

Aplanado eficiente de grandes modelos Modelica

Mariano Botta

FCEIA, UNR

Agosto 2015

Contenido de la charla

- 1 Motivaciones y Objetivos
- 2 Conceptos Previos
- 3 Algoritmo de aplanado
- 4 Resolución de conexiones

Motivaciones

- Modelado, Simulación y Control en Tiempo Real con Aplicaciones en Electrónica de Potencia.
- Trabajar con sistemas a grandes escalas.
- Simulación en paralelo utilizando los métodos de cuantificación de estado.
- Aprovechar las ventajas de Modelica para describir modelos grandes.

Objetivos

- Mantener las características de Modelica en las sucesivas etapas de compilación.
- Específicamente, en la etapa de Aplanado.
- Mantener definiciones de arreglos y ecuaciones *for* en:
 - 1 Reducción de clases.
 - 2 Resolución de conexiones.

Contenido de la charla

- 1 Motivaciones y Objetivos
- 2 Conceptos Previos**
- 3 Algoritmo de aplanado
- 4 Resolución de conexiones

Modelica

- Lenguaje de modelado orientado a objetos.
- Modelado de sistemas complejos, con componentes mecánicos, eléctricos, electrónicos, hidráulicos, térmicos, etc.
- Desarrollado por la asociación sin fines de lucro “Modelica Association”.
- Los modelos son descritos en texto plano.
- Entornos de desarrollo: OpenModelica, MathModelica, Dymola, etc.
- Librería con componentes ya definidos.

Clases

- Define un objeto.
- Son instanciadas mediante la definición de variables.
- Tienen tres secciones:
 - 1 Definiciones.
 - 2 Ecuaciones.
 - 3 Sentencias.
- Clases especializadas: *model*, *record*, *block*, *connector*, *function*, *package*

```
class X
    // Definiciones de
        variables y clases
equation
    // Ecuaciones
statements
    // Sentencias
end X;
```

Herencias de Clases

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

La clase hijo obtiene las características 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;
```


Tipos de Variables

- **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*.

```
package Circuits
  type Current = flow Real;
  type Voltage = Real;
  connector Pin
    Voltage v;
    Current i;
  end Pin;
  type TenPin = Pin[10];
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.**

```
type TenPin = Pin[10];
```

```
TenPin pines;  
Pin pines2 [10];
```

Modificaciones

Aparecen en

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

Podemos

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

```
package Circuits
  model CircuitX
    Capacitor cap;
    Resistor res;

    ...

  equation

  ...

end CircuitX;
model MainCircuit
  Capacitor x(C = 2);
  CircuitX co1 (cap(C = 10));
  CircuitX co2 (cap(C = 15));
end MainCircuit;
end Circuits;
```

Ecuaciones

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.

Ecuaciones de Igualdad

```
p.v - n.v = v;  
i = p.i;  
i = -n.i;
```

Ecuación *for*

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

N = 4

```
v[1] = p[1].v - n[1].v;  
v[2] = p[2].v - n[2].v;  
v[3] = p[3].v - n[3].v;  
v[4] = p[4].v - n[4].v;
```

Ecuaciones *connect*

Conectores

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

Ejemplo

Clase Pin:

- Voltaje: Variables de potencial.
- Corriente: Variables de flujo.

Ecuaciones *connect*

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.

```
model LC_circuit
  Capacitor cap(v(start = 1));
  inductor ind(L = 2);
  ground gr;
equation
  connect(ind.p, cap.p);
  connect(ind.n, cap.n);
  connect(cap.n, gr.p);
end LC_circuit;
```

```
// Variables de Potencial
ind.n.v = cap.n.v;
cap.n.v = gr.p.v;
```

```
// Variables de flujo
ind.n.i + cap.n.i + gr.p.i = 0;
```

Tamaños de un modelo

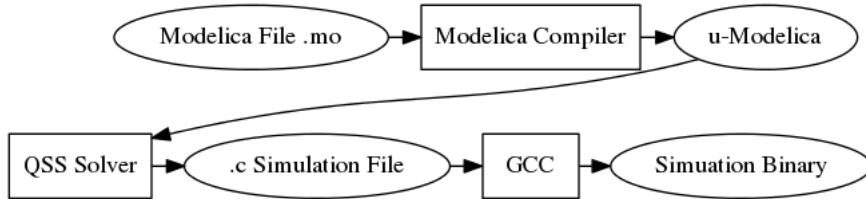
Tamaño de la descripción

Consideramos la cantidad de clases, variables, ecuaciones, etc. usadas.

Tamaño del sistema

Consideramos la dimensionalidad del modelo.
Cantidad total de variables.

Simulación de Modelos Modelica



Simulación de Modelos Grandes

Problemas

Expandir las variables y ecuaciones vectorizadas provoca una pérdida de eficiencia en las etapas de compilación.

Imposibilidad de trabajar con sistemas de gran tamaño.

Objetivos

Algoritmo de aplanado con costo computacional constante con respecto a la dimensionalidad del modelo.

Mantener las definiciones de arreglos y ecuaciones *for* durante toda la etapa de compilación.

Contenido de la charla

- 1 Motivaciones y Objetivos
- 2 Conceptos Previos
- 3 Algoritmo de aplanado**
- 4 Resolución de conexiones

Modelo aplanado

Modelo aplanado

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

Algoritmo de aplanado

Esta dividido en dos etapas:

Reducción de composiciones

- 1 Expansión de herencia.
- 2 Simplificación de tipos.
- 3 Aplicación de modificaciones.
- 4 Reducción de instancias.

Resolución de conexiones

Descomposición de ecuaciones *connect* en ecuaciones simples.

Algoritmo de aplanado

Reducción de composiciones

Algoritmo de Aplanado - Reducción de composiciones

```
Flat(C):  
  Expand(C);  
  foreach v in Variables(C):  
    t = ResolveType(v);  
    if isBasic(t) then  
      ChangeType(v,t);  
    else if isClass(t) AND NOT isConnector(t) then  
      ApplyModification(C,t,Modification(v));  
      Flat(t);  
      RemoveComposition(C,t);  
      if isConnector(t) then  
        ChangeType(v,t);  
      else  
        Remove(t);  
      end if;  
    end if;  
  end foreach;  
  foreach e in Equations(C):  
    ChangeVarName(e);
```

Expansión de herencias de clases

La clase hijo hereda las variables y ecuaciones del padre.

Expansión de herencias 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;
```


Expansión de herencias 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;
```

Simplificación de tipos - ResolveType

Determina el tipo real de una variable y sus características:

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

Simplificación de tipos - ResolveType

Determina el tipo real de una variable y sus características:

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

```
package Circuits
  model Capacitor
    extends OnePort;
    parameter Real C ← 1;
  equation
    C * der(v) ← i;
  end Capacitor;

  model LC_circuit ←
    Capacitor cap(v(start ← 1));
    inductor ind(L ← 2);
    Pin p1,p2,p3;
  equation
    ...
  end LC_circuit;
```

Simplificación de tipos - ResolveType

Determina el tipo real de una variable y sus características:

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

```
package Circuits ←  
  model Capacitor  
    extends OnePort;  
    parameter Real C ← 1;  
  equation  
    C * der(v) ← i;  
  end Capacitor;  
  
  model LC_circuit  
    Capacitor cap(v(start ← 1));  
    inductor ind(L ← 2);  
    Pin p1,p2,p3;  
  equation  
    ...  
  end LC_circuit;
```

Simplificación de tipos - ResolveType

Determina el tipo real de una variable y sus características:

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

```
package Circuits ←  
  model Capacitor  
    extends OnePort;  
    parameter Real C ← 1;  
  equation  
    C * der(v) ← i;  
  end Capacitor;  
  
  model LC_circuit  
    Capacitor cap(v(start ← 1));  
    inductor ind(L ← 2);  
    Pin p1,p2,p3;  
  equation  
    ...  
  end LC_circuit;
```

Aplicación de modificaciones - ApplyModification

- 1 Expandir la clase.
- 2 Agregar las modificaciones a la variable correspondiente.

Aplicación de modificaciones - ApplyModification

- 1 Expandir la clase.
- 2 Agregar las modificaciones a la variable correspondiente.

Capacitor c ($C=5, v(\text{start}=2)$)

Aplicación de modificaciones - ApplyModification

- 1 Expandir la clase.
- 2 Agregar las modificaciones a la variable correspondiente.

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

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

```
model Capacitor
  Pin p;
  Pin n;
  Real v (start=2);
  Real i;
  parameter Real C = 5;
equation
  v = p.v - n.v;
  i = p.i;
  i = -n.i;
  C * der(v) = i;
end Capacitor;
```


Reducción de instancias - RemoveComposition

RemoveComposition

- ① Reemplaza las instancias de clases (previamente aplanada).
- ② Añade las variables y ecuaciones internas.
- ③ Renombrar las variables añadidas.
- ④ Si la instancia está vectorizada:
 - ① Agrega las variables vectorizadas.
 - ② Encapsulamos las ecuaciones dentro de una ecuación *for*.

RemoveComposition

Variables

- 1 Agrega un prefijo al nombre de las variables: "nombreInstancia_".
- 2 Mantiene los prefijos de tipos de las variable.
- 3 Mantiene las definiciones de arreglos y agrega nuevas si la instancia lo está.

Ecuaciones

- 1 Renombra las variables que correspondan.
- 2 Reemplaza el operador "." por guiones.
- 3 Encapsula las ecuaciones en un *for* si la instancia esta vectorizada.

Algoritmo de Aplanado - Reducción de composiciones

```
Flat(C) :  
  Expand(C) ;  
  foreach v in Variables(C) :  
    t = ResolveType(v) ;  
    if isBasic(t) then  
      ChangeType(v, t) ;  
    else if isClass(t) AND NOT isConnector(t) then  
      ApplyModification(C, t, Modification(v)) ;  
      Flat(t) ;  
      RemoveComposition(C, t) ;  
      if isConnector(t) then  
        ChangeType(v, t) ;  
      else  
        Remove(t) ;  
      end if ;  
    end if ;  
  end foreach ;  
  foreach e in Equations(C) :  
    ChangeVarName(e) ;
```

RemoveComposition: Ejemplos

```
package Circuits
  model LC_circuit
    Pin p1,p2,p3;
    equation
      p1.v = p2.v;
      p2.v = p3.v;
    end LC_circuit;

    model LC_line
      constant Integer N = 10;
      LC_circuit lc[N];
      ground gr;
    equation
      connect(lc[N].p1,lc[N].p2)
      for i in 1:N-1 loop
        connect(lc[i+1].p3,lc[i].p2);
      end for;
      for i in 1:N loop
        connect(gr.p,lc[i].p1);
      end for;
    end LC_line;
  end Circuits;
```

```
package Circuits
  model LC_circuit
    Pin p1,p2,p3;
    flow Real p1_i,p2_i,p3_i;
    Real p1_v,p2_v,p3_v;
    equation
      p1_v = p2_v;
      p2_v = p3_v;
    end LC_circuit;
  end Circuits;
```

RemoveComposition: Ejemplos

```
model LC_circuit
  Pin p1,p2,p3;
  flow Real p1_i,p2_i,p3_i;
  Real p1_v,p2_v,p3_v;
equation
  p1_v = p2_v;
  p2_v = p3_v;
end LC_circuit;

model LC_line
  constant Integer N = 10;
  LC_circuit lc[N];
  ground gr;
equation
  connect(lc[N].p1,lc[N].p2)
  for i in 1:N - 1 loop
    connect(lc[i + 1].p3,lc[i].p2);
  end for;
  for i in 1:N loop
    connect(gr.p,lc[i].p1);
  end for;
end LC_line;
```

```
package Circuits
  model LC_line
    constant Integer N = 10;
    Pin p1[N],p2[N],p3[N];
    flow Real lc_p1_i[N],lc_p2_i[N],lc_p3_i[N];
    Real lc_p1_v[N],lc_p2_v[N],lc_p3_v[N];
    ground gr;
  equation
    for i in 1:N - 1 loop
      p1_v[N] = p2_v[N];
      p2_v[N] = p3_v[N];
    end for;
    connect(lc_p1[N],lc_p2[N])
    for i in 1:N - 1 loop
      connect(lc_p3[i + 1],lc_p2[i]);
    end for;
    for i in 1:N loop
      connect(gr_p[i],lc_p1[i]);
    end for;
  end LC_line;
end Circuits;
```

Contenido de la charla

- 1 Motivaciones y Objetivos
- 2 Conceptos Previos
- 3 Algoritmo de aplanado
- 4 Resolución de conexiones

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

- ➊ 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.
- ➋ Agregamos un nodo que representa a la ecuación *connect*.
- ➌ Si la ecuación estaba dentro de un *for*, etiquetamos el nodo *connect* con el rango de iteración.
- ➍ Agregamos dos aristas, entre cada variable y el nodo que representa al *connect*.
- ➎ Por cada arista, si la variable asociada tiene un índice de acceso, agregamos esa referencia a la arista.
- ➏ Normalizamos la variable iteradora. Es decir, llevamos a todas al mismo nombre de variable.

Grafo Vectorizado

Poner codigo y grafo de ejemplo