Aplanado eficiente de grandes modelos Modelica

Mariano Botta

FCEIA, UNR

Agosto 2015

Contenido de la charla

Motivaciones

- Modelado, Simulación y Control en Tiempo Real con Aplicaciones en Electrónica de Potencia.
- Simulación en paralelo utilizando los métodos de cuantificación de estado.
- Modelos grandes.
- Aprovechar las ventajas de Modelica.

Contenido de la charla

Modelica

- Orientado a Objetos.
- Modelado de sistemas complejos, con componentes mecánicos, eléctricos, electrónicos, hidráulicos, térmicos, etc.
- Desarrollado por la asociación sin fines de lucro “Modelica Asociation”.
- Entornos de desarrollo: OpenModelica, MathModelica, Dymola, etc.
- El modelo está en texto plano

Clases

- Define un objeto.
- Son instanciadas mediante la definición de variables.
- Tienen tres secciones:
  1. Definiciones.
  2. Ecuaciones.
  3. Sentencias.

```
1  class X
2      // Definiciones de variables y
           clases
3  equation
4      // Ecuaciones
5  statements
6      // Sentencias
7  end X;
```

Prefijos de Clases

Prefijos de clase: *model*, *record*, *block*, *connector*, *function*, *package*

- Mejoran la lectura del código:
- Agregan restricciones a la clase

```
1  class Circuits
2      cclass Pin
3          Real v;
4          flow Real i;
5      end Pin;
6      class Componente
7          Pin n,p;
8      equation
9          n.v = p.v;
10     end Componente;
11 end Circuits;
```

```
1  package Circuits
2      connector Pin
3          Real v;
4          flow Real i;
5      end Pin;
6      model Componente
7          Pin n,p;
8      equation
9          n.v = p.v;
10     end Componente;
11 end Circuits;
```

Herencias de Clases

- Agrega significado semántico al modelo.
- Facilita la reutilización de código.
- Se utiliza la palabra reservada: *extends*.

```
1  model OnePort
2      Pin p;
3      Pin n;
4      Real v;
5      Real i;
6  equation
7      v = p.v - n.v;
8      i = p.i;
9      i = -n.i;
10 end OnePort;
11 model Capacitor
12     extends OnePort;
13     parameter Real C = 1;
14 equation
15     C * der(v) = i;
16 end Capacitor;
```

Declaraciones de tipo

- Tipos básicos: *Real*, *Integer*, *Boolean* y *String*
- Las clases definen un nuevo tipo.
- Sinónimos de tipos:
  type Nombre = [Prefijos] Tipo-Existente [Array] [Modificaciones]

Prefijos de Tipo: *flow*, *constant*, *parameter*, *discrete*, *input* y *output*.

```
1 package Circuits
2     type Current = flow Real;
3     type Voltage = Real;
4     connector Pin
5         Voltage v;
6         Current i;
7     end Pin;
8     type TenPin = Pin[10];
9 end Circuits;
```

Definiciones de variables

1. **Prefijos de tipos**: *flow*, *constant*, *parameter*, *discrete*, *input* y *output*.
2. **Tipo**: Nombre del tipo de la variable. Puede ser un tipo básico, una clase o un sinónimo de tipo. Ejemplo: *Real*, *String*, *Pin*, *TenPin*.
3. **Nombre de la variable**.
4. **Dimensión**: Modelica permite la definición de arreglos.
5. **Modificaciones**.

```
1  type TenPin = Pin[10];
2
3  TenPin pines;
4  Pin pines2 [10];
```

Modificaciones

Aparecen en:

1. Declaraciones de variables.
2. Sinónimo de tipo.
3. Definiciones de herencia.

Se permite:

1. Cambiar el valor inicial de una variable.
2. Anidar modificaciones.
3. Alterar la definición de una variable.
4. Cambiar la definición de un tipo.

```
1  package Circuits
2      model CircuitX
3          Capacitor cap;
4          Resistor res;
5          ...
6      equation
7
8      ...
9
10     end CircuitX;
11     model MainCircuit
12         Capacitor x(C = 2);
13         CircuitOne co1 (cap(C = 10));
14         CircuitOne co2 (cap(C = 15));
15     end MainCircuit;
16 end Circuits;
```

Ecuaciones de igualdad

Las ecuaciones no representan una asignación, sino igualdades. Pueden tener expresiones complejas de ambos lados de la igualdad y expresan una relación entre las variables.

Igualdades

```
1    p.v - n.v = v;
2    i = p.i;
3    i = -n.i;
```

Ecuación For

```
1    for i in 1:N loop
2        v[i] = p[i].v - n[i].
             v;
3        i[i] = p[i].i;
4        i[i] = -n[i].i;
5        C[i] * der(v[i]) = i;
6    end for;
```

Ecuaciones Connect

Los conectores:

- Son clases con ciertas restricciones.
- Se definen con el prefijo *connector*.
- No tienen ecuaciones.
- Tienen variables de dos tipos:
  1. Variables de potencial. Ejemplo: presión, voltaje, etc.
  2. Variables de flujo: definidas con el prefijo flow. Ejemplo: corriente, caudal, etc.
- Ejemplo: Clase *Pin*.

Ecuaciones Connect

Las ecuaciones *connect*:

- Conectan dos clases del mismo tipo.
- Genera relaciones entre las variables internas de los conectores:
  - Las variables de potencial dentro de una misma conexión deben ser iguales entre sí.
  - Las variables de flujo siguen las reglas de Kirchhoff: la suma de los flujos es igual a cero. Para mantener esta regla hay que considerar como flujo positivo aquel que tenga dirección hacia dentro del componente. En caso contrario, será considerado negativo.

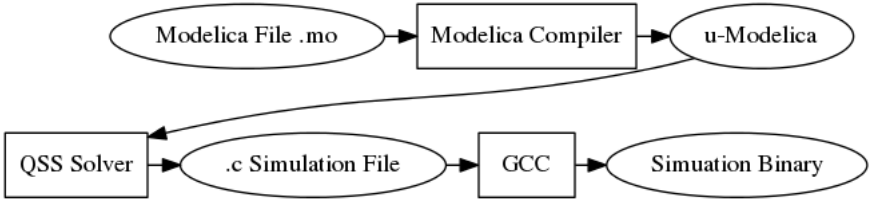
Ecuaciones Connect

```
1   package Circuits
2       ...
3       model ground
4           Pin p;
5       equation
6           p.v = 0;
7       end ground;
8       model inductor
9           extends OnePort;
10          parameter Real L = 1;
11      equation
12          L * der(i) = v;
13      end inductor;
14      model LC_circuit
15          Capacitor cap(v(start = 1));
16          inductor ind(L = 2);
17          ground gr;
18      equation
19          connect(ind.p,cap.p);
20          connect(ind.n,cap.n);
21          connect(cap.n,gr.p);
22      end LC_circuit;
23  end Circuits
```

```
1   // Variables de Potencial
2   ind.p.v = cap.p.v;
3   ind.n.v  = cap.n.v;
4   cap.n.v = gr.p.v;
5
6   // Variables de flujo
7   ind.p.i + cap.p.i = 0;
8   ind.n.i + cap.n.i + gr.p.i = 0;
```

Contenido de la charla

Simulación de Modelos Modelica



Modelica Compiler

Las ecuaciones de un modelo deben ser representadas como Ecuaciones Diferenciales Ordinarias (ODE):

1. **Aplanado del modelo**.
2. **Reducción de las conexiones**.

Tamaño de un modelo

- Tamaño físico: Cantidad de clases, variables, ecuaciones, etc. usadas.
- Tamaño lógico: Consideramos la dimensionalidad del modelo.

Tamaño de un modelo

- Tamaño físico: Cantidad de clases, variables, ecuaciones, etc. usadas.
- Tamaño lógico: Consideramos la dimensionalidad del modelo.

Queremos trabajar con grandes dimensionalidades

Contenido de la charla

Modelo Aplanado

1. Modelo monolítico.
2. Carencias de clases.
3. Variables de tipo básicos.
4. No posee ecuaciones *connect*.

Simulación de Modelos Modelica

```
1  package Circuits
2      model LC_circuit
3          capacitor cap(v(start = 1));
4          inductor ind(L = 2);
5          Pin p1,p2,p3;
6      equation
7          connect(ind.p,p3);
8          connect(ind.p,cap.p);
9          connect(cap.n,p1);
10         connect(ind.n,p2);
11     end LC_circuit;
12
13     model LC_line
14         constant Integer N = 10;
15         LC_circuit lc[N];
16         ground gr;
17     equation
18         connect(lc[N].p1,lc[N].p2)
19         for i in 1:N - 1 loop
20             connect(lc[i + 1].p3,lc[i].p2);
21         end for;
22         for i in 1:N loop
23             connect(gr.p,lc[i].p1);
24         end for;
25     end LC_line;
26 end Circuits;
```

Algoritmo de Aplanado

```
1  Flat(C):
2      Expand(C);
3      foreach v in Variables(C):
4          t = ResolveType(v);
5          if isBasic(t) then
6              ChangeType(v,t);
7          else if isClass(t) AND NOT isConnector(t) then
8              ApplyModification(C,t,Modification(v));
9              Flat(t);
10             RemoveComposition(C,t);
11             if isConnector(t) then
12                 ChangeType(v,t);
13             else
14                 Remove(t);
15             end if;
16         end if;
17     end foreach;
18     foreach e in Equations(C):
19         ChangeVarName(e);
20     foreach e in Statements(C):
21         ChangeVarName(e);
```

Expand

Eliminación de Herencia de clases

La clase hijo hereda las variables y ecuaciones del padre

Expand

Eliminación de Herencia de clases

La clase hijo hereda las variables y ecuaciones del padre

```
1  model OnePort
2      Pin p;
3      Pin n;
4      Real v;
5      Real i;
6  equation
7      v = p.v - n.v;
8      i = p.i;
9      i = -n.i;
10 end OnePort;
11 model Capacitor
12     extends OnePort;
13     parameter Real C = 1;
14 equation
15     C * der(v) = i;
16 end Capacitor;
```

Expand

Eliminación de Herencia de clases

La clase hijo hereda las variables y ecuaciones del padre

```
model OnePort
    Pin p;
    Pin n;
    Real v;
    Real i;
equation
    v = p.v - n.v;
    i = p.i;
    i = -n.i;
end OnePort;
model Capacitor
    extends OnePort;
    parameter Real C = 1;
equation
    C * der(v) = i;
end Capacitor;
```

```
model Capacitor
    Pin p;2
    Pin n;
    Real v;
    Real i;2
    parameter Real C = 1;
equation
    v = p.v - n.v;
    i = p.i;
    i = -n.i;
    C * der(v) = i;
end Capacitor;
```

ResolveType

Determina el tipo real de una variable y sus características:

- Prefijos de Tipos.
- Definición de Arreglos.
- Presencia de modificaciones.

1. Buscar en la clase.
2. Si no esta, subo al padre. Repito.
3. Si es sinónimo, aplico ResolveType al nuevo nombre.
4. Devuelvo el tipo encontrado.

ResolveType: Ejemplo

Definición de *Voltage*

```
1  package Circuits
2      model Capacitor
3          extends OnePort;
4          parameter Real C(start = 1);
5      equation
6          C * der(v) = i;
7      end Capacitor;
8
9      model LC_circuit
10         Capacitor cap(v(start = 1));
11         inductor ind(L = 2);
12         Pin p1,p2,p3;
13     equation
14         connect(ind.p,p3);
15         connect(ind.p,cap.p);
16         connect(cap.n,p1);
17         connect(ind.n,p2);
18     end LC_circuit;
19 end Circuits;
```

ResolveType: Ejemplo

Definición de *Voltage*

```
1  package Circuits
2      model Capacitor
3          extends OnePort;
4          parameter Real C(start = 1);
5      equation
6          C * der(v) = i;
7      end Capacitor;
8
9      model LC_circuit =
10         Capacitor cap(v(start = 1));
11         inductor ind(L = 2);
12         Pin p1,p2,p3;
13     equation
14         connect(ind.p,p3);
15         connect(ind.p,cap.p);
16         connect(cap.n,p1);
17         connect(ind.n,p2);
18     end LC_circuit;
19 end Circuits;
```

ResolveType: Ejemplo

Definición de *Voltage*

```
1  package Circuits ←
2      model Capacitor
3          extends OnePort;
4          parameter Real C ← 1;
5      equation
6          C * der(v) ← i;
7      end Capacitor;
8
9      model LC_circuit
10         Capacitor cap(v(start ← 1));
11         inductor ind(L ← 2);
12         Pin p1,p2,p3;
13     equation
14         connect(ind.p,p3);
15         connect(ind.p,cap.p);
16         connect(cap.n,p1);
17         connect(ind.n,p2);
18     end LC_circuit;
19 end Circuits;
```

ResolveType: Ejemplo

Definición de *Voltage*

```
1  package Circuits ←
2      model Capacitor
3          extends OnePort;
4          parameter Real C ← 1;
5      equation
6          C * der(v) ← i;
7      end Capacitor;
8
9      model LC_circuit
10         Capacitor cap(v(start ← 1));
11         inductor ind(L ← 2);
12         Pin p1,p2,p3;
13     equation
14         connect(ind.p,p3);
15         connect(ind.p,cap.p);
16         connect(cap.n,p1);
17         connect(ind.n,p2);
18     end LC_circuit;
19 end Circuits;
```

ApplyModification

1. Expandir la clase.
2. Buscar variable y agregar modificaciones.

Capacitor c (C=5,v(start=2))

ApplyModification

1. Expandir la clase.
2. Buscar variable y agregar modificaciones.

Capacitor c (C=5,v(start=2))

```
1  model Capacitor
2      Pin p;
3      Pin n;
4      Real v;
5      Real i;
6      parameter Real C = 1;
7  equation
8      v = p.v - n.v;
9      i = p.i;
10     i = -n.i;
11     C * der(v) = i;
12 end Capacitor;
```

```
1  model Capacitor
2      Pin p;
3      Pin n;
4      Real v (start=2);
5      Real i;
6      parameter Real C = 5;
7  equation
8      v = p.v - n.v;
9      i = p.i;
10     i = -n.i;
11     C * der(v) = i;
12 end Capacitor;
```

Remove Composition

1. Elimina instancia de clase.
2. Añade las variables y ecuaciones internas.
3. Renombrar las variables.
4. Si la instancia está vectorizada:
   1. Agrega las variables vectorizadas.
   2. Encapsulamos las ecuaciones dentro de una ecuación *for*.

Remove Composition: Variables

1. Agregamos un prefijo al nombre de las variables: "nombreInstancia_".
2. Mantenemos prefijos de variable.
3. Mantenemos las definiciones de arreglos y agregamos nuevas si la instancia lo está.

Remove Composition: Variables

1. Agregamos un prefijo al nombre de las variables: "nombreInstancia_".
2. Mantenemos prefijos de variable.
3. Mantenemos las definiciones de arreglos y agregamos nuevas si la instancia lo está.

```
model Capacitor
    Real p_v;
    flow Real p_i;
    Real n_v;
    flow Real n_i;
    Real v;
    Real i;
    parameter Real C = 1;
equation
    ...
end Capacitor;
model LC_circuit
    capacitor cap(v(start = 1));
    inductor ind(L = 2);
    Pin p1,p2,p3;
equation

```

```
model LC_circuit
    Real cap_p_v;
    flow Real cap_p_i;
    Real cap_n_v;
    flow Real cap_n_i;
    Real cap_v (start = 1);
    Real cap_i;
    parameter Real cap_C = 1;

    inductor ind(L = 2);
    Pin p1,p2,p3;
equation
    ...
end LC_circuit;
```

Contenido de la charla

Resolución de conexiones

Esta etapa se divide en tres secciones:

1. Generación de un grafo vectorizado a partir de los connects.
2. Determinación de componente conexas del grafo generado.
3. Generación de ecuaciones a partir de las soluciones del punto anterior.

Grafo Vectorizado

Imagen de ejemplo de un grafo vectorizado

Grafo Vectorizado

1. Agregamos un nodo por cada variable. Si el flujo del conector (variable) es hacia el exterior, agregamos la variable con signo negativo. Si ya había sido agregada sin signo negativo, generamos un nuevo nodo.
2. Agregamos un nodo que representa a la ecuación *connect*.
3. Si la ecuación estaba dentro de un *for*, etiquetamos el nodo *connect* con el rango de iteración.
4. Agregamos dos aristas, entre cada variable y el nodo que representa al *connect*.
5. Por cada arista, si la variable asociada tiene un índice de acceso, agregamos esa referencia a la arista.
6. Normalizamos la variable iteradora. Es decir, llevamos a todas al mismo nombre de variable.

Grafo Vectorizado

Poner codigo y grafo de ejemplo