Introduction à OpenModelica

V. Hilaire

UTBM/IRTES-SET/IMSI

Bibliographie

- Introduction to Modeling and Simulation of Technical and Physical Systems with Modelica. Peter Fritzson. Wiley.
- Tutoriaux disponibles sur le site www.openmodelica.org

Plan

Introduction
Elements de base
La structuration en classes
Héritage
Connecteur
Les événements et le discret
Conclusion générale

1 Introduction

2 Elements de base

3 La structuration en classes

4 Héritage

5 Connecteur

6 Les événements et le discret

7 Conclusion générale

Pourquoi Modelica (en bref) ?

- Modelica est un langage déclaratif, orienté objet, acausal, basé sur des équations
- Modelica est spécialement adapté à la modélisation et la simulation de systèmes
- Modelica permet de décrire des systèmes hybrides (on peut utiliser du discret et du continu)

Quelques domaines d'application de Modelica

- Mécatronique
- Electronique
- Systèmes de puissance
- Hydraulique
- Aérodynamique
- Ingénierie manufacturière
- et surtout Modelica permet de combiner facilement des modèles hétérogènes

Modelica c'est quoi ?

- Modelica n'est pas un outil
- C'est une spécification gratuite et disponible d'un langage
- Il existe plusieurs outils disponible qui implémente la spécification :
  - OpenModelica de l' Open Source Modelica Consortium
  - MathModelica by MathCore
  - Dymola par Dassault systems / Dynasim
  - MapleSim par MapleSoft

Modelica, langage de nouvelle génération de modélisation

- Langage déclaratif qui permet de décrire des spécifications de haut niveau avec un excellent niveau de correction
  - Il est déclaratif car on ne programme pas le comportement du système c'est Modelica qui s'en charge à partir de a déclaration sous forme d'équations et/ou fonctions mathématiques
  - Le niveau de correction est bon car on reste à un haut niveau d'abstraction et on évite de tomber dans le piège de sur/sous-spécifier un système
- Tous les concepts du langage dérivent du concept de classe
  - Les éléments du langage sont fortement typés avec une syntaxe proche de Java & Matlab
  - Autorise la décomposition hiérarchique d'un système en sous-systèmes
  - Efficace en terme d'exécution (comparable à du C)

V. Hilaire

Introduction

Elements de base

La structuration en classes

Héritage

Connecteur

Les événements et le discret

Conclusion générale

Orientation objet

- La nature statique et déclarative des modèles mathématiques est mise en avant
- Le concept d'objet est utilisé comme principe de structuration
  - Un modèle est un objet, qui peut être composé d'objets
  - L'espace d'état est décrit par des objets (Real, Integer, models, functions, packages, parameterized classes, ...)
  - Les opérations sur les matrices (type Matlab), l'arithmétique scalaire, les fonctions, les itérateurs sont disponibles
- Les propriétés dynamiques des modèles sont exprimées sous forme d'équations

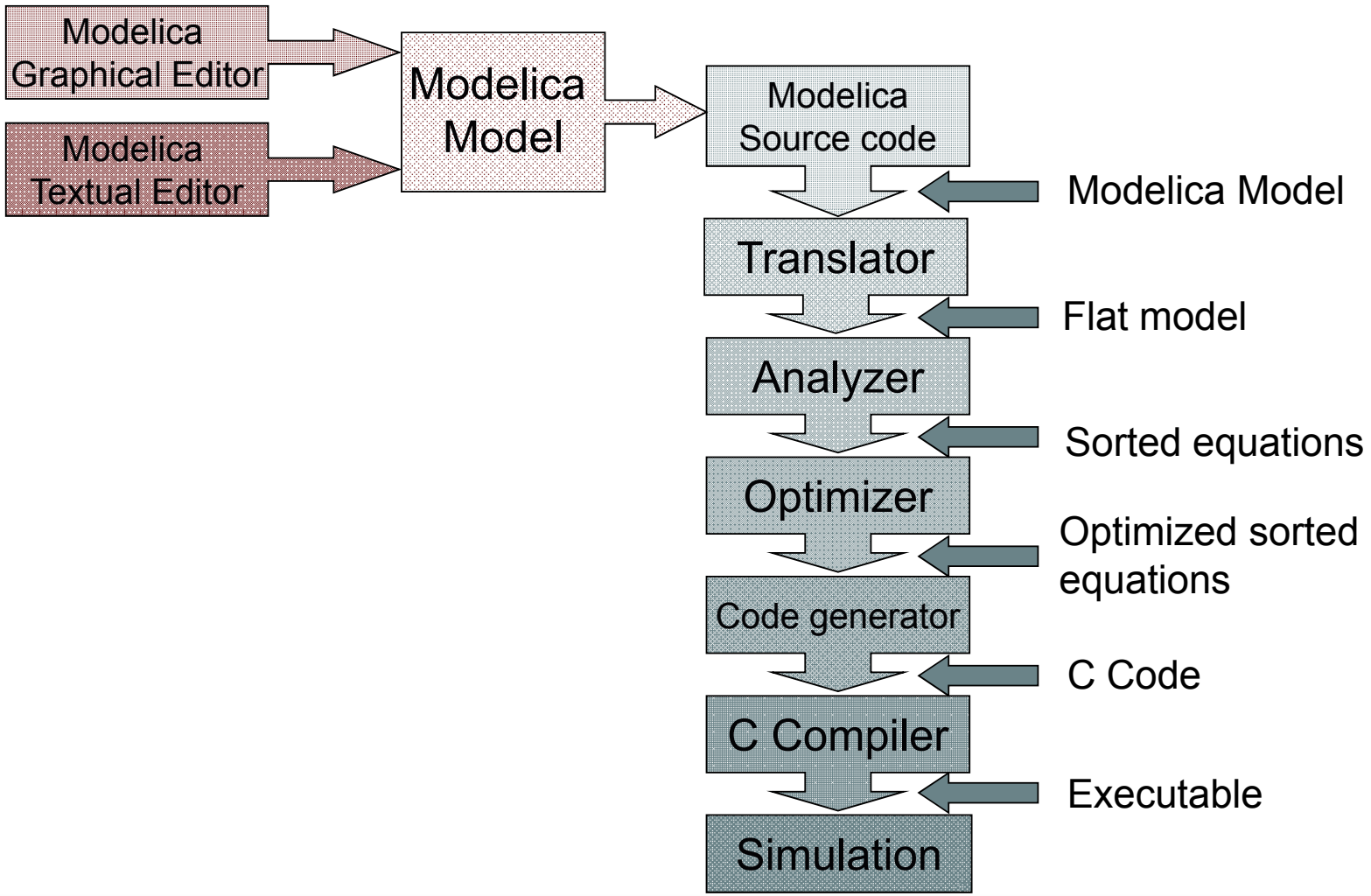
Modélisation acausale

- Le principe est que l'ordre d'exécution n'est pas décidé lors de la modélisation
- Sur l'exemple d'une résistance électrique :
  - On peut la caractériser par l'équation $U = R * i$
  - Les possibilités causales sont : $i = U/R$ ou $U = R * i$ ou $R = U/i$

Principe

- Langages textuel ou graphique orientés objets
- Le comportement peut être décrit par des systèmes d'équations algébrique et/ou différentielles (simulation continue)
- Le comportement peut être décrit par des changements d'états déclenchés par des événements (simulation discrète)

La compilation



Plan

1 Introduction

2 Elements de base

3 La structuration en classes

4 Héritage

5 Connecteur

6 Les événements et le discret

7 Conclusion générale

Exemple : une résistance

Classe Resistor

```
class Resistor
  parameter Real R=100;
  Real u,i;
equation
  R * i = u;
end Resistor;
```

Déclaration de modèle

- Le mot clé **class** définit un modèle (sous forme de classe) que l'on doit nommer pour le distinguer et le typer
- Le mot clé end termine la déclaration

Les variables d'état d'un modèle

- A la suite du mot clé **class** et du nom de la classe Resistor, on a les variables d'états du modèle
- Chaque variable d'état (ici une seule) est typé et nommé
- Le type de la variable d'état **Real** indique qu'on manipule un nombre réel
- le mot clé parameter indique que c'est un paramètre du modèle (et sa valeur)

Types primitifs

- Boolean : true ou false
- Real : réel (virgule flotante) par exemple 2.4e-6
- String : chaîne de caractères “Ceci est une chaîne”
- Enumeration : énumération de littéraux ShirtSize.Medium

Le comportement d'un modèle

- La partie qui suit le mot clé **equation** décrit le comportement du modèle
- Ce comportement est décrit par des équations (ici une seule)

Les paramètres

- Dans l'exemple on voit apparaitre le mot clé **parameter**
- Ce mot clé définit un paramètre de classe
- Une classe (un modèle) peut ainsi définir une variable dont la valeur peut être changée en début de simulation
- Une classe peut définir autant de paramètres que nécessaires

Les constantes

- Les paramètres sont constants pendant la simulation
- Pour définir une constante qui ne peut jamais être changée on utilise le mot clé **constant** qui précède le type et le nom de l'attribut : constant Real PI=3.141592653589793; ou constant String redcolor = "red";

Plan

Introduction
Elements de base
La structuration en classes
Concepts de base
Différents types de classe
Héritage
Connecteur
Les événements et le discret
Conclusion générale

1 Introduction

2 Elements de base

3 La structuration en classes

4 Héritage

5 Connecteur

6 Les événements et le discret

7 Conclusion générale

Plan

1 Introduction

2 Elements de base

3 La structuration en classes
- Concepts de base
- Différents types de classe

4 Héritage

5 Connecteur

6 Les événements et le discret

Le concept de classe

- La classe permet la définition d'un type
- Les éléments de ce type, on parle d'instances, possèdent les même propriétés :
  - La déclaration de l'espace d'état d'un modèle
  - Son comportement

Classe Capacitor

```
class Capacitor
  Real u,i;
  parameter Real C;
equation
  i = C * der(u);
end Capacitor;
```

Classe Source

```
class Source
  Real u;
  parameter Real A(start = 10),w(start = 20);
  parameter Real VA = 220;
  parameter Real f = 50;
  constant Real PI = 3.14;
equation
  v = VA * sin(2 * PI * f * time);
end Source;
```

Création d'instances

Classe Circuit

```
model Circuit
  Source AC;
  Resistor R1(R = 100);
  Capacitor C;
end Circuit;
```

Initialisation de membres

- L'initialisation de membres d'une classe se fait de la même façon que pour les types prédéfinis avec le mot clé **start**
- L'ordre et le type lors de l'énumération des valeurs initiales doit respecter la déclaration de la classe

Initialisation par équation

Classe Controller

```
class Controller
Real y;
equation
der(y)=a*y+b*u;
initial equation
der(y)=0;
end Controller;
```

Initialisation par équation

- La solution lors de l'initialisation pour la classe de l'exemple précédent est donné par l'application de l'équation initiale à l'équation décrivant le comportement
- $\text{der(y)} = 0 \implies y = -\frac{b}{a} * u$

Plan

Introduction

Elements de base

La structuration en classes

Concepts de base

Différents types de classe

Héritage

Connecteur

Les événements et le discret

Conclusion générale

1 Introduction

2 Elements de base

3 La structuration en classes
- Concepts de base
- Différents types de classe

4 Héritage

5 Connecteur

6 Les événements et le discret

Classe et plus si affinités

- Avec Modelica, quasiment tout est une classe
- Pour faciliter la lecture et la maintenance certains usages particuliers ont été distingués
- Chaque usage est différentié par un mot clé
- Sont définies ainsi des spécialisations de classe (record, function, ...)

Record

La classe Record est utilisé pour définir une structure de donnée particulière qui ne peut pas contenir d'équations (pas de comportement)

Record Person

```
record Person
Real age;
String name,surname;
end Person;
```

Réutilisation de classe

- Le concept de classe permet la réutilisation de modèles
- Il permet également d'adapter les classes membres en apportant des modifications
- Cette démarche permet de définir de manière incrémentale les éléments d'une simulation

Réutilisation de classe

classe LowPassFilter

```
class LowPassFilter
parameter Real T=1;
Real u,y(start=1);
equation
T*der(y)+y=u;
end LowPassFilter;
```

Réutilisation de classe

classe FiltersInSeries

```
class FiltersInSeries
LowPassFilter F1(T=2),F2(T=3);
equation
F1.u=sin(time);
F2.u=F1.y;
end FiltersInSeries
```

Plan

Introduction
Elements de base
La structuration en classes
Héritage
Héritage simple
Héritage multiple
Modification
Connecteur
Les événements et le discret
Conclusion générale

1 Introduction

2 Elements de base

3 La structuration en classes

4 Héritage

5 Connecteur

6 Les événements et le discret

7 Conclusion générale

Plan

1 Introduction

2 Elements de base

3 La structuration en classes

4 Héritage
- Héritage simple
- Héritage multiple
- Modification

5 Connecteur

6 Les événements et le discret

Héritage simple

- Le mécanisme d'héritage permet de dire qu'une classe hérite de l'espace d'états et du comportement d'une autre classe
- Cela se traduit par une recopie des définitions de l'état des équations
- Pour spécifier un héritage on utilise le mot clé **extends** suivi du nom de la classe dont on hérite
- L'héritage est un mécanisme puissant pour réutiliser et spécialiser des modèles

Héritage simple

record ColorData

```
record ColorData
parameter Real red = 0.2;
parameter Real blue = 0.6;
Real green;
end ColorData;
```

classe Color

```
class Color extends ColorData;
equation
red + blue + green = 1;
end Color;
```

Plan

1 Introduction

2 Elements de base

3 La structuration en classes

4 Héritage
- Héritage simple
- Héritage multiple
- Modification

5 Connecteur

6 Les événements et le discret

Héritage multiple

- Il est possible d'hériter de plusieurs classes différentes
- Pour cela on utilise plusieurs fois le mot clé **extends** suivi des noms de classe
- Les définitions identiques vont être fusionées
- L'héritage multiple de définitions différentes d'un même item est une erreur

Héritage multiple

classe ColoredPoint

```
class ColoredPoint extends Point; extends Color;
end ColoredPoint;
```

Plan

1 Introduction

2 Elements de base

3 La structuration en classes

4 Héritage
- Héritage simple
- Héritage multiple
- Modification

5 Connecteur

6 Les événements et le discret

Héritage et modification

V. Hilaire

Introduction

Elements de base

La structuration en classes

Héritage

Héritage simple

Héritage multiple

Modification

Connecteur

Les événements et le discret

Conclusion générale

- La modification est une façon concise de combiner héritage et déclaration de classes
- Un modifier modifie la déclaration héritée
- Par exemple, dans ce qui suit l'altitude est précisée avec une valeur de départ différente

Héritage et modification

classe Body

```
model Body
Real mass;
Real altitude;
String name;
end Body;
```

Héritage et modification

classe Rocket

```
model Rocket extends Body;
parameter Real massLossRate=0.000277;
Real altitude(start= 59404);
Real velocity(start= -2003);
Real acceleration;
Real thrust;
Real gravity;
equation
thrust-mass*gravity= mass*acceleration;
der(mass)= -massLossRate*abs(thrust);
der(altitude)= velocity;
der(velocity)= acceleration;
end Rocket;
```

Plan

Introduction

Elements de base

La structuration en classes

Héritage

Connecteur

Modélisation objet de connection

Equation de connection

Les événements et le discret

Conclusion générale

1 Introduction

2 Elements de base

3 La structuration en classes

4 Héritage

5 Connecteur

6 Les événements et le discret

7 Conclusion générale

Le principe

- Le principe de modélisation orientée objet (classe) résulte en un ensemble d'éléments de modèle
- Chaque élément possède ses propres caractéristiques
- Chaque élément possède sa propre dynamique
- Un élément de modèle (classe) peut également être composé d'autres éléments
- Une autre façon de concevoir des systèmes complexes est de connecter des éléments de modèle

Plan

Introduction
Elements de base
La structuration en classes
Héritage
Connecteur
Modélisation objet de connection
Equation de connection
Les événements et le discret
Conclusion générale

1 Introduction

2 Elements de base

3 La structuration en classes

4 Héritage

5 Connecteur
- Modélisation objet de connection
- Equation de connection

6 Les événements et le discret

Le principe

- La connection se fait par une classe particulière : **connector**
- Ce type particulier de classe introduit deux types de variables :
  - Les variables de type effort/potentiel/niveau (non flux)
  - Les variables de type flux

Pin

```
connector Pin
Voltage v;
flow Current i;
end Pin;
```

- La variable v est de type potentiel
- La variable i est de type flux, la somme des courants connecté vaut toujours 0

Plan

Introduction
Elements de base
La structuration en classes
Héritage
Connecteur
Modélisation objet de connection
Equation de connection
Les événements et le discret
Conclusion générale

1 Introduction

2 Elements de base

3 La structuration en classes

4 Héritage

5 Connecteur
- Modélisation objet de connection
- Equation de connection

6 Les événements et le discret

- Les connections entre connecteurs se font par des équations avec modelica
- Pour des variables de type effort/potentiel/niveau le principe se ramène à une égalité (loi des noeuds de Kirchoff)
- Pour des variables de type flux le principe se ramène à une somme nulle (loi des mailles de Kirchoff)
- La syntaxe est : **connect(connecteur1,connecteur2)**
- Cette equation de type particulier doit être déclarée dans la section **equation** d'une classe
- connecteur1 et connecteur2 doivent être des connecteurs déclarés dans la classe ou être membre d'une variable de la classe

Classe TwoPins

```
partial class TwoPins
  Real v;
  Real i;
  Pin p;
  Pin n;
equation
  v = p.v - n.v;
  0 = p.i + n.i;
  i = p.i;
end TwoPins;
```

Classe Resistor

```
class Resistor extends TwoPins
parameter Real R=100;
equation
v=R*i
end Resistor;
```

Classe Source

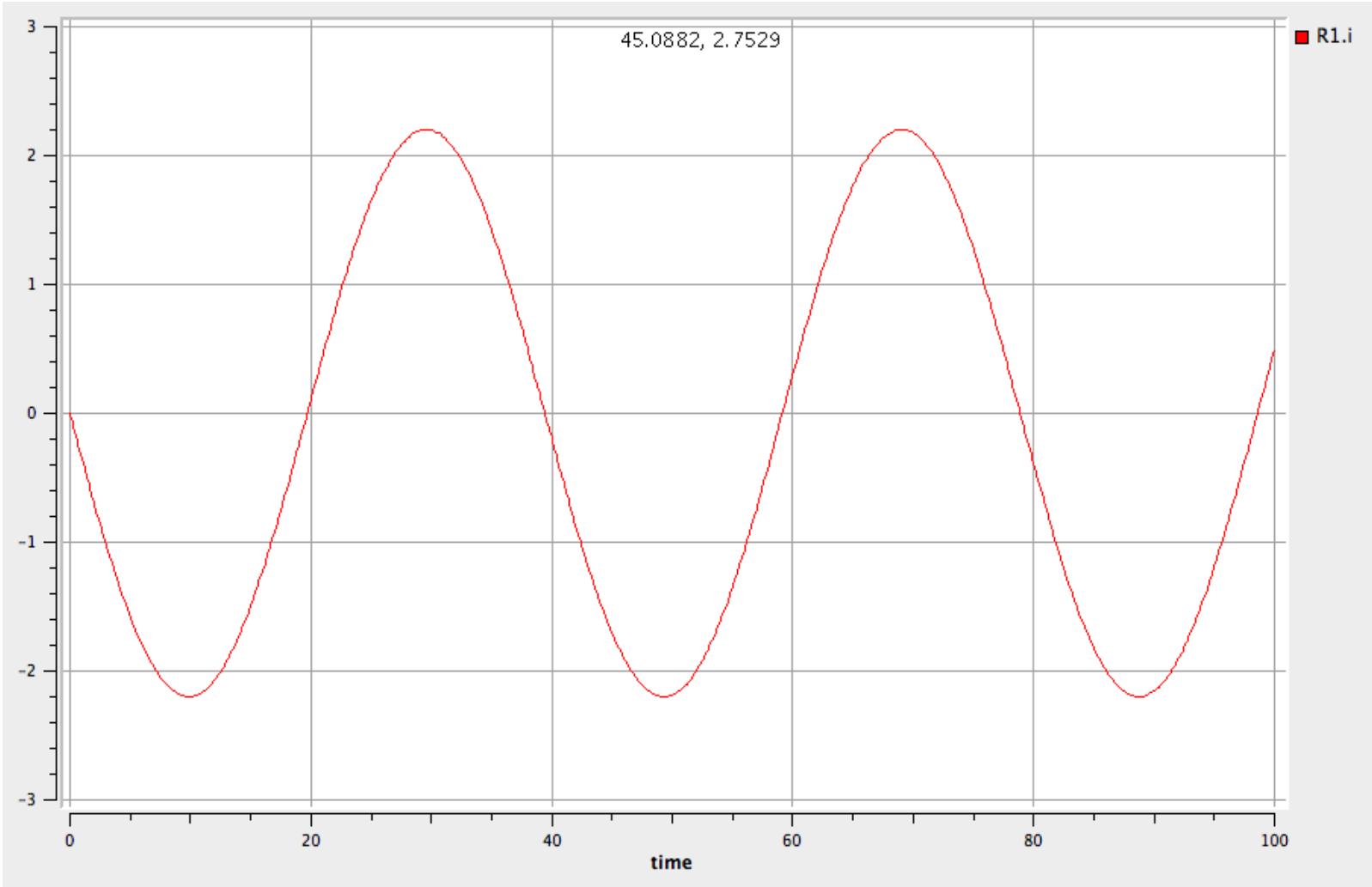
```
class Source
  extends TwoPins;
  parameter Real A(start = 10),w(start = 20);
  parameter Real VA = 220;
  parameter Real f = 50;
  constant Real PI = 3.14;
equation
  v = VA * sin(2 * PI * f * time);
end Source;
```

Classe Ground

```
class Ground
  Pin p;
equation
  p.v = 0;
end Ground;
```

Classe Circuit

```
class Circuit
  Source AC;
  Resistor R1(R = 100);
  Ground G;
equation
  connect(AC.p,R1.p);
  connect(R1.n,AC.n);
  connect(AC.n,G.p);
end Circuit;
```


45.0882, 2.7529
R1.i
3
2
1
0
-1
-2
-3
0
20
40
60
80
100
time

Plan

1 Introduction

2 Elements de base

3 La structuration en classes

4 Héritage

5 Connecteur

6 Les événements et le discret

7 Conclusion générale

La notion d'événement

- Les événements sont ordonnés dans le temps et forment une séquence d'exécution (un scénario) du système
- Un événement est un point sur l'axe du temps, il est instantané (sa durée est nulle)
- Au sein de modelica un événement peut être associé à :
  - Des conditions qui déclenchent son occurence
  - Des variables associées à cet événement
  - Un comportement associé à l'événement (équations activées/désactivées lors de l'occurence)

if-équation

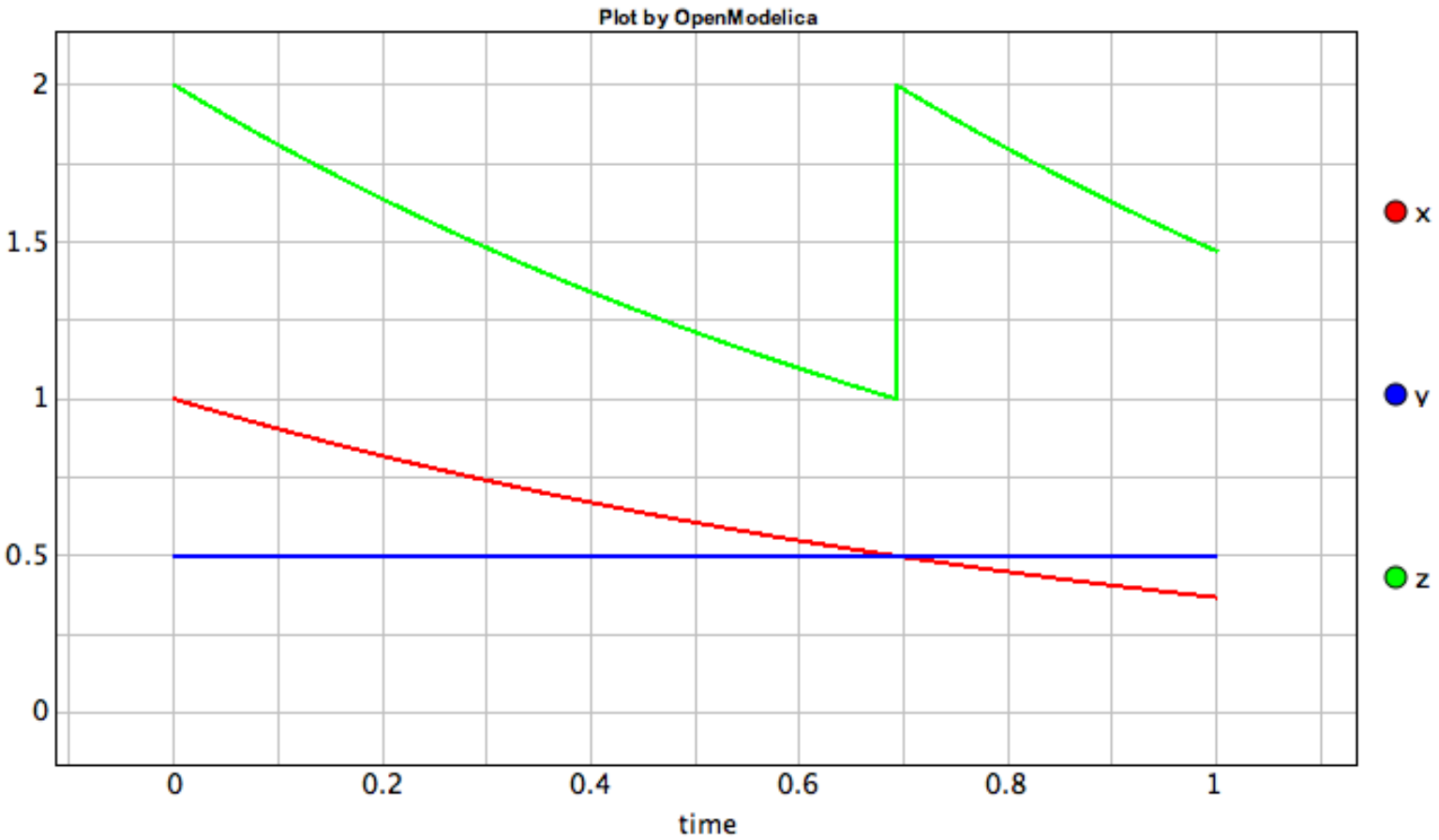
- La structure if-then-else peut s'appliquer à des équations
- La sémantique est que selon la condition (if) :
  - Si la condition est vraie, le simulateur applique la condition then
  - Si la condition est fausse, le simulateur applique la condition else
- Le temps et les variables sont continus
- La syntaxe est de la forme :

```
if [condition] then
[equations]
elseif [condition] then
[equations]
else
[equations]
end if;
```

Exemple de if-équation

class Continuity

```
class Continuity
Real x(start = 1);
parameter Real y(start=0.5);
Real z;
equation
  der(x) = -x;
if (y<x) then
z=2*x;
else
z=4*x;
end if;
end Continuity;
```


Plot by OpenModelica
2
1.5
1
0.5
0
0
0.2
0.4
0.6
0.8
1
time
x
v
z

Variables discrètes

- Dans une when-équation on peut manipuler des variables discrètes
- Les variables de type Integer, String, Boolean, enumeration sont discrètes par nature
- Les variables modifiées dans une when-équation sont automatiquement discrètes
- Les variables préfixées par le mot clé **discrete** sont discrètes
- Pour une variable discrète, dans une when-équation, on peut faire référence à la valeur de la variable avant l'occurence de l'événement
- Cette valeur est renvoyée par **pre(x)** où x est le nom de la variable

when-équation

- L'instruction when peut également s'appliquer à des équations
- La sémantique est que lors de l'occurence d'un événement (et pour cet instant seulement) une équation est activée/désactivée
- La syntaxe est de la forme :

```
when [event or condition] then
[equations]
elsewhen [event or condition] then
[equations]
end when;
```

Fonction reinit

- Cette fonction doit être utilisée dans le corps d'une when-équation
- Cette fonction prend deux paramètres d'entrée x et expr variables de type Real
- Lorsque l'événement déclenchant l'équation when intervient alors la variable x prend la valeur donnée par expr

Exemple de reinit

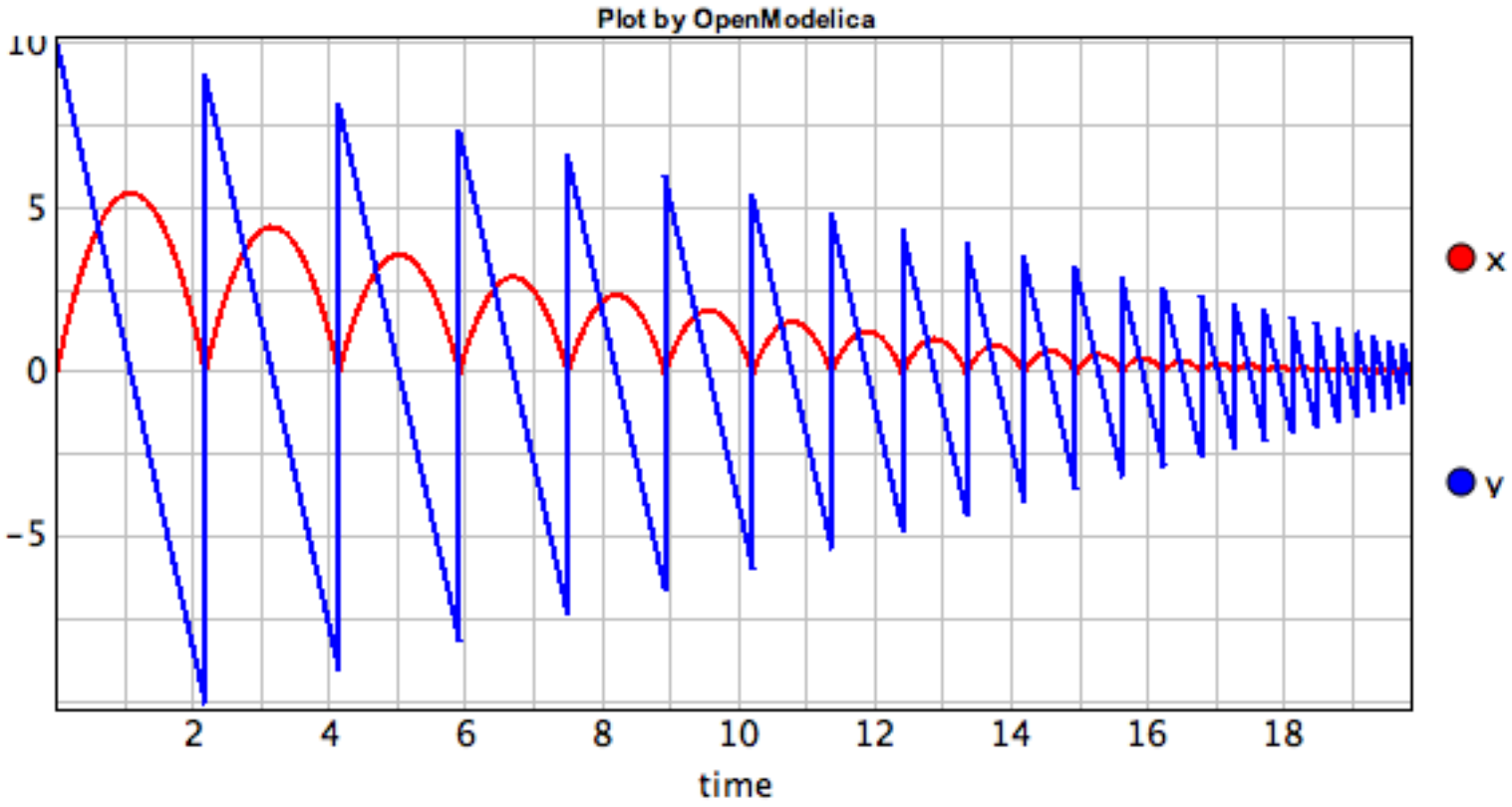
BouncingBall2

```
class BoucingBall2
  parameter Real g = 9.18;
  parameter Real c = 0.9;
  Real x(start = 0),y(start = 10);

equation
  der(x) = y;
  der(y) = -g;
  when x < 0 then
      reinit(y, -c * y);

  end when;
end BoucingBall2;
```

Simulation de BouncingBall2

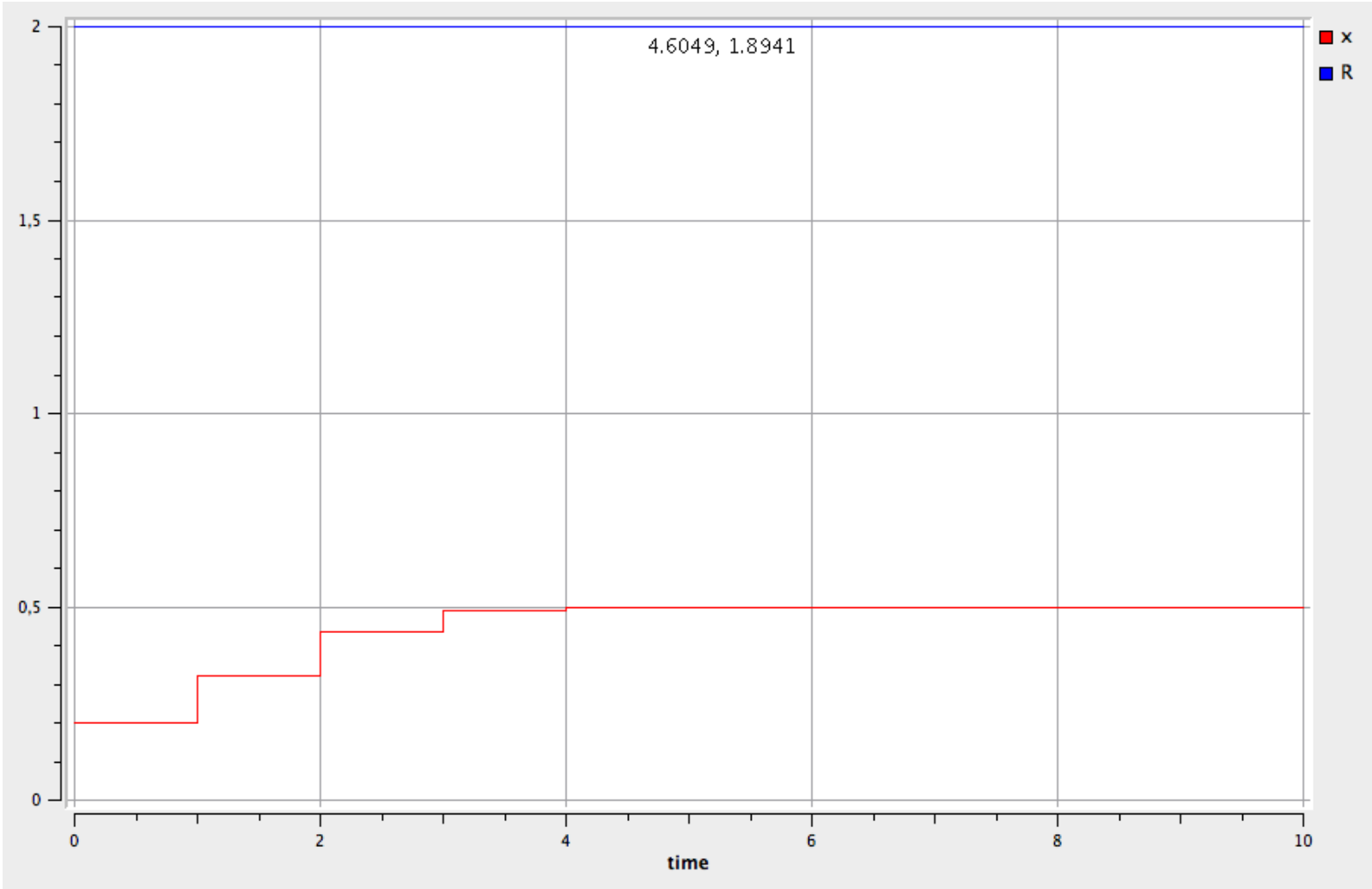


Autres constructions discrètes

LogisticFunction

```
model LogisticFunction
  discrete Real x(start = 0.2);
  parameter Real R(start = 2);
equation
  when sample(1, 1) then
      x = pre(x) * R * (1 - pre(x));
  end when;
end LogisticFunction;
```

Simulation de LogisticFunction



Plan

1 Introduction

2 Elements de base

3 La structuration en classes

4 Héritage

5 Connecteur

6 Les événements et le discret

7 Conclusion générale

Bilan

- OpenModelica est un langage de modélisation pour la simulation
- Le principes majeurs sont :
  - déclaratif
  - acausal
  - orienté objet

A venir

- De nombreux éléments d'OpenModelica n'ont pas été présentés
- Les fonctions
- Les connecteurs
- Les éléments de type déterministes (algorithmes par exemple)

QUESTIONS ?