Modelica

Un langage pour modéliser et simuler des systèmes dynamiques hybrides

(suite)

http://www.lirmm.fr/~reitz/Modelica

Philippe.Reitz@LIRMM.fr

Équipe MAREL

novembre 2017

Sommaire

4a. Modelica
Analyse du langage

- Présenté comme un langage orienté objet
  - Notion de classe :
    - Définition d'éléments
      - variables (terminologie LOO : attributs, champs)
      - classes imbriquées, de toutes sortes :
        - » Types, classes, enregistrements
        - » Modèles, connecteurs
        - » Fonctions, paquets
    - Instanciation
    - Héritage multiple
      - Modification, redéfinition d'éléments hérités
      - Ajout de nouveaux éléments
    - La notion de redéfinition autorise la généricité
  - Toutefois :
    - Pas de notion de méthode *à la LOO*
      - méthode = fonction ayant un paramètre implicite (receveur) jouant un rôle particulier (liaison dynamique de code)
    - Pas de création ou suppression d'objets en cours d'exécution

4a. Modelica – exemple héritage simple et composition

```
model A
  parameter Real r;
end A;

model B
  extends A(r=-1); // modification de la valeur
  parameter Integer i;
end B;

model C
  replaceable A m(final r=-2);
    // m une variable liée à un objet de classe A modifiable, r non modifiable
  replaceable class T = A; // T est une variable liée à une classe modifiable
  T n;                     // n est une variable liée à un objet de classe T
end C;

model D
  extends C(redeclare replaceable B m(i=-3)); // redéclaration de la classe de m
end D;

A oA(r=1);
B oB1(r=2, i=3), oB2(i=4);
C oC(m=oA,n=oA);
D oD(m(i=-4));
```

4a. Modelica – exemple héritage multiple

```
model A
  parameter Real r=0;
end A;

model B
  extends A(r=-1);
  parameter Integer i;
end B;

model C
  extends A(r=-2);
end C;

model D
  extends B;
  extends C(r=-3); // sinon r=-1
end D;

A oA(r=1);
B oB(r=2, i=3);
C oC;
D oD(i=3);
```

```
parameter Real oA.r = 1.0;
parameter Real oB.r = 2.0;
parameter Integer oB.i = 3;
parameter Real oC.r = -2.0;
parameter Real oD.r = -3.0;
parameter Integer oD.i = 3;
```

4a. Modelica – exemple généricité

- Paquet pile générique

```
package GenericStack

  replaceable class Element
  end Element;

  record Stack
    parameter Integer  maxsize = 10;
    Integer            size = 0;
    Element[maxsize]   vec;
  end Stack;

  function Push
    input Stack si;
    input Element e;
    output Stack so;
  algorithm
    so := si;
    if so.size<so.maxsize then
      so.size := so.size+1;
      so.vec[so.size] := e;
    end if;
  end Push;
```

- Exemple d'instanciation

```
  package IntegerStack = GenericStack(
      redeclare type Element = Integer);
  package StringStack = GenericStack(
      redeclare type Element = String);

  import IS = testGeneric2.IntegerStack;
  IS.Stack stk(maxsize=20);
  Integer item;
algorithm
  stk := IS.Push(stk, 35);
  stk := IS.Push(stk, 400);
  stk := IS.Push(stk, 44);
  item := IS.Top(stk);
  stk := IS.Pop(stk);
```

4a. Modelica
Analyse du langage

- Présenté comme un langage orienté composant
  - Notion de *modèle* = composant :
    - Chaque composant a une dynamique propre
      - Équations et algorithme
  - Notion de connexion = *connecteur*
    - Les objets du genre *connecteurs* sont les seuls connectables
      - Équations de connexion
    - Un composant connectable est un modèle ayant des champs connecteurs
    - Notion d'élément *implicitement* connectable (**`inner`**/**`outer`**)
      - Si absente, nécessiterait de nombreuses équations de connexion
  - Toutefois :
    - Pas de création ou de suppression de composants
      - Mais connexions modifiables lors d'une simulation
        - » L'ensemble des configurations de connexions est connu avant simulation

4a. Modelica – exemple inner/outer

```
class A
  outer Real p; // p est un puits
  Real  a = -1;
end A;

class B
  Real u = 1;
  A    v(a=2);
end B;

class C
  inner Real p; // p est une source
  A     x;
  B     y(v(a=3));
end C;

class D
  C o1(p=4),    // o1.p = 4 = o1.x.p = o1.y.v.p
    o2(p=5);    // o2.p = 5 = o2.x.p = o2.y.v.p
end D;
```

4b. Exercices

- Exercice 1 : installation et test
  - Installer OpenModelica
  - Tester la balle rebondissante
  - La balle est une sphère
    - Modifier le code afin de tenir compte de son rayon

4b. Exercices

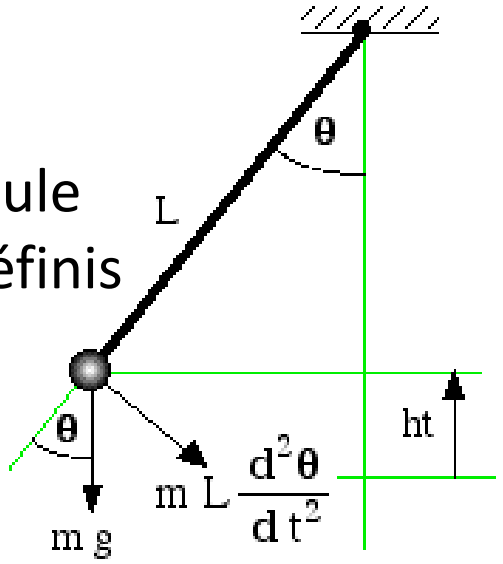
- Exercice 2 : le circuit RC
  - Tester la version 1 du circuit RC
    - Visualiser la tension aux bornes de la source S
    - Visualiser la tension U_C aux bornes du condensateur C
  - Tester différentes valeurs de la fréquence de la source : F = 1Hz, 10Hz, 100Hz, 1KHz ; que remarque-t-on pour la tension U_C ?

4b. Exercices

- Exercice 3 : le pendule simple
  - Ecrire le modèle permettant d'étudier un pendule simple, en exploitant les types physiques prédéfinis de Modelica (`Modelica.SIunits.*`)

    ```
    import SI = Modelica.SIunits;
    ```

    permet d'écrire ensuite le type : `SI.Length`



  - Dans un premier temps, supposer qu'il n'y a pas d'amortissement ; l'équation à résoudre est alors :

$$mL^2\ddot{\theta} + mLg\sin\theta = 0$$

  - Prendre ensuite en compte un facteur d'amortissement β, l'équation devenant :

$$mL^2\ddot{\theta} + mLg\sin\theta + \beta L^2\dot{\theta} = 0$$

Données : $L = 0{,}5\text{m}, m = 0{,}2\text{Kg}, \beta = 1, \theta_0 = 75°$

4b. Exercices

- Exercice 4 : programmation orientée objet

À résoudre d'abord avec votre LOO favori

- écrire une classe *uneFigure* permettant de représenter des figures du plan.
  - Toute figure est caractérisée par au moins un point de référence, et peut être déplacée (indication d'un déplacement pour chaque coordonnée).
  - étant donnée une figure, il est possible d'en connaitre la taille ou de lui demander de se décrire (messages sur la console).
- écrire alors une classe *unFigurePlane*, spécialisation de *Figure*, pour qui la taille correspond à sa surface.
- écrire une classe *unCercle* permettant de représenter un cercle ; un cercle est une figure plane qui se caractérise par son rayon, et dont le point de référence est son centre.
- faire de même avec les rectangles (hauteur et largeur parallèles aux axes, le point de référence étant le coin inférieur gauche), les triangles (hauteur et base) et les carrés (coté).

4b. Exercices

- Exercice 4 : exemple d'affichage

– Si *f* est un objet `uneFigure`, affiche sur le terminal

```
Je suis une figure
Mon point de référence est en (x, y)
Ma taille vaut z
```

– Si *f* est un objet `uneFigurePlane`, affiche sur le terminal

```
Je suis une figure plane
Mon point de référence est en (x, y)
Ma surface vaut z
```

– Si *f* est un objet `unCarré`, affiche sur le terminal

```
Je suis un carré
Mon centre est en (x, y)
Ma surface vaut z
Mon coté vaut c
```

4b. Exercices

- Exercice 4 (suite)
  - définir la classe qui permet de représenter un segment de droite (sa taille est sa longueur, et le point de référence est l'une de ses extrémités), ainsi qu'une classe permettant de représenter un point (sa taille est toujours nulle).
  - définir la classe des rectangles évidés par des cercles (nous supposerons que le cercle intérieur est toujours inclus dans le rectangle qu'il évide).
  - généraliser la classe afin que toute figure plane puisse être évidée par une autre figure plane.
  - écrire une classe permettant de représenter des figures pesantes, lesquelles sont caractérisées par une densité.
    - connaissant la densité et la taille d'une figure pesante, il est possible d'en déduire son poids.
  - définir alors la classe des cercles pesants.

4c. Au delà de Modelica

- Intégrer la dimension spatiale
  - Présentation SCP