Introduction à Modgen

par

Claude Charette

Claude.Charette@statcan.ca

Statistique Canada

Atelier pour les utilisateurs de Modgen

27 mai 2008

(révisé par Chantal Hicks, mai 2017)

www.statcan.gc.ca/spsd/Modgen_f.htm

Table des matières

Introduction 3
Installation de Modgen 4
  Composants préalables à Modgen 12 4
  Modgen 12 4
Composants de Modgen 6
  Pré-compilateur 6
  Bibliothèque 7
Bilinguisme 7
  Pré compilateur 8
  Modèles 8
Éléments d'un modèle Modgen 9
  Contenu d'un modèle 9
    Acteurs 9
    Ensembles d'acteurs 14
    Liens 15
    Paramètres 15
    Tableaux 16
    Groupes de tableaux ou de paramètres 16
    Types 17
  Déroulement d'un modèle 18
    Lecture des paramètres 18
    Validation des paramètres 18
    Pré-simulation 18
    Simulation 19
    PostSimulation 21
    Tabulation 21

Introduction

Modgen a été conçu pour faciliter la programmation de modèles de microsimulation. Son but est d'enlever autant que possible les obstacles à la création d'un modèle de microsimulation. Voici quelques uns de ces obstacles.

- Programmation de l'interface
- Documentation
- Programmation de l'engin de simulation
- Bilinguisme

Ces obstacles sont tous supprimés par l'utilisation de Modgen puisque Modgen fournit l'interface et l'engin de simulation, est bilingue et facilite la documentation. Évidemment, cette liste n'est pas exhaustive et n'est donnée qu'à titre d'exemple.

Une conséquence de l'utilisation de Modgen pour créer un modèle de microsimulation est la non nécessité d'engager un programmeur pour programmer le modèle. En effet, Modgen s'occupe de la majorité de la programmation. Ce qu'il reste est normalement suffisamment simple pour qu'une personne n'ayant pas de connaissances approfondies en programmation puisse le faire. Il se peut même que le fait d'avoir des connaissances approfondies en programmation soit nuisible puisque le concepteur doit alors se limiter à un style de programmation qui sera comprise par l'analyste responsable du modèle.

Aussi, il n'est pas nécessaire pour utiliser Modgen de comprendre en détail ce qu'il fait. Toutefois, il peut être utile d'avoir une idée générale de ce que fait Modgen. C'est ce que tente de faire le présent document.

Installation de Modgen

Il y a deux produits, et donc deux programmes d'installation, reliés à Modgen. Ces produits sont :

- Composants préalables à Modgen 12
- Modgen 12

Voici ce que chacun de ces produits contient ainsi qu'une description de leur utilité.

Composants préalables à Modgen 12

De façon générale, les Composants Préalables à Modgen 12 contiennent tout ce qui est nécessaire pour que les modèles fonctionnent sur une machine. Si un concepteur souhaite distribuer un modèle, il faut donc que les utilisateurs du modèle installent les Composants Préalables à Modgen 12 avant d'installer le modèle. Ce produit contient :

- Licence pour l'utilisation des modèles Modgen
- Documentation pour l'utilisateur des modèles Modgen :

  Cette documentation concerne les utilisateurs des modèles. Elle ne fournit aucune aide pour la création des modèles. De façon générale, il s'agit de l'information concernant l'interface des modèles. Cette documentation comprend :

  - Guide de l'interface visuelle de Modgen 12
  - Notes de publication des Composants Préalables à Modgen 12

- Application Cleaner

  Cette application nettoie les fichiers temporaires oubliés sur une machine lorsqu'une simulation d'un modèle échoue. Si la simulation termine avec succès, il ne reste aucun fichier temporaire. Elle est installée avec les Composants Préalables à Modgen 12 et est exécutée à chaque nouvelle session Windows. Il est également possible de la démarrer manuellement mais alors, il faut éviter de le faire lorsqu'une simulation est en cours.

- Serveur automatisé de Modgen

  Un composant permettant à un autre programme la lecture en l'enregistrement des fichiers des paramètres d'un scénario.

- Composants préalables de Microsoft

  Quelques fichiers fournis par Microsoft qui sont requis pour tourner un modèle Modgen.

Modgen 12

De façon générale, Modgen 12 contient ce qui est nécessaire au développement de modèles. Évidemment, Visual Studio 2015 ou 2017 est nécessaire et n'est pas inclus dans Modgen 12! Précisément, ce produit contient :

- Les Composants Préalables à Modgen

- Licences pour la création des modèles de Modgen

- Ce qui est nécessaire à la création d'un nouveau modèle :

  Cela inclut le pré-compilateur Modgen, la bibliothèque et des fichiers d'entête.

- Documentation pour le concepteur de modèles :
  Cette documentation concerne les concepteurs de modèles. Elle fournit donc de l'aide concrète sur la création de modèles, la syntaxe à utiliser, etc. Cette documentation comprend :

  - Guide du concepteur de Modgen 12
  - Notes de publication de Modgen 12

- Capacités d'intégration avec Visual Studio

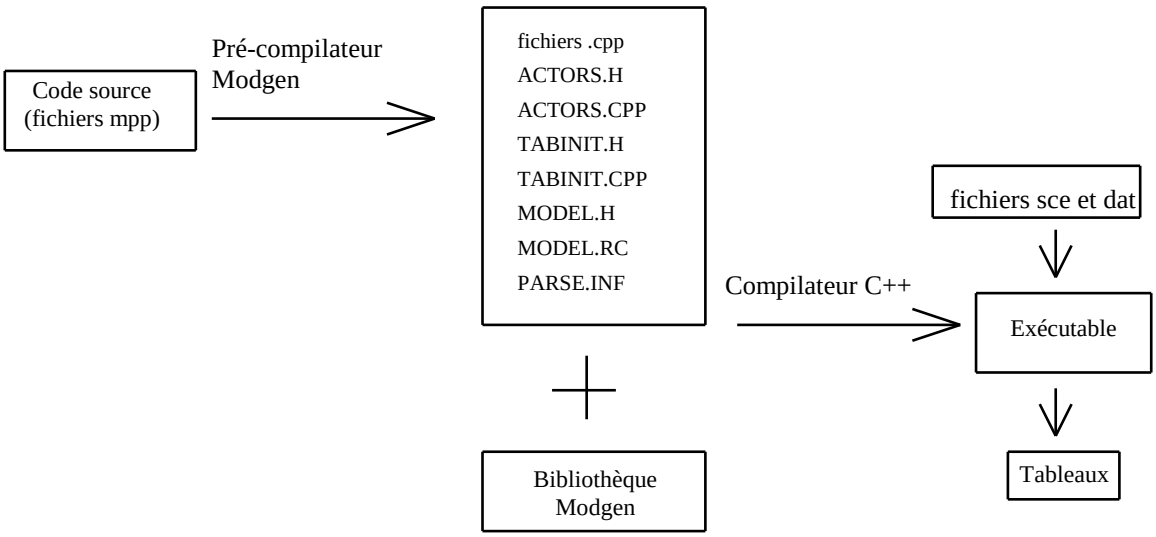
Composants de Modgen

Il y a trois composants principaux qui forment Modgen. Ces composants sont :

- Pré-compilateur Modgen
- Bibliothèque Modgen
- Serveur automatisé

Les deux premiers composants sont utilisés ensemble pour créer des modèles et seront décrits dans le document. Le serveur automatisé, qui permet d'accéder aux fonctions de lecture et d'écriture de fichiers de paramètres, ne sera pas discuté dans le présent document.

En rappel, le schéma suivant illustre les liens entre les fichiers nécessaires à la création d'un fichier exécutable de modèle de simulation dans le cadre de l'application. Il présente aussi les entrées et les sorties dont a besoin le fichier exécutable du modèle.



Notons que Modgen n'est pas indépendant de Visual Studio. Les concepteurs de modèles doivent avoir la version de Visual Studio correspondante afin de pouvoir créer les modèles (Visual Studio 2015 ou 2017 pour Modgen 12).

Pré-compilateur

Le pré-compilateur est l'exécutable « Modgen.exe ». Le langage Modgen dans lequel un concepteur écrit un modèle est une extension du langage C++. Avant d'être compilé, le code des modèles doit être transformé en code C++. C'est ce que fait le pré-compilateur. Il lit les fichiers .mpp contenant le code du modèle et crée les fichiers .cpp équivalents, en plus des fichiers spéciaux nécessaires à la création du modèle. De façon générale, le pré-compilateur Modgen a la responsabilité de comprendre le modèle de données du modèle voulu et de l'implanter. Voici une description plus précise de ce que fait Modgen, à l'étape de pré-compilation :

- Vérifie que la syntaxe est correcte pour tout élément du langage Modgen. La syntaxe C++ sera par la suite vérifiée par le compilateur C++.

- Vérifie que les fonctions essentielles comme « Simulation » sont présentes dans le code du modèle.

- Comprend les acteurs et crée les classes en rassemblant toutes les déclarations des acteurs éparpillées dans les fichiers .mpp

- Comprend les tableaux voulus et crée les classes pour les tableaux

- Comprend les relations entre les états et les autres symboles (états dérivés, événements, tableaux) et crée le code nécessaire pour maintenir ces relations

- Crée le code nécessaire pour mettre à jour le temps pour chaque acteur

- Crée une structure interne permettant au modèle de créer un système d'aide du modèle au besoin et de façon autonome.

Notons que cette liste n'est pas exhaustive. Le pré-compilateur est appelé automatiquement lorsqu'un mode est compilé.

Bibliothèque

La bibliothèque de Modgen contient les éléments que tous les modèles ont en commun. Lors de la compilation, des liens seront faits avec la bibliothèque Modgen de façon à l'incorporer dans l'exécutable du modèle. De cette façon, chaque modèle est un exécutable indépendant. Entre autres, elle contient :

- l'interface de tout modèle Modgen

- l'engin de simulation

- le générateur d'aide

Le langage C++ requiert quelques fichiers d'entête pour incorporer la bibliothèque dans un modèle. Ces fichiers sont installés avec Modgen. En plus de la bibliothèque de Modgen, il y a d'autres bibliothèques et fichiers d'entêtes de Microsoft qui sont fournis avec Visual Studio et incorporés dans le modèle.

Bilinguisme

La loi canadienne exige que toute application logicielle créée par un organisme du gouvernement du Canada soit bilingue, c'est-à-dire que le français et l'anglais soient traités également. Modgen étant une application logiciel créée à Statistique Canada, il se doit d'être bilingue. Toutefois, dans le cas de Modgen, il y a plus d'un niveau au bilinguisme requis. De fait, Modgen est une application qui permet de créer des applications. Il est donc nécessaire que Modgen soit bilingue, mais aussi qu'il permette de créer des applications bilingues.

Pré compilateur

Modgen lui-même est bilingue. Le français et l'anglais sont traités aussi également qu'il est possible de le faire. La langue choisie soit nécessairement être donnée en argument dans la ligne de commande du pré compilateur.

Exemple :

c:\Program Files\StatCan\Modgen 12\Modgen.exe –FR

En pratique, le pré compilateur Modgen est appelé lorsqu'un modèle est compilé. On peut changer la langue choisie de Modgen avec un utilitaire installé avec Modgen qui s'appelle Langue Modgen Language.

Modèles

Modgen contient tout ce qui est nécessaire pour que l'interface des modèles soit bilingue, c'est-à-dire en français et en anglais. De plus, il est possible et très simple pour les concepteurs de modèles d'utiliser d'autres langues que le français et l'anglais. Il suffit alors de traduire l'un des fichiers contenant toutes les chaînes de caractères de Modgen (ModgenEN.mpp, ModgenFR.mpp) et de l'inclure dans le modèle comme module.

Même si Modgen est bilingue, l'interface n'aura que la(les) langue(s) définie(s) dans le modèle. Pour obtenir le bilinguisme dans un modèle, il faut définir le modèle comme étant bilingue. Voici comment définir les langues utilisées au gouvernement du Canada :

```
// Voici les langues dans lesquelles le modèle peut être vu
languages
{
      EN,    // English
      FR     // Français
};
```

Il ne faut pas oublier que tout modèle distribué par le gouvernement du Canada doit être bilingue. Il se peut qu'il ne soit pas clair si un modèle est un logiciel ou un produit de données. Peu importe, puisque dans les deux cas, il doit être entièrement bilingue s'il est produit par un organisme du gouvernement du Canada.

Éléments d'un modèle Modgen

Modgen permet de créer deux types de modèles différents :

- Modèles de cas

  Un modèle de cas est un modèle dans lequel la simulation se déroule cas par cas. Un cas est alors typiquement constitué d'un acteur principal avec, au besoin, quelques acteurs secondaires formant son entourage. Toutefois, dans certains modèles, il est possible d'avoir plus d'un acteur principal par cas si, par exemple, un cas simule une famille.

  Le nombre de cas à simuler est défini comme paramètre de contrôle de l'exécution. Les modèles « LifePaths », « PopSim », et « Pohem » sont des exemples de modèles de cas.

- Modèles basés sur le temps

  Un modèle basé sur le temps est un modèle dans lequel est simulé une population, ensemble, pendant un certain temps. La durée de la simulation est alors définie comme paramètre de contrôle de l'exécution. Les modèles « CVMM » et « HIVMM » sont des exemples de modèles basés sur le temps.

Contenu d'un modèle

Les modèles de cas et les modèles basés sur le temps sont constitués de symboles Modgen. Ce sont ces symboles et les relations entre eux qui gèrent la simulation. Une courte description de chaque catégorie de symboles est donnée ici. Pour une description plus détaillez, veuillez vous référer au Guide du concepteur de Modgen 12.

Acteurs

Tout d'abord, les modèles Modgen simulent la vie d'un acteur. Un acteur peut être n'importe quelle entité que le concepteur de modèle souhaite simuler. Ce peut être une personne, un logement, un plan de pension, etc.

Les acteurs sont définis par leurs :

- États
- Événements
- Fonctions
- Crochets

États

Les états décrivent les caractéristiques de l'acteur. Il existe deux grandes catégories d'états dans un modèle Modgen :

- États simples :

  Les états simples sont les états dont la valeur n'est pas maintenue automatiquement par Modgen. Plutôt, la valeur des états simples est modifiée dans le code créé par le concepteur de modèle. Les états simples ne doivent être modifiés qu'à l'intérieur d'un événement, ou d'une fonction appelée à l'intérieur d'un événement.

  Exemple :

```
int   age_int ;   //FR Âge intègre
```

  Les états simples peuvent avoir une valeur initiale. Afin de les distinguer des états dérivés, on assigne une valeur initiale à un état simple en utilisant des accolades.

  Exemple :

```
int   age_int = {0} ;   //FR Âge intègre
```

- États dérivés :

  Les états dérivés sont les états dont la valeur est indiquée sous forme d'expression dans la déclaration. Modgen maintient la valeur de ces états tout au long de la simulation. Habituellement, ces états sont dérivés d'autres états, d'où leur nom.

  Exemple :

```
//FR Année
MODELED_TIME modeled_year = COERCE(MODELED_TIME, year);
```

  En plus des expressions simples comme dans l'exemple précédent, il existe également des fonctions que le concepteur de modèle peut choisir d'utiliser pour créer des états dérivés.

  Exemple :

```
//FR Âge au début de l'année
int   age_debut_annee = value_at_latest_change( annee, age_int);
```

Généralement, les états sont déclarés de type :

- Entier (int)
- Réel (float ou double)
- Logique (logical)
- Temps (TIME)
- Avec un type créé dans le modèle (classification, étendue, partition)

Événements

Les événements ont un rôle clé dans les modèles Modgen puisque c'est par l'exécution d'événements que la simulation est faite. Ces événements sont constitués d'une paire de fonctions :

- fonction pour déterminer le temps de l'événement
- fonction pour déterminer les conséquences de l'événement

Le temps des événements est mis à jour au besoin. Lorsque le temps doit être recalculé, Modgen évalue de nouveau la valeur de la fonction temps de l'événement. Modgen suppose que le temps de l'événement doit être recalculé lorsque :

- un des états utilisés dans la fonction temps de l'événement a changé de valeur
- l'événement a eu lieu.

La fonction temps de l'événement ne devrait pas affecter les acteurs. Cela implique que les états ne peuvent en aucun cas être modifiés à l'intérieur d'une fonction temps d'un événement. Le pré-compilateur Modgen veille à ce que ça soit respecté.

C'est tout le contraire pour la fonction de l'événement. La fonction qui détermine les conséquences de l'événement est en fait ce qui est exécuté lorsque l'événement a lieu. C'est à l'intérieur des fonctions d'événements que les états sont modifiés. Un état ne devrait jamais être modifié à l'extérieur d'un événement, ou d'une fonction appelée par la fonction de l'événement, une fois l'initialisation terminée. Modgen veille à ce que ça soit respecté.

Fonctions

Il est possible d'utiliser des fonctions membres d'un acteur pour généraliser une section de code qui sera utilisée dans un ou plusieurs événements. De façon générale, il est nécessaire de créer une fonction lorsqu'une partie de code sera réutilisée à plusieurs endroits. Dans un tel cas, l'utilisation d'une fonction simplifie la maintenance puisque lorsque le code doit être modifié, il ne doit être modifié qu'à un seul endroit. Ne pas utiliser une fonction dans ces cas là augmente le risque éventuel d'introduire des bogues. Parfois, on peut souhaiter utiliser une fonction même si le code n'est pas réutilisé, soit parce qu'il le sera éventuellement, soit afin d'obtenir plus de modularité.

En plus des fonctions créées par le concepteur du modèle, il existe des fonctions que tout acteur doit posséder. Il s'agit des fonctions « Start » et « Finish »

Fonction « Start »

La fonction « Start » doit être présente pour chaque acteur. Elle sera appelée une fois pour chaque acteur, lors de sa création. Elle contient l'initialisation de l'acteur, y compris l'initialisation faite par Modgen. Notons qu'un état initialisé dans la fonction « Start » n'affecte ni les tableaux, ni les états dérivés tels que « entrances() ». Pour cette raison, il vaut mieux faire ces initialisations dans la fonction « Start » et non dans la fonction qui crée l'acteur. Si le concepteur de modèle n'a aucune initialisation à faire et n'a pas inclus la fonction « Start », Modgen créera une fonction contenant simplement l'initialisation des états aux valeurs de défaut faite par Modgen.

Exemple 1 :

```
actor Personne  //FR Personne
{
      void  Start();    //FR Initialise une nouvelle personne
};


void Personne::Start()
{
}
```

Exemple 2 :

```
actor Personne  //FR Personne
{
      //FR Commence Personne
      void  Start( int nObs, logical lImmigrant, double dTempsDebut ,
                  logical lEnfant, Personne *prMere );
};

/*NOTE(Personne.Start, FR)
      Cette fonction commence la personne.
*/
void Personne::Start( int nObs, bool bImmigrant, TIME tTempsDebut,
                        bool bEnfant)
{

      immigre_apres_recensement = bImmigrant;
      num_recense = nObs;

      ne_apres_recensement = bEnfant;

      time = CoarsenMantissa(tTempsDebut);
...

      vivant = TRUE;
      emigrant = FALSE;


      // Année
      annee = (int) time;
};
```

Dans l'exemple précédent, les états « vivant » et « emigrant » auraient dû être initialisés à même leur déclaration. De façon générale, les états initialisés dans la fonction « Start » sont les états dont la valeur n'est pas constante, mais peut dépendre, par exemple, des valeurs données en argument à la fonction.

Fonction « Finish »

Il existe aussi une fonction « Finish » pour contenir tout ce qui doit être fait lorsqu'un acteur est détruit. Par exemple, on peut vouloir que tous les acteurs auxquels l'acteur est lié soient détruits également. Si le concepteur de modèle n'a rien de particulier à faire avant que l'acteur soit détruit et a omis la fonction « Finish », alors Modgen créera une fonction de base

automatiquement. Notons que Modgen détruit tout lien aux autres acteurs lorsque la fonction « Finish » est appelée, sans toutefois appeler la fonction « Finish » de ces acteurs.

Exemple :

```
actor Personne  //FR Personne
{
      void  Finish();   //FR Termine Personne
};

void Personne::Finish()
{
    // Vide pour le moment
}
```

Crochets

Les crochets permettent de relier des fonctions à des événements ou à d'autres fonctions. Cela permet de diviser un événement ou une fonction en différentes sections qui appartiennent à des modules différents.

En particulier, on utilise beaucoup les crochets avec les fonctions « Start » et « Finish » afin de faire l'initialisation qui se rapporte à un module à l'intérieur de ce module même. Par défaut, tous les crochets s'insèreront à la fin d'une fonction. Si on souhaite qu'ils soient tous insérés à un autre endroit, on peut le faire en définissant « IMPLEMENT_HOOK » à l'endroit approprié de la fonction.

Voici un exemple où les crochets s'insèreront au début de la fonction :

```
/* NOTE(Person.Finish,FR)
      Fonction qui nettoie l'acteur, une fois qu'il a terminé.

*/
void Person::Finish()
{
      Person      *prChild = {NULL};
      int          nIndex = {0};

      IMPLEMENT_HOOK();

...

      if ( tentative && sex == FEMALE )
      {
            mlChildren->FinishAll();
            mlChildrenAtHome->FinishAll();
            mlBiologicalChildren->FinishAll();
      }
}
```

Notons que dans le cas de la fonction « Finish », il fait plus de sens d'avoir les crochets au début de la fonction puisqu'à la fin de la fonction, les propriétés de l'acteur ne sont plus valides.

Il arrive aussi qu'on veuille insérer quelques crochets à un endroit différent des autres. On peut alors créer une fonction intermédiaire à laquelle se raccrochent les crochets, et appeler cette fonction au bon endroit.

Exemple :

```
actor Person      //FR Individu principal
{
      void  Start( ...);      //FR Fonction qui lance l'acteur Personne
      void  StartClockHere(); //FR Fonction fictive Start Clock

};

/* NOTE(Person.StartClockHere,FR)
      Fonction fictive qui permet d'initialiser l'horloge avant toutes
      les autres fonctions accrochées et fonctions dérivées.
*/
void Person::StartClockHere()
{
}

void Person::Start( ... )
{

...
      time = birth;
      StartClockHere();
...
      year_of_birth = year;
      month_of_birth = month_of_year;
      day_of_birth = day_of_month;
...
}
```

Dans cet exemple, la fonction « StartClockHere » sera exécutée immédiatement après que le temps ait été initialisé. Les autres crochets à la fonction « Start » ne sont exécutés qu'à la fin de la fonction. Ça leur permet d'utiliser les états de l'horloge tels que « year », « month », etc. pour initialiser d'autres états. La fonction « StartClockHere » ne contient pas de code, mais des crochets s'y rattachent :

```
actor Person      //FR Individu principal
{
      //FR Initialise l'horloge à la naissance de l'acteur
      void StartClock();

      hook StartClock, StartClockHere;

};
```

C'est la fonction « StartClock », définie dans un autre module, qui contient le code qui initialise l'horloge et ses états, comme « year », « month », etc.

Ensembles d'acteurs

Les ensembles d'acteurs sont des collections d'acteurs maintenues dynamiquement par Modgen. Cela signifie que Modgen s'occupe de l'adhésion au groupe selon le critère fourni par Modgen.

Les ensembles d'acteur peuvent être multidimensionnels. Dans ce cas, les dimensions sont constituées d'états des acteurs regroupés en ensembles. Pour chaque combinaison de valeur des dimensions, un sous-ensemble est créé. L'appartenance d'un acteur à un seul sous-ensemble est automatiquement maintenue par Modgen. Pour en savoir davantage sur les ensembles d'acteurs, vous pouvez vous référer au Guide du concepteur de Modgen.

Liens

Les liens définissent des relations entre acteurs de même type ou de type différents. Il peut y avoir des liens d'un acteur à un seul autre acteur, d'un acteur à plusieurs acteurs, ou de plusieurs acteurs à plusieurs acteurs. Dans tous les cas, les liens réciproques sont automatiquement créés et maintenus par Modgen. Cela implique que le code du modèle n'a qu'à assigner l'un des liens et le lien réciproque sera modifié pour refléter le changement.

Exemples de liens :

```
// Déclare le lien un-à-plusieurs pour l'appartenance à une famille
link
      Person.lFamily      //FR Famille
      Family.mlMembers[]  //FR Membres de la famille
;


// lien entre parents et enfants
link
      Person.mlChildren[]     //FR Enfants
      Person.mlParents[]      //FR Parents
;

// lien entre la personne et son(sa) conjoint(e)
link Person.lSpouse;          //FR Conjoint(e)
```

Pour de plus amples renseignements sur les liens, vous pouvez vous référer au Guide du concepteur de Modgen 12.

Paramètres

Les paramètres permettent à l'utilisateur du modèle d'avoir un certain contrôle sur la simulation. Les paramètres devraient être utilisés partout où le concepteur souhaite donner du contrôle à l'utilisateur du modèle. Le fait de permettre à l'utilisateur du modèle de préciser des hasards pour contrôler différents aspects de la simulation, par exemple, donne à l'utilisateur la possibilité d'explorer différents scénarios.

Il existe deux grandes catégories de paramètres :

- paramètres

  Il s'agit de la catégorie la plus fréquente. On utilise un paramètre lorsqu'on laisse à l'utilisateur le soin d'entrer une valeur. Les valeurs des paramètres se trouvent dans les fichiers .dat. L'utilisateur contrôle donc directement la valeur de ces paramètres.

  Exemple :

  ```
  parameters
  ```

```
{

      //FR hazard total de mortalité (m total)
      double            MortHazard[MODELED_AGE][SEX];

};
```

- paramètres générés par le modèle

  Il s'agit de paramètres qui doivent être dérivés, souvent à partir d'autres valeurs données en paramètres. Pour chaque paramètre généré par le modèle, il doit y avoir du code correspondant dans la fonction « PreSimulation » décrite à la section suivante.

  Exemple :

```
parameters
{

      //FR Hasards de mortalité résiduels
      model_generated double
            ResidualMortHazard[MODELED_AGE][SEX];

};
```

Tableaux

Les tableaux forment la sortie des modèles Modgen. Il existe deux types différents de tableaux :

- tableaux

  Les tableaux croisés d'un modèle génèrent des résultats agrégés d'une simulation.

- tableaux d'utilisateurs

  Ces tableaux sont mal nommés. Il ne s'agit pas ici des utilisateurs des modèles mais des utilisateurs de Modgen. Autrement dit, il s'agit des concepteurs de modèles!

  Les tableaux d'utilisateurs permettent aux concepteurs de modèle de créer du code qui calcule la valeur de chaque cellule. Pour chaque tableau d'utilisateur, du code doit être placé dans une fonction « UserTables » pour contrôler le contenu du tableau. Typiquement ce code lit et combine des valeurs de différents tableaux pour en créer d'autres.

Groupes de tableaux ou de paramètres

Il est possible de regrouper certains symboles en groupes. Ces groupes servent surtout à regrouper les symboles dans l'interface du modèle mais sont également décrits dans l'aide générée par Modgen du modèle. Les symboles qu'il est possible de grouper sont :

- paramètres du modèle
- paramètres générés par le modèle

- tableaux et tableaux d'utilisateurs

L'appartenance d'un symbole à un groupe doit être précisée dans la déclaration du groupe. Notons qu'un groupe peut toujours contenir un autre groupe du même genre, ce qui permet de créer une hiérarchie.

Exemple d'un groupe de paramètres :

```
parameter_group DEMOGRAPHIE  //FR Paramètres démographiques
{
      HasardMortalite, HasardNatalite, AgeMaximal, ProbabiliteGarcon,
      VariationFecondite
};
```

Exemple d'un groupe de tableaux :

```
table_group TableauxDemographiques { //FR Tableaux démographiques
    TauxNaissance,
    Population,
    Population5Ans
};
```

Types

Les symboles de type sont des symboles de support. Ils ne sont pas essentiels à la simulation puisqu'ils ne jouent aucun rôle précis dans le déroulement d'un modèle. Toutefois, ils sont partout puisqu'ils servent à définir d'autres symboles comme les états ou les paramètres. Ils servent également à donner des étiquettes aux tableaux. Il y a trois catégories de symboles de types :

- Classifications

  Ensemble de niveaux ou de catégories avec leurs étiquettes.

  Exemple :

  ```
  classification LANGUE2
  {
        //FR francophone
        L2_FRANCO,
        //FR non francophone
        L2_NON_FRANCO
  };
  ```

- Étendues (« range »)

  Intervalle de nombres entiers.

  Exemple :

  ```
  range AGE_NATALITE {15, 49} ; //FR Âge pour les taux de natalité
  ```

- Partitions

  Ensemble de points limites d'une variable d'état continue (ou discrète).

  Exemple :

```
//FR Groupes d'âge
partition GROUPE_AGE
{
      5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75,
      80, 85, 90, 95, 100
};
```

Les descriptions présentées ici sont tirées du Guide du concepteur de Modgen 12. Vous pouvez vous y référer pour une description plus complète.

Déroulement d'un modèle

En plus des symboles contenus dans le modèle, le modèle est également constitué d'une série de fonctions générales qui en contrôlent le déroulement. Le déroulement présenté ici est celui d'un modèle de cas. Toutefois, la majorité de ce qui est présenté ici s'applique également aux modèles basés sur le temps. Notons que cette section ignore la question du traitement en parallèle qui est intégré à Modgen.

Lecture des paramètres

La première étape d'une simulation d'un modèle de cas ou basé sur le temps est la lecture des paramètres. Il n'est pas nécessaire de créer une fonction globale dans le code du modèle pour gérer la lecture ou la modification interactive des paramètres. Cela est fournit par Modgen.

Validation des paramètres

Modgen comprend des fonctions permettant au concepteur de valider et modifier les valeurs des paramètres fournies par l'utilisateur. Pour de plus amples renseignements sur ces fonctions, vous pouvez vous référer au Guide du concepteur de Modgen 12.

Pré-simulation

Après la lecture des paramètres vient la phase de pré-simulation. La phase de pré-simulation ne fera rien du tout à moins que le concepteur du modèle ait défini une fonction « PreSimulation » ou plus. Notons qu'il est possible de créer plus d'une fonction « PreSimulation » afin d'avoir plus de modularité. Dans une version future de Modgen, il sera même possible de préciser l'ordre d'exécution des fonctions PreSimulation d'un modèle.

Habituellement, les fonctions « PreSimulation » servent à calculer les paramètres générés par le modèle. À ce stade, il n'y a toujours qu'un seul fil d'exécution alors on peut modifier les paramètres générés par le modèle sans craindre les collisions entre différents fils d'exécution. De plus, il est impossible pour une fonction « PreSimulation » de modifier les acteurs, puisque les acteurs n'ont pas encore été créés.

Exemple :

```
parameters
{

	//FR Hasards de mortalité totaux ( m total)
	double			MortHazard[MODELED_AGE][SEX];

	//FR Hasards de mortalité due à la maladie X de la population(mx)
	double			MortHazardX[MODELED_AGE][SEX];

	//FR Hasards de mortalité due à la maladie C de la population(mc)
	double			MortHazardC[MODELED_AGE][SEX];

	//FR Hasards de mortalité résiduels
	model_generated double
		ResidualMortHazard[MODELED_AGE][SEX];
};

void PreSimulation()
{	int		nAge = {0};
	int		nSex = {0};
	int		nTime = {0};

	for (nAge = 0; nAge < SIZE(MODELED_AGE); nAge++)
	{
		for (nSex = 0; nSex < SIZE(SEX); nSex++)
		{
			// Calcul des hazards de mortalité résiduels
			ResidualMortHazard[nAge][nSex] =
				MortHazard[nAge][nSex]
				- MortHazardC[nAge][nSex]
				- MortHazardX[nAge][nSex];
		}
	}
}
```

Simulation

Une fois la pré-simulation terminée commence la phase de simulation. Le modèle doit absolument contenir une fonction « Simulation ». Typiquement, dans le cas d'un modèle de cas, il s'agira simplement d'une boucle appelant la fonction « CaseSimulation » pour chaque cas du modèle.

Exemple :

```
void Simulation()
{
	long  lCase = 0;

	for ( lCase = 0; lCase < CASES() && !gbInterrupted &&
		!gbCancelled && !gbErrors; lCase++ )
	{
	    StartCase();
```

```
            CaseSimulation();
            SignalCase();
        }
}
```

La fonction « Simulation » est plus compliquée pour un modèle basé sur le temps, parce qu'elle doit également créer la population de début. Elle doit également contenir la boucle d'événements contenue dans la fonction « CaseSimulation » présentée ci-bas pour les modèles de cas.

Notons que cette fonction est incluse dans les modèles créés par un des assistants de création d'un modèle de Modgen.

CaseSimulation

Dans un modèle de cas, Modgen n'exige pas une fonction appelée « CaseSimulation ». Par contre, l'utilisation d'une telle fonction simplifie le code. Cette fonction est responsable de la simulation d'un cas, et commence par créer l'acteur de départ du cas. C'est à l'intérieur de cette fonction que doit se trouver la boucle d'événements contrôlant la simulation du cas.

Exemple d'une boucle d'événements :

```
// boucle d'événements pour le cas actuel
while ( !gpoEventQueue->Empty() )
{

        if ( gbCancelled || gbErrors)
        {
                // en cas d'erreurs, ferme le cas, détruis tous les acteurs
                gpoEventQueue->FinishAllActors();
        }
        else
        {
                // avance le temps des acteurs au temps de l'événement
                //    suivant
                gpoEventQueue->WaitUntil( gpoEventQueue->NextEvent() );

                // exécute l'événement suivant
                gpoEventQueue->Implement();
        }
}
```

Cette boucle est au cœur de la simulation d'un modèle et ne devrait être modifié qu'avec grand soin. Notons d'ailleurs qu'elle est incluse dans les modèles créés par un des assistants de création d'un modèle de Modgen.

Dans les faits, cette boucle contrôle l'engin de simulation de Modgen. À l'intérieur de l'engin de simulation, il y a une queue d'événements qui maintient le temps d'attente de chaque événement du modèle. Cette queue est ordonnée de sorte que les événements sont exécutés en un ordre précis. L'engin de simulation à l'intérieur de Modgen exécute les événements dans l'ordre suivant :

- l'événement ayant le temps le plus rapproché est exécuté en premier
- Si les temps sont identiques, alors l'événement ayant la plus haute priorité est exécuté en premier

- Si les priorités sont également identiques, alors les événements sont exécutés en ordre alphabétique

- Si les noms sont aussi identiques, alors l'événement de l'acteur créé en premier dans la simulation est exécuté en premier.

Notons que si les noms sont identiques, alors il s'agit forcément d'événements d'acteurs différents puisque les événements ne se trouvent qu'une seule fois par acteur dans la queue d'événements.

PostSimulation

Au besoin, le concepteur de modèles peut définir une fonction « PostSimulation » Si elle est définie, cette fonction est exécutée après la phase de simulation, mais avant la tabulation des données. Il est rare que cette fonction soit utile. Toutefois, si des variables globales sont utilisées dans le modèle, il peut être utile de les réinitialiser dans cette fonction. De cette façon, si l'utilisateur redémarre une simulation sans avoir fermé le modèle, les variables globales sont initialisées aux mêmes valeurs que lors de la première simulation.

Tabulation

La dernière phase produit les tableaux. Cette expression est toutefois trompeuse puisque la tabulation dans Modgen est faite au fur et à mesure. Il s'agit simplement ici d'enregistrer les tableaux produits pendant la simulation dans la base de données de sortie.

Toutefois, les tableaux d'utilisateurs sont calculés à cette étape avant d'être enregistrés dans la base de données. C'est à cette étape que sont exécutées les fonctions « UserTables » créées par le concepteur du modèle pour calculer les valeurs des cellules des tableaux d'utilisateur.