE;
  //
  // Captage du pack au rang 0
  //
  EntityTabConstantDescriptor* descPlateforme =
    (EntityTabConstantDescriptor*)(pConvoyeur->GetConstantDescriptor(CONVOYEUR_PLATEFORME));
  BasicEntity* pPack = descPlateforme->GetValueElement(0);
  //
  // Recherche du lieu de stockage, fonction de la taille des briques
  //
  BasicEntity* pAtelier = pConvoyeur->GetSuperEntity();
  DescriptedEntity* pMagasin = (DescriptedEntity*)pAtelier->GetComponent(MAGASIN_EXPEDITION);
  DescriptedEntity* pUneBrique = (DescriptedEntity*)pPack->GetElement();
  int index = 0;
  if(pUneBrique->IsInstanceOf(MAXI_BRIQUE))
    index = 1;
  BasicEntity* pLoge = pMagasin->GetElement(index);
  //
  // Ajout du pack dans la loge ...
  //
  pLoge->AddElement(pPack);
  if(gMyTrace) printf("\nt = %d stockage '%s' ('%s') dans '%s'",
                      pCurrentSim->Clock(),
                      pPack->InstanceName(),
                      ClassIdToCharTab[pUneBrique->ClassSymbol()],
                      pLoge->InstanceName());
  // ... et remplacement par position vide
  PositionVide* pPositionVide = new PositionVide();
  descPlateforme->SetValueElement(pPositionVide, 0);
  //
  // cumul de production dans une loge ou l'autre
  //
  float production;
  if(index == 0) {
    production = pMagasin->GetFloatVarValue(PRODUCTION_MINI);
    production += 3. * pUneBrique->GetFloatConstValue(VOLUME);
    pMagasin->SetFloatVarValue(PRODUCTION_MINI, production);
  }
  else {
    production = pMagasin->GetFloatVarValue(PRODUCTION_MAXI);
    production += 3. * pUneBrique->GetFloatConstValue(VOLUME);
    pMagasin->SetFloatVarValue(PRODUCTION_MAXI, production);
  }
  //
  // Tassement en tete et alimentation en queue
  //
  pConvoyeur->SetIntArgValue(TASSEMENT_CONVOYEUR,
                   RANG, 1);//= premiere brique a decaler
  pConvoyeur->ExecVoidMethod(TASSEMENT_CONVOYEUR);
  // alimentation
  pConvoyeur->ExecVoidMethod(ALIMENTATION_CONVOYEUR);

  return result;
};
bool verserCubi_stateTransitionProcedure_Body(EntityMethod* pM) {
//-----------------------------------------------------------------
// operation d'alimentation manuelle de la cuve
//-----------------------------------------------------------------
  Operation* pOt = (Operation*)(pM->DescribedEntity());
  BasicEntity* pReserveCubis = pM->GetEntityArgValue(OPERATION_TARGET);
  bool result = TRUE;
  //
  // on sort le premier cubi venu
  //
  DescriptedEntity* pCubi = (DescriptedEntity*)pReserveCubis->GetElement();
  pReserveCubis->RemoveElement(0);
  //
  // on le verse dans la cuve
  //
  Entity* pAtelier = GetFirstEntity(ATELIER_REMPLISSAGE);
  Entity* pCuve = pAtelier->GetComponent(CUVE);
  float volumeCourant = pCuve->GetFloatVarValue(VOLUME_CONTENU);
  float volumeCubi = pCubi->GetFloatConstValue(VOLUME);
  float volumeApres = volumeCourant+volumeCubi;
  pCuve->SetFloatVarValue(VOLUME_CONTENU, volumeApres);

  if(gMyTrace) {
    int clock = pCurrentSim->Clock();
```

```
    Date date = ClockToDate(clock);
    if(date.minute < 10)
      printf("\n\t%dh0%d'", date.hour, date.minute);
    else
      printf("\n\t%dh%d'", date.hour, date.minute);
    printf(" niveau avant = %5.2f", volumeCourant);
    printf("       alim manuelle de %5.2f", volumeCubi);
    printf("  apres = %5.2f", volumeApres);
  }

  return result;
};

################################################
#  méthodes prédéfinies de : ContinuousProcess  #
################################################

void alimentationAutomatiqueProcess_goOneStepForwardProcess_Body(ProcessMethod* pM)
//-----------------------------------------
// augmentation niveau cuve en un pas
//-----------------------------------------
{
  ContinuousProcess* pCP = pM->DescribedContinuousProcess();
  Cuve* pCuve = (Cuve*)pCP->ProcessedEntity();
  int clock = pCurrentSim->Clock();
  Date date = ClockToDate(clock);

  float contenance = pCuve->GetFloatConstValue(VOLUME);
  float volumeCourant = pCuve->GetFloatVarValue(VOLUME_CONTENU);
  float entreeDispo = pCuve->GetFloatConstValue(ENTREE_DISPONIBLE);
  if(gMyTrace) {
    if(date.minute < 10)
      printf("\n\t%dh0%d'", date.hour, date.minute);
    else
      printf("\n\t%dh%d'", date.hour, date.minute);
    printf(" niveau avant = %5.2f", volumeCourant);
    printf("  dispo = %5.2f", entreeDispo);
  }

  Robinet* pRobinet = (Robinet*)pCuve->GetComponent(ROBINET_ALIM);
  float debitAlim = pRobinet->GetFloatConstValue(DEBIT);

  float transfertMaxEnUnPas = debitAlim * pCP->Pas();
  if(gMyVerif) printf("\n\ttransfertMaxEnUnPas = %5.2f", transfertMaxEnUnPas);
  //
  // on transfere au plus ce que permet le debit
  //              ou moins si entree insuffisante ou debordement
  //
  float entreePotentielle = transfertMaxEnUnPas;
  if(entreeDispo < transfertMaxEnUnPas) {
    entreePotentielle = entreeDispo;
  }
  float debordement = volumeCourant + entreePotentielle - contenance;
  float entreeReelle = entreePotentielle;
  if(debordement > 0.) {
    if(gMyVerif) printf("\n\tdebordement = %5.2f", debordement);
    entreeReelle = entreePotentielle - debordement;
  }
  if(gMyTrace) printf("  alim = %5.2f", entreeReelle);
  volumeCourant = volumeCourant + entreeReelle;
  pCuve->SetFloatVarValue(VOLUME_CONTENU, volumeCourant);
  if(gMyTrace) printf("  apres = %5.2f", volumeCourant);
  entreeDispo = entreeDispo - entreePotentielle;
  pCuve->SetFloatConstValue(ENTREE_DISPONIBLE, entreeDispo);
  //
  // s'il n'y a plus rien en entree, on stoppe le processus
  //
  if(entreeDispo < 0.0001)  // attention a la precision sur les reels
    pCP->ToBeStopped(TRUE);
};
void alimentationAutomatiqueProcess_initializeProcess_Body(ProcessMethod* pM)
//-----------------------------------------
// ce processus est initialise a intervelles reguliers
// par l'autogeneration de son evenement declencheur
//-----------------------------------------
{
  ContinuousProcess* pCP = pM->DescribedContinuousProcess();
  pCP->ToBeStopped(FALSE);
  // car mis a TRUE dans GoOneStepForward quand le transfert est termine
  // (pour stopper le processus et provoquer l'auto-generation)

  BasicEntity* pCuve = pCP->ProcessedBasicEntity();;
  Robinet* pRobinet = (Robinet*)pCuve->GetComponent(ROBINET_ALIM);
  int etatRobinet = pRobinet->GetIntVarValue(ETAT_OUVERTURE);
  //
  // warning en principe jamais affiche (because precondition)
  //
  if(etatRobinet == FERME) {
    printf("\n!!! t = %d Le robinet d'alimentation est ferme, de maniere inattendue !!!", pCurrentSim->Clock());
    exit(0);
  }
```

```
  else {
    if(gMyVerif) printf("\n!!! t = %d Le robinet d'alimentation est bien ouvert, comme attendu !!!", pCurrentSim-
>Clock());
  }
};
int alimentationAutomatiqueProcess_processPostponementDelay_Body(ProcessMethod* pM)
//-------------------------------------------
// Definition de la fonction de delai de reexamen du processus.
//-------------------------------------------
{
  ContinuousProcess* pCP = pM->DescribedContinuousProcess();
  BasicEntity* pAlimentationAutomatiqueProcess = pCP->ProcessedEntity();
  return GetIntParameterValue("ATELIER", "rythmeAlimAuto");
};
int alimentationAutomatiqueProcess_processPrecondition_Body(ProcessMethod* pM)
//-------------------------------------------
// Le robinet d'alimentation doit etre ouvert pour que le processus avance.
// Si c'est pas le cas, l'alimentation est repoussee du 'postponement delay'
// (car FERME = 0 ; une valeur negative stopperait definitivement l'alim.)
//-------------------------------------------
{
  //
  // le robinet d'alimentation doit etre ouvert
  //
  ContinuousProcess* pCP = pM->DescribedContinuousProcess();
  BasicEntity* pCuve = pCP->ProcessedEntity();
  Robinet* pRobinet = (Robinet*)pCuve->GetComponent(ROBINET_ALIM);
  int etatRobinet = pRobinet->GetIntVarValue(ETAT_OUVERTURE);
  return etatRobinet; // OUVERT = 1
};

###############################################
#  méthodes prédéfinies de : DiscreteProcess  #
###############################################


###############################################
#  méthodes prédéfinies de : ReadSDFProcess  #
###############################################

void fluxEntreeCuveLectureProcess_initializeProcess_Body(ProcessMethod* pM)
//-------------------------------------------
// Definition de la fonction qui code l'initialisation
// du processus continu de lecture d'un fichier sequentiel.
//-------------------------------------------
{
  ReadSequentialDataFileProcess* pCP = (ReadSequentialDataFileProcess*)(pM->DescribedContinuousProcess());

  UserFileId fileClassId = pCP->FileClassId();
  SequentialDataFile* pF = App_NewSequentialDataFileFromId(fileClassId);
  if(! pF)
    UnknownClassSymbol("SequentialDataFile", fileClassId);
  pCP->SetFile(pF);

  //-------------------------------------------
  // recuperation des specifications du fichier
  //-------------------------------------------
  pCP->SetNamesFromFile(pF);

  //-------------------------------------------
  // etablissement du chemin complet du fichier
  // a partir des specifications du fichier et du processus
  //-------------------------------------------
  pCP->AssignFullNameToFile(pF);

  //---------------------
  // ouverture du fichier
  //---------------------
  pF->Open();
  int n;
  // n = pF->GetNbTopCommentLines(); ssi la simulation demarre a la date
  //                                 du premier enregistrement du fichier

  time_t timeDebFile = DateToAbsoluteTime(1, JAN, 2007, 0, 0, 0);
  time_t timeDebSim = pCurrentSim->AbsoluteBeginningTime();

  // timeDiff : nb de secondes entre 1er JAN 0h et DATE_DEBUT (.sim)
  time_t timeDiff = timeDebSim - timeDebFile;

  // clockDiff : nb UT = nb de 'UNITE_TPS * NB_UNITE_TPS' = nb de '1 * MINUTE'
  int clockDiff = TimeIntervalToClockInterval(timeDiff);

  // la duree entre les "dates" de deux enregistrements successifs,
  // c'est la longueur du pas du processus ! D'ou le nb de lignes a sauter :
  int n0 = clockDiff / pCP->Step();

  // ... sans oublier les lignes d'entete
  n = n0 + pF->GetNbTopCommentLines();

  pF->Skip(n);
};
```

```
void fluxEntreeCuveLectureProcess_postReadProcess_Body(ProcessMethod* pM)
//-----------------------------------------------
// Definition de la fonction qui code l'exploitation des valeurs
// lues sur un enregistrement.
//-----------------------------------------------
{
  ReadSequentialDataFileProcess* pCP = (ReadSequentialDataFileProcess*)(pM->DescribedContinuousProcess());
  Cuve* pCuve = (Cuve*)pCP->ProcessedEntity();
  //
  // le fichier logique dont les champs viennent d'etre values par lecture
  //
  SequentialDataFile* pF = pCP->GetFile();
  //
  // la valeur du champ 'quantite' devient l'entree disponible de la cuve
  //
  float q = pF->GetFloatFieldValue(QUANTITE);
  int heure = pF->GetIntFieldValue(H);
  int minute = pF->GetIntFieldValue(MIN);
  if(gMyTrace) {
    if(minute < 10)
      printf("\n\n%dh0%d' : %5.2f litres disponibles en entree de la cuve",
          heure, minute, q);
    else
      printf("\n\n%dh%d' : %5.2f litres disponibles en entree de la cuve",
          heure, minute, q);
  }
  pCuve->SetFloatConstValue(ENTREE_DISPONIBLE, q);
};
void fluxEntreeCuveLectureProcess_stopProcess_Body(ProcessMethod* pM)
//-----------------------------------------------
// Definition de la fonction qui code l'arret du processus de lecture.
//-----------------------------------------------
{
  ReadSequentialDataFileProcess* pCP = (ReadSequentialDataFileProcess*)(pM->DescribedContinuousProcess());
  SequentialDataFile* pF = pCP->GetFile();
  //
  // puisque fin fichier atteinte, on arrete l'alimentation automatique
  //
  StopAlimAuto();
  //
  // instructons du code predefini
  //
  pF->Close();
  pCP->ToBeStopped(TRUE);
};

####################################
#  méthodes prédéfinies de : Event  #
####################################

bool alimAutoEvent_generateNextEvent_Body(EventMethod* pM)
//-----------------------------------------------
// Generation de l'event de lancement de la prochaine alim. automatique
// sauf si l'ouvrier est parti
//-----------------------------------------------
{
  Event* pCurrentEvent = pM->DescribedEvent();
  Event* pNewEvent = pCurrentEvent->RawCopy();
  int previousEventDate = pCurrentEvent->OccurrenceClockTime();

  int delai = gIntParam_ATELIER_rythmeAlimAuto_;
  //
  // le delai part de l'init du process, pas de l'instant de son arret
  //
  int nextEventClockTime = previousEventDate + delai;
  pNewEvent->SetOccurrenceClockTime(nextEventClockTime);
  pCurrentEvent->SetNextEvent(pNewEvent);
  //
  // on arrete l'auto-generation si l'ouvrier est parti
  //
  Entity* pOuvrier = GetFirstEntity(OUVRIER);
  int availabilityStatus = pOuvrier->GetIntVarValue(AVAILABILITY_STATUS);
  if(gMyVerif) printf("\ndans alimAutoEvent_generateNextEvent, available = %d", availabilityStatus);

  return availabilityStatus;
};
int arriveeOuvrier_defineNextEventClockTime_Body(EventMethod* pM)
//-----------------------------------------------
// L'ouvrier est arrive a x, il repart a x+dureePlage
//-----------------------------------------------
{
  if(gMyTrace) printf("\n!!! t = %d ouvrier ARRIVEE !!!", pCurrentSim->Clock());
  int departClockTime;

  Event* pArriveeEvent = pM->GetEventArgValue(PREVIOUS_EVENT_ARGUMENT);
  int arriveeClockTime = pArriveeEvent->OccurrenceClockTime();

  int duree = gIntParam_POSTE_duree_;
  departClockTime = arriveeClockTime + duree;

  return departClockTime;
```

```
};
int departOuvrier_defineNextEventClockTime_Body(EventMethod* pM)
//-----------------------------------------------
// Si l'ouvrier est parti dejeuner, il reprend a heureDebutPM (e.g 14h)
// S'il est parti le soir, il reprend a heureDebutAM le lendemain (e.g. 8h)
//-----------------------------------------------
{
  if(gMyTrace) printf("\n!!! t = %d ouvrier DEPART !!!", pCurrentSim->Clock());
  int arriveeClockTime;

  Event* pDepartEvent = pM->GetEventArgValue(PREVIOUS_EVENT_ARGUMENT);
  int departClockTime = pDepartEvent->OccurrenceClockTime();

  Date departEventDate = ClockToDate(departClockTime);

  int heureDebutAM = GetIntParameterValue("POSTE", "heureDebutAM");
  int heureDebutPM = GetIntParameterValue("POSTE", "heureDebutPM");

  if(departEventDate.hour < heureDebutPM) {
    arriveeClockTime =
      DateToClock(departEventDate.mday,
           departEventDate.month,
           departEventDate.year,
           heureDebutPM, 0, 0);
  }
  else {
    int dureeJour = TimeIntervalToClockInterval(24*60*60);
    arriveeClockTime =
      DateToClock(departEventDate.mday,
           departEventDate.month,
           departEventDate.year,
           heureDebutAM, 0, 0)
      +dureeJour;
  }
  return arriveeClockTime;
};

########################################
#  méthodes prédéfinies de : EntitySpec  #
########################################

pEntityTab* lePackDeTete_entityListInstantiator_Body(EntitySpecMethod* pM)
//-----------------------------------------------
// Construit une liste d'entites correspondant a la specification.
//-----------------------------------------------
{
  EntitySpec* es = pM->DescribedEntitySpec();
  pEntityTab* result = es->GetExpansion();
  result->erase();
  UserClassId classId = pM->GetIntArgValue(CLASSID_ARGUMENT);  // PACK

  Entity* leConvoyeur = (Entity*)GetFirstEntity(CONVOYEUR);
  EntityTabConstantDescriptor* descPlateforme =
    (EntityTabConstantDescriptor*)(leConvoyeur->GetConstantDescriptor(CONVOYEUR_PLATEFORME));
  BasicEntity* pObj = descPlateforme->GetValueElement(0);
  if(pObj->IsInstanceOf(classId))
    result->push_back(pObj);

  return result;
};

bool lesAteliers_selectorPredicate_Body(EntitySpecMethod* pM)
//-----------------------------------------------
// Determine si l'entite pE correspond a la specification.
//-----------------------------------------------
{
  bool result = false;
  /*
  Entity* pE = pM->GetEntityArgValue(CANDIDATEENTITY_ARGUMENT);
  Entity* pEref = pM->GetEntityArgValue(REFERENCEENTITY_ARGUMENT);
  */
  result = true;

  return result;
};

##############################################
#  méthodes prédéfinies de : DescValueMonitor  #
##############################################

float volumeContenuCuve_whenSetFloat_Body(MonitorMethod* pM)
//-----------------------------------------------
// remontee au-dessus seuil  maxi  -> fermeture robinet
// descente au-dessous  seuil  moyen -> ouverture robinet
//-----------------------------------------------
{
  Descriptor* pVD = pM->GetDescriptorArgValue(MONITOREDDESCRIPTOR_ARGUMENT);
  float oldValue = pM->GetFloatArgValue(CURRENTFLOATVALUE_ARGUMENT);
  float newValue = pM->GetFloatArgValue(CANDIDATEFLOATVALUE_ARGUMENT);

  bool seuilMaxiAtteint = false;
```

```
  bool seuilMoyenAtteint = false;
  //
  // la cuve ...
  //
  Entity* laCuve = pVD->DescribedEntity();
  float niveauMaxi = laCuve->GetFloatConstValue(VOLUME_MAXI_CONTENU);
  if((oldValue <= niveauMaxi)
     &&
     (newValue > niveauMaxi)) {
    seuilMaxiAtteint = true;
    float volume = laCuve->GetFloatConstValue(VOLUME);
    if(newValue > volume)
      newValue = volume;
  }
  else if(newValue < 0.) {
      newValue = 0.;
    }
  else {
    float niveauMini = laCuve->GetFloatConstValue(VOLUME_MINI_CONTENU);
    float niveauMoyen = (niveauMaxi + niveauMini)/2.;
    if((oldValue >= niveauMoyen)
     &&
        (newValue < niveauMoyen)) {
      seuilMoyenAtteint = true;
    }
  }
  //
  // ... et son robinet d'alimentation
  //
  Entity* leRobinetAlim;
  if(seuilMaxiAtteint) {
    leRobinetAlim = laCuve->GetComponent(ROBINET_ALIM);
    if(gMyTrace)
      if(leRobinetAlim->GetIntVarValue(ETAT_OUVERTURE) == OUVERT)
     printf("\nt = %d !!! FERMETURE ROBINET ALIM !!!", pCurrentSim->Clock());
    leRobinetAlim->SetIntVarValue(ETAT_OUVERTURE, FERME);
    //
    // on stoppe l'alimentation automatique
    //
    StopAlimAuto();
  }
  else if(seuilMoyenAtteint) {
    leRobinetAlim = laCuve->GetComponent(ROBINET_ALIM);
    if(gMyTrace)
      if(leRobinetAlim->GetIntVarValue(ETAT_OUVERTURE) == FERME)
     printf("\nt = %d !!! OUVERTURE ROBINET ALIM !!!", pCurrentSim->Clock());
    leRobinetAlim->SetIntVarValue(ETAT_OUVERTURE, OUVERT);
    //
    // on reprend l'alimentation automatique
    //
    StartAlimAuto();
  }

  return newValue;
};
float volumeContenuPack_whenSetFloat_Body(MonitorMethod* pM)
//-------------------------------------------
// Definition de la fonction qui sera declenchee lors de
// chaque modification de la variable surveillee.
//-------------------------------------------
{
  Descriptor* pVD = pM->GetDescriptorArgValue(MONITOREDDESCRIPTOR_ARGUMENT);
  float oldValue = pM->GetFloatArgValue(CURRENTFLOATVALUE_ARGUMENT);
  float newValue = pM->GetFloatArgValue(CANDIDATEFLOATVALUE_ARGUMENT);

  if(gMyTrace) {
    DescriptedEntity* pCuve = GetFirstEntity(CUVE);
    float niveauCuve = pCuve->GetFloatVarValue(VOLUME_CONTENU);
    printf("\nt = %d remplissage de '%s' avec %5.3f litres -> niveauCuve = %5.3f", pCurrentSim->Clock(), pVD->Des-
cribedEntity()->InstanceName(), newValue-oldValue, niveauCuve);
  }

  return newValue;
};

###############################################
#  méthodes prédéfinies de : StructureMonitor  #
###############################################

BasicEntity* logeRebut_whenAddElement_Body(MonitorMethod* pM)

//-------------------------------------------
// Definition de la fonction qui sera declenchee lors de
// chaque modification de la structure d'un Monitor.
//-------------------------------------------
{
  BasicEntity* loge = pM->GetEntityArgValue(CURRENTENTITYCONTAINER_ARGUMENT);
  BasicEntity* packToAdd = pM->GetEntityArgValue(CANDIDATEENTITYELEMENT_ARGUMENT);
  DescriptedEntity* rebut = (DescriptedEntity*)loge->GetSuperSet();
  // ci-dessous : voir le commentaire du descripteur
  int lastProbaRaised = rebut->GetIntConstValue(DERNIERE_PROBA_AUGMENTEE);
```

```
  // par convention : petites briques en 0 grandes en 1
  int nbPetites = rebut->GetElement(0)->GetNumberOfElements();
  int nbGrandes = rebut->GetElement(1)->GetNumberOfElements();

  Entity* leConvoyeur = GetFirstEntity(CONVOYEUR);
  Entity* generateurDeBriques = leConvoyeur->GetEntityConstValue(CONVOYEUR_GENERATEUR);
  float probaPetite =
    generateurDeBriques->GetFloatConstValue(PROBA_PETITE_BRIQUE);
  float oldProbaPetite = probaPetite;  // pour la trace si gMyVerif

  bool anyChange = FALSE;
  // ci-dessous :
  // >= parce que le test est fait avant la mise au rebut
  // lastProbaRaised != 0 pour ne pas augmenter 2 fois de suite la proba
  //                     de generer une petite brique
  if((nbPetites >= gIntParam_REBUTPETITES_seuilMaxi_)
     &&
     (nbPetites >= nbGrandes*2)
     &&
     (lastProbaRaised != 0)) {
    //trop de petites au rebut -> on augmente la proba des petites
    probaPetite /= gRealParam_REBUT_coeffReducProba_;
    if(probaPetite > 1.) probaPetite = 1.0;
    generateurDeBriques->SetFloatConstValue(PROBA_PETITE_BRIQUE, probaPetite);
    rebut->SetIntConstValue(DERNIERE_PROBA_AUGMENTEE, 0);
    anyChange = TRUE;
  }
  else if((nbGrandes >= gIntParam_REBUTGRANDES_seuilMaxi_)
     &&
     (nbGrandes >= nbPetites*2)
     &&
     (lastProbaRaised != 1)) {
    //trop de grandes au rebut -> on diminue la proba des petites
    probaPetite *= gRealParam_REBUT_coeffReducProba_;
    generateurDeBriques->SetFloatConstValue(PROBA_PETITE_BRIQUE, probaPetite);
    rebut->SetIntConstValue(DERNIERE_PROBA_AUGMENTEE, 1);
    anyChange = TRUE;
  }
  if(gMyTrace && anyChange) {
    printf("\nproba alimentation : petite = %4.2f -> %4.2f / grande = %4.2f -> %4.2f", oldProbaPetite, probaPe-
tite, 1.-oldProbaPetite, 1.-probaPetite);
  }
  return packToAdd; // sera ajoute comme element de la loge
};

##########################
#  méthodes de : Entity  #
##########################

BasicEntity* alimentationConvoyeur_Body(EntityMethod* pM)
//-------------------------------------------------------
// renvoie la brique generee et placee en derniere position
//-------------------------------------------------------
{
  Convoyeur* pE = (Convoyeur*)pM->DescribedEntity();
  Brique* pNouvelleBrique = NULL;
  //
  // generation de la nouvelle brique ...
  //
  GenerateurDeBriques* pGenBrq =
    (GenerateurDeBriques*)pE->GetEntityConstValue(CONVOYEUR_GENERATEUR);

  // la probalilite de generer une petite brique
  float probaPetite = pGenBrq->GetFloatConstValue(PROBA_PETITE_BRIQUE);

  // tirage uniforme dans [0 99]
  int tirage = RandomInteger(100);

  if(((float)tirage)/100. < probaPetite)
    pNouvelleBrique = new MiniBrique();
  else
    pNouvelleBrique = new MaxiBrique();
  //
  // le descripteur a valeur 'plateforme' du convoyeur
  //
  EntityTabConstantDescriptor* descPlateforme =
    (EntityTabConstantDescriptor*)(pE->GetConstantDescriptor(CONVOYEUR_PLATEFORME));
  //
  // on fait 'place vide' (desallocaton de la PositionVide surchargee)
  //
  int longueurConvoyeur = pE->GetIntConstValue(CONVOYEUR_LONGUEUR);
  BasicEntity* pPV = descPlateforme->GetValueElement(longueurConvoyeur-1);
  DeleteEntity(pPV);
  //
  // ... et placement en queue du convoyeur
  //
  descPlateforme->SetValueElement(pNouvelleBrique, longueurConvoyeur-1);

  return pNouvelleBrique;
};
```

```
void initialisationConvoyeur_Body(EntityMethod* pM)
//-------------------------------------------------
// remplace les positions vides pres du generateur par des briques
//-------------------------------------------------
{
  Convoyeur* pConvoyeur = (Convoyeur*)pM->DescribedEntity();
  int nbSlots = pConvoyeur->GetIntConstValue(CONVOYEUR_LONGUEUR);
  //
  // le descripteur a valeur 'plateforme' du convoyeur
  //
  EntityTabConstantDescriptor* descPlateforme =
    (EntityTabConstantDescriptor*)(pConvoyeur->GetConstantDescriptor(CONVOYEUR_PLATEFORME));
  if(gMyTrace) printf("\nContenu du convoyeur a l'initialisation : {");
  //
  // recherche du premier rang vide (k)
  //
  int k;
  for(k=0;k<nbSlots;k++) {
    BasicEntity* pE = descPlateforme->GetValueElement(k);
    if(pE->IsInstanceOf(POSITION_VIDE))
      break;
    else {
      if(gMyTrace) {
     char* type = "p";
     if(pE->IsInstanceOf(MAXI_BRIQUE))
       type = "G";
     printf("%s ", type);
      }
    }
  }
  //
  // alimentation du convoyeur avec nbSlot-k   generations
  //                           dont nbSlot-k-1 tassements
  //
 for(int k2=nbSlots-1; // premier slot a alimenter = nbSlots-1
     k2>=k;            // dernier slot a alimenter = k
     k2--) {
    BasicEntity* pE = pConvoyeur->ExecEntityMethod(ALIMENTATION_CONVOYEUR);
    if(gMyVerif) {
      printf("\n\tgeneration de %x '%s'", pE, pE->InstanceName());
    }
    if(gMyTrace) {
      char* type = "p";
      if(pE->IsInstanceOf(MAXI_BRIQUE))
     type = "G";
      printf("%s ", type);
    }
    if(k2 > k) {
      pConvoyeur->SetIntArgValue(TASSEMENT_CONVOYEUR,
                         RANG, k2);   // k2 = premiere brique a decaler
      pConvoyeur->ExecVoidMethod(TASSEMENT_CONVOYEUR);
    }
  }
  if(gMyTrace) printf("}\n");
};
void tassementConvoyeur_Body(EntityMethod* pM)
//-------------------------------------------
// le rang des entites a partir du rang <arg> est dimunue de 1
//-------------------------------------------
{
  //
  // le rang a partir duquel on decale les entites
  //
  int rang = pM->GetIntArgValue(RANG);

  Entity* pConvoyeur = pM->DescribedEntity();
  int taille = pConvoyeur->GetIntConstValue(CONVOYEUR_LONGUEUR);
  //
  // le descripteur a valeur 'plateforme' du convoyeur
  //
  EntityTabConstantDescriptor* descPlateforme =
    (EntityTabConstantDescriptor*)(pConvoyeur->GetConstantDescriptor(CONVOYEUR_PLATEFORME));
  //
  // on fait 'place vide' (desallocaton de PositionVide surchargee)
  //
  BasicEntity* pPV = descPlateforme->GetValueElement(rang-1);
  DeleteEntity(pPV);
  //
  // on decale
  //
  int k;
  for(k=rang;k<taille;k++) {
    BasicEntity* pE = descPlateforme->GetValueElement(k);
    descPlateforme->SetValueElement(pE, k-1);
  }
  //
  // la derniere position est une position vide
  //
  PositionVide* pVide = new PositionVide();
  descPlateforme->SetValueElement(pVide, k-1);
};
```

```
############################
#  méthodes de : Process  #
############################


########################
#  méthodes de : Event  #
########################


########################
#  Fonctions globales  #
########################

void App_ParseMainArguments (int argc, char* argv[])
//--------------------------------------------
// Description  : interpretation of main function arguments
// Precondition :
// Retour       :
//--------------------------------------------
{
  if (argc < 6) {
    printf("Usage : main system_parameters simulation_parameters output_specifications structure directives");
    printf("        [-m(emory)] [-v(erbose)] [-t(race)] [-f(trace)]\n");
    exit(0);
  };

  for (int i=6;i<argc;i++) {
    if (!strcmp(argv[i], "-v"))
      gTraceActionMode  = TRACE_ACTION_ON;
    if (!strcmp(argv[i], "-m"))
      gTraceNewDelMode  = TRACE_NEWDEL_ON;
    if (!strcmp(argv[i], "-t"))
      gMyTrace          = TRUE;
    if (!strcmp(argv[i], "-f"))
      gMyVerif          = TRUE;
  };
}
char* EnumEtatToChar (int etat)
//--------------------------------------------
// Description  :
// Precondition :
// Retour       :
//--------------------------------------------
{
  switch(etat) {
    case 0 : {return "FERME"; break;}
    case 1 : {return "OUVERT"; break;}
  }
};
bool OuvrierPartBientot(int delai) {
//--------------------------------------------
// Description  : vrai ssi l'ouvrier part dans moins de 'delai' UT (minutes)
// Precondition :
// Retour       :
//--------------------------------------------
  bool result = FALSE;
  int now = pCurrentSim->Clock();
  pCurrentSim->GoHeadInAgenda();
  while(pCurrentSim->IsCurrentInAgenda()) {
    Event* pEvt = pCurrentSim->GetCurrentInAgenda();
    if(! strcmp(pEvt->ClassName(), "departOuvrier")) {
      int dateDepart = pEvt->OccurrenceClockTime();
      if((dateDepart - now) < delai) {
     result = TRUE;
      }
    }
    pCurrentSim->GoNextInAgenda();
  }
  return result;
}
void StartAlimAuto()
//--------------------------------------------
// Description  : Place dans l'agenda un evenement d'alim. automatique
// Precondition :
// Retour       : aucun
//--------------------------------------------
{
  //
  // recherche de l'evenement 'flux d'entree' pour date d'occurrence
  //
  Event* pFluxEvent = NULL;
  pCurrentSim->GoHeadInAgenda();
  while(pCurrentSim->IsCurrentInAgenda()) {
    pFluxEvent = pCurrentSim->GetCurrentInAgenda();
    Process* pP = pFluxEvent->FirstProcess();
    if(! strcmp(pP->ClassName(), "fluxEntreeCuveLectureProcess")) {
      break;
    }
    pCurrentSim->GoNextInAgenda();
```

```
  }
  if(pFluxEvent) {
    Event* pNewAlimEvent = new AlimAutoEvent();
    int dateOccurrence = pFluxEvent->OccurrenceClockTime();
    pNewAlimEvent->SetOccurrenceClockTime(dateOccurrence);
    //
    // le nouvel evenement est le premier d'une serie autogeneree
    //
    pNewAlimEvent->IsAutoGenerate(TRUE);
    bool planned = pCurrentSim->InsertInAgendaEvents(pNewAlimEvent);
    if(! planned) {
      pNewAlimEvent->DeleteProcesses();
      pNewAlimEvent->DeleteInDepth();
    }
  }
  else {
    printf("\n!! Absence inattendue d'un evenement de flux d'entree dans l'agenda.\n");
  }
}
void StopAlimAuto ()
//--------------------------------------------
// Description  : Stoppe l'alimentation automatique en cours et/ou programmee.
// Precondition : aucune
// Retour       : aucun
//--------------------------------------------
{
  //
  // Si c'est l'evenement en cours qui gere l'alimentation automatique,
  // le processus est stoppe, mais l'evenement est immediatement regenere
  // a la date de la prochaine arrivee de jus de fruit en amont de la cuve
  // (cas du depassement du seuil maxi de la cuve).
  //
  Event* popEvent = pCurrentSim->CurrentlyPoppedEvent();
  Process* popProcess = popEvent->FirstProcess();
  if(! strcmp(popProcess->ClassName(), "alimentationAutomatiqueProcess")) {
    ((ContinuousProcess*)popProcess)->ToBeStopped(TRUE);
  }
  //
  // Si un [autre] evenement dans l'agenda (autogenere ou continuation)
  // gere l'alimentation automatique, il est detruit
  // (cas de l'arret de la production).
  //
  pEventTab* eventsToDelete = new pEventTab();
  pCurrentSim->GoHeadInAgenda();
  while(pCurrentSim->IsCurrentInAgenda()) {
    Event* pEvt = pCurrentSim->GetCurrentInAgenda();
    Process* pP = pEvt->FirstProcess();
    if(! strcmp(pP->ClassName(), "alimentationAutomatiqueProcess")) {
      eventsToDelete->push_back(pEvt);
    }
    pCurrentSim->GoNextInAgenda();
  }
  int N = eventsToDelete->size();
  for(int k=0;k<N;k++) {
    Event* pEvtToStop = eventsToDelete->get_element(k);
    pCurrentSim->RemoveFromAgendaEvents(pEvtToStop);
    pEvtToStop->DeleteProcesses();
    pEvtToStop->DeleteInDepth();
  }
  delete eventsToDelete;
}
#include "UserTypes.h"
#include <unistd.h>
//--------------------------------------------
// Fonction 'main'
//--------------------------------------------

main (int argc, char* argv[])
{
     #ifdef _SWIG_
          xs_init = C_App_xs_init;
     #endif

     App_ParseMainArguments(argc, argv);
     ParseParameterFile(argv[1]);
     InitSimulation();
     ParseSimulationFile(argv[2]);
     ParseOutputSpecFile(argv[3]);
     ParseStructureFile(argv[4]);
     ParseDirectiveFile(argv[5]);

     //
     // Observation situation des activites du plan en fin de simulation
     //
     //BasicEntity* pArb = GetFirstBasicEntity(ACTIVITIES_RESOURCES_BLOCK);
     //Activity* pRootAct = (Activity*)(pArb->GetComponent(ACTIVITY));
     //pRootAct->DisplaySituation();

     //
     // Statistiques construction/destruction objets (voir .dir)
     //
```

```
    DeleteEntityInstances();
    DeleteOutputSpecs();
    gTraceNewDelMode = TRACE_NEWDEL_ON;
    MemoryReport();
    gTraceNewDelMode = TRACE_NEWDEL_OFF;
    DeleteEntityClasses();
    delete pCurrentSim;
    // fin Statistiques

    #ifdef _INTERACTIVE_
        printf("FIN MAIN");
    #endif
    return 0;
}
```

Annexe 3 : fichiers arguments de la simulation sim1.s1

```
// ----------------------------------
// System parameters
// ----------------------------------
//TRACE PARSING;

     "RANDOM_GENERATOR" "seed"   <- 15833 ""; //0 ""; //

     "CONVOYEUR"   "longueur"    <- 10 "";
     "CONVOYEUR"   "probaPetite" <- 0.50 "";

     "CUVE"        "debitAlim"   <- 6.0 "";
     "CUVE"        "debitVidange"<- 5.0 "";
     "CUVE"        "Volume" <- 50.0 "";
     "CUVE"        "VolumeMiniContenu"     <- 5.0 "";
     "CUVE"        "VolumeMaxiContenu"     <- 45.0 "";
     "CUVE"        "VolumeInitial"         <- 45.0 "";

     "ATELIER" "rythmeAlimAuto"   <- 5 "";

     "REBUT"       "coeffReducProba"  <- 0.8 "";
     "REBUTPETITES""seuilMaxi"   <- 10 "";
     "REBUTGRANDES""seuilMaxi"   <- 10 "";

     "SIMULATION"  "heureDebut"  <- 6 "";

     "POSTE"       "duree"       <- 240 "";
     "POSTE"       "heureDebutAM"<- 8 "";
     "POSTE"       "heureDebutPM"<- 14 "";

     "MINIBRIQUE"  "Volume" <- 1.0 "";
     "MAXIBRIQUE"  "Volume" <- 2.0 "";

     "CUBITAINER"  "Volume" <- 10.0 "";

// ----------------------------------
// Simulation parameters
// ----------------------------------
//TRACE PARSING;

INITRAND <pi><"RANDOM_GENERATOR" "seed">;

INITIALISATION SIMULATION
     UNITE_TPS MINUTE; //DAY HOUR SECOND
     NB_UNITE_TPS 1;
     DATE_DEBUT 1 JAN 07 <pi><"SIMULATION" "heureDebut"> 0 0;
     DATE_FIN   2 JAN 07;
     IN_DIR "/home/rellier/APPLIS_DIESE/KBS/cdiese/PackMan_SD/in/";
     //IN_DIR "../in/";
     OUT_DIR "/home/rellier/APPLIS_DIESE/KBS/cdiese/PackMan_SD/out/";
     //OUT_DIR "../out/";
;

// ----------------------------------
// Spécificatons de sortie
// ----------------------------------
//TRACE PARSING;

// -------------------------------
// SAVE DESCRIPTOR
// -------------------------------
SAVE DESCRIPTOR    "magasinExpedition"     END  "production1.txt"  APPEND    CLOCK
          "productionMaxi"   0
          "productionMini"   0;

//SAVE DESCRIPTOR  "cuve"             ALL  "niveauCuve1_jhm.txt"   NEW  DATE_JHM
//        "volumeContenu"    0;

SAVE DESCRIPTOR    "cuve"             ALL  "niveauCuve1_clock.txt" NEW  CLOCK
          "volumeContenu"    0;


// ----------------------------------
//Structure du systeme
// ----------------------------------
//TRACE PARSING;

// ****************************************************************
// le systeme technique
// ****************************************************************

+ I robinetAlim leRobinetAlim
     debit = <pr><"CUVE" "debitAlim">;
;

+ I robinetVidange leRobinetVidange
```

```
     debit = <pr><"CUVE" "debitVidange">;
;

+I generateurDeBriques leGenerateur
     probaPetiteBrique = <pr><"CONVOYEUR" "probaPetite">;
;

+ I reserveCubis laReserveCubis
          POUR i3 = (1) A (10)
               + E cubitainer,;
          FIN POUR
;

+ I magasinExpedition leMagasin
     + E loge lesPetitsPacks
          contenance = 100;
     ;
     + E loge lesGrandsPacks
          contenance = 100;
     ;
;

+ I rebut leRebut
     + E logeRebut lesPetitsRebuts
          contenance = 50;
     ;
     + E logeRebut lesGrandsRebuts
          contenance = 50;
     ;
;

+ I maxiBrique uneMaxiBrique;

{i1} = <pi><"CONVOYEUR"  "longueur">;

+ I atelierRemplissage latelier,
     + C cuve laCuve
          <- C <I><robinetAlim,>;
          <- C <I><robinetVidange,>;
          volume = <pr><"CUVE" "Volume">;
          volumeMiniContenu = <pr><"CUVE" "VolumeMiniContenu">;
          volumeMaxiContenu = <pr><"CUVE" "VolumeMaxiContenu">;
          volumeContenu     = <pr><"CUVE" "VolumeInitial"> no_monitor;
     ;
     <- C <I><, laReserveCubis>;
     + C convoyeur leConvoyeur
          convoyeurGenerateur = <I><generateurDeBriques,>;
          convoyeurPlateforme << <I><, uneMaxiBrique>;
          POUR i2 = (2) A ({i1})
               convoyeurPlateforme << I positionVide,; ;
          FIN POUR
     ;
     <- C <I><, leMagasin>;
     <- C <I><, leRebut>;
;

// ********************************************************************
// le systeme opérant
// ********************************************************************

+ I ouvrierOperant,;

+ I ouvrier lOperateurConvoyeur
     availabilityStatus = 0;
     power = 0.25;
;

+ I simpleResourcePool lePoolOuvrier,
     <- E <I><, lOperateurConvoyeur>;
;

+ I resourcePoolDisjunction lesPools,
     <- E <I><, lePoolOuvrier>;
;

+ I operatingSystem pOS,
     <- C <I><, lesPools>;
;

// ********************************************************************
// la strategie (vide) du manager
// ********************************************************************

+ I activityIteration iterationPlagesTravail;

+ I activitiesResourcesBlock arb1
     <- C <I><, iterationPlagesTravail>;
     <- C <I><, lePoolOuvrier>;
;
+ I activitiesResourcesBlockSet arbSet
     <- E <I><,arb1>;
```

```
;
+ I strategy laStrategie
     <- C <I><,arbSet>;
;

// ************************************************************************
// spécification du système de production
// ************************************************************************

+ I controlledSystem cs
     attachedEntity = <I><,latelier>;
;

+ I manager leManager
     <- C <I><, laStrategie>;
;

+ I productionSystem ps,
     <- C <I><controlledSystem,>;
     <- C <I><operatingSystem,>;
     <- C <I><manager,>;
;

ENTITE_SIMULEE <I><,ps>;

// ********************************************************************
// compléments sur la strategie
// ********************************************************************

//---------------------
// spécifications
//---------------------

+ SpE lesAteliers;

+ SpE lePackDeTete;

+ I operatedObjectSpecification lAtelierSpec,
     objectEntitySpecAttribute = <SpE><lesAteliers>;
     selectorFctId = 1; // ANYONE=1, MAX=2, ALL=3, DISJ=4, SETS=5
;

+ I operatedObjectSpecification leConvoyeurSpec,
     preExpandedList << <I><, leConvoyeur>;
;

+ I operatedObjectSpecification leRebutSpec,
     preExpandedList << <I><, leRebut>;
;

+ I operatedObjectSpecification lePackSpec,
     objectEntitySpecAttribute = <SpE><lePackDeTete>;
     selectorFctId = 1; // ANYONE=1
;

//---------------------
// activités primitives
//---------------------

+ I miseEnRoute laMiseEnRoute
/* objet opéré spécifié dans le constructeur */
;

+ I creationPack uneCreationPack
     operatedObjectSpecAttribute = <I><, leConvoyeurSpec>;
;

+ I remplissagePack unRemplissagePack
     operatedObjectSpecAttribute =
          + I operatedObjectSpecification,
               preExpandedList << <I><, laCuve>;
          ;
     ;
     destinationSpecAttribute =
          <I><, lePackSpec>;
     transferInformationAttribute =
          + I infoTransfertCuvePack info1;
     ;
;

+ I fermetureBriquesPack desFermeturesBriques
     operatedObjectSpecAttribute = <I><, lePackSpec>;
;

+ I conditionnementPack unConditionnementPack
     operatedObjectSpecAttribute = <I><, lePackSpec>;
/* performer spécifié dans le constructeur */
;

+ I stockagePack unStockagePack
     operatedObjectSpecAttribute = <I><, leConvoyeurSpec>;
```

```
;

+ I miseAuRebut uneMiseAuRebut
     operatedObjectSpecAttribute = <I><, leRebutSpec>;
;

+ I arretProduction lArretProduction
     operatedObjectSpecAttribute = <I><, lAtelierSpec>;
;

+ I alimentationManuelle uneAlimManuelle
     operatedObjectSpecAttribute =
          + I operatedObjectSpecification,
               preExpandedList << <I><, laReserveCubis>;
          ;
     ;
;

//---------------------
// activités agrégées
//---------------------

+ I activityOptional alimManuelleIfNeeded
     <- E <I><, uneAlimManuelle>;
     isOneShot = 1;
;

+ I sequenceMiseEnMagasin uneMiseEnMagasin
     <- E <I><, uneCreationPack>;
     <- E <I><, unRemplissagePack>;
     <- E <I><, desFermeturesBriques>;
     <- E <I><, unConditionnementPack>;
     <- E <I><, unStockagePack>;
;

+ I iterationMagasinOuRebut laProduction
     + E activityConjunction unCycleDeProduction,
          + E activityDisjunction magasinOuRebut,
                    <- E <I><, uneMiseEnMagasin>;
                    //
                    <- E <I><, uneMiseAuRebut>;
          ;
          <- E <I><,alimManuelleIfNeeded>;
     ;
;

//---------------------
// finalement, le plan
//---------------------
<I> <,iterationPlagesTravail>
     + E activityBefore sequencePlageTravail,
          <- E <I><, laMiseEnRoute>;
          //
          <- E <I><, laProduction>;
          //
          <- E <I><, lArretProduction>;
     ;

          readyToRun = 1;
;


// ************************************************************************
// la dynamique du système
// ************************************************************************

+ P fluxEntreeCuveLectureProcess lectureFluxEntree
     PAS = 5;  // 5 minutes
     NOM_SIMPLE_FICHIER "flux1.txt";
     INIT_PRCD_EVT  DATE_OCCUR = 0;
               PRIORITE = 0;
     ;
;

// -------------------------------------
// on programme la première arrivée de l'ouvrier
// -------------------------------------
{i2} = <pi><"POSTE" "heureDebutAM"> - <pi><"SIMULATION" "heureDebut">;
+ V arriveeOuvrier debutPoste  // on veut que ce soit à 'heureDebutAM', donc ...
     DATE_OCCUR = {i2}*60;  // UT = 1 minute
     I resourceMobilizationProcess = <I><ouvrier lOperateurConvoyeur>;
// Précision ci-dessus : explicite mais non nécessaire parce que, par défaut,
// il y a un ProcessedEntityClassId(RESOURCE) dans le constructeur de
// ResourceMobilizationProcess. L'entité opérée est donc, par défaut,
// la première instance de RESOURCE. Ici, c'est aussi la seule : l'ouvrier.
;

// -------------------------------------
// on programme l'interprétation du plan
// -------------------------------------
+ V "updateSituationEvent" maj
```

```
     DATE_OCCUR = 0;
     MIN_DELAY_NEXT = 1;
     MAX_DELAY_NEXT = 1;
;


// ------------------------------------
// Ce fichier contient les directives de simulation.
// Il doit etre ajuste au besoin, selon les regles enoncees dans le chapitre
// 'Les fichiers' de la documentation de BDIESE.
// ------------------------------------
//TRACE PARSING;
//TRACE NEWDEL;
//TRACE ACTION;

//INITRAND;

CHRONO INIT;
CHRONO START;

RUN;

DISPLAY OUTPUTSPEC EVENT;
DISPLAY OUTPUTSPEC PROCESS;
DISPLAY OUTPUTSPEC CPROCESS;
DISPLAY OUTPUTSPEC ENTITE basicEntity;
//DISPLAY OUTPUTSPEC ALL;

DELETE ENTITYSPEC;
DELETE MONITOR;
DELETE FILE;
DELETE PARAMETRE;
//
// Inhiber les 5 lignes ci-dessous si main.cc execute :
// - Observation situation des activites du plan
// - et/ou Statistiques construction/destruction objets
//
//DELETE ENTITE INSTANCE;
//DELETE OUTPUTSPEC;
//RAPPORT MEMOIRE;  // ssi gTraceNewDelMode = TRACE_NEWDEL_ON
//DELETE ENTITE CLASSE;
//DELETE SIMULATION;

//DELETE ALL;

CHRONO PAUSE;
CHRONO MICRO DISPLAY ENDL "duree = "; MESSAGE ENDL;
CHRONO CLOSE;
```

Annexe 4 : contenu du fichier de données séquentielles

A la suite des deux lignes d'entête, il y a une ligne pour chaque tranche de 5 minutes à partir de 0 heures jusqu'à 23h55. Dans chaque ligne, il y a 3 données : l'heure, la minute et la quantité. Les quantités successives résultent de la répétition de la série arbitraire suivante : 10. 7. 8. 8. 10. 7. Le fichier commence et finit donc ainsi :

```
// serie des quantites disponibles en entee de la cuve
//-----------------------------------------------------
0    0    10.
0    5    7.
0    10   8.
0    15   8.
0    20   10.
0    25   7.
0    30   10.
0    35   7.
0    40   8.
0    45   8.
0    50   10.
0    55   7.
1    0    10.
// ...
23   30   10.
23   35   7.
23   40   8.
23   45   8.
23   50   10.
23   55   7.
```