COMPARACIÓN ENTRE MODELICA 2.0 Y ECOSIMPRO/EL 3.2

Luis José Yebra Muñoz
CIEMAT-PSA. Apdo. 22, E-04200, Tabernas, Almería, Spain, luis.yebra@psa.es

Ramón Perez Vara.
Empresarios Agrupados, Magallanes 3, 28015 Madrid, Spain
e-mail: rpv@empre.es

Sebastián Dormido Bencomo
Dpto. Informática y Automática, Facultad de Ciencias
UNED, Fac. de Ciencias, Dpto. de Informática y Automática. Avda. Senda del Rey 9, 28040 Madrid, Spain
e-mail: sdormido@dia.uned.es

Manuel Berenguel Soria
Dpto. Lenguajes y Computación, Escuela Politécnica Superior
Universidad de Almería, Ctra. Sacramento s/n, 04120 Almería, Spain
e-mail: beren@ual.es

Resumen

Modelica y EcosimPro/EL son lenguajes orientados a objeto y basados en ecuaciones para el modelado de sistemas físicos continuos y de eventos discretos. En este artículo se presenta un estudio comparativo entre ambos lenguajes a nivel general, en el que se pretende poner de manifiesto las distintas características que cada lenguaje incorpora. Para realizar esta labor se han intentado abstraer las propiedades más significativas de los lenguajes de modelado orientados a objeto en general y verificar cuales de ellas están presentes en ambos lenguajes. Porteriormente se implementa el modelo de un sistema de control sencillo en ambos lenguajes, comentando con cierto grado de detalle las propiedades que se han utilizado en cada una de las implementaciones. Finalmente se presentan unas conclusiones indicando aquellas características que podrían ser incorporadas o modificadas en ambos lenguajes con la intención de aumentar las prestaciones que ambos ofrecen para modelado y simulación de sistemas.

Palabras Clave: modelado orientado a objetos, EcosimPro, EL, Dymola, Modelica.

1 INTRODUCCIÓN

El presente trabajo pretende hacer un estudio comparativo entre dos lenguajes de modelado de sistema físicos existentes en la actualidad, Modelica y EcosimPro/EL (en adelante EL). El primero ha sido desarrollado por "Modelica Association", una organización sin animo de lucro con sede en Linköping, Suecia. La creación de dicho lenguaje por parte de esta asociación se debe a la necesidad de definir un lenguaje de modelado y simulación estándar, fruto del acuerdo entre un grupo de especialistas de estas disciplinas.

Por otra parte, EL es un lenguaje de modelado y simulación de sistemas físicos desarrollado por EA International para su herramienta de modelado EcosimPro.

Ambos lenguajes son fundamentalmente declarativos y orientados a objetos, y contienen todos los elementos necesarios para modelar y simular sistemas dinámicos continuos, discretos e híbridos de manera acausal.

Para realizar la comparación se ha utilizado como referencia Modelica 2.0, tomando como base la última versión del documento público que lo define ([1]) y para EL la documentación incorporada en EcosimPro [2]. En este análisis se van a obviar detalles de ambos lenguajes que estén relacionados con características que no sean relevantes en las actividades de modelado y simulación. Un estudio comparativo exhaustivo y metículoso podría generar una enorme cantidad de documentación. En este trabajo se presentan de forma sintética conceptos de modelado y simulación de sistemas de ambos lenguajes y la forma en que los implementa cada uno de ellos.

Posteriormente se modela y simula un sistema de control de velocidad angular en ambos lenguajes. El sistema es especialmente interesante e ilustrativo porque es sencillo pero plantea diversas necesidades de modelado que se abordan de distintas formas en

cada uno de los lenguajes y se proponen extensiones a los mismos.

Finalmente se concluye el trabajo con una sección que resume las diferencias fundamentales entre ambos lenguajes y propone extensiones a los mismos.

2 COMPARACIÓN MODELICA 2.0 VS. ECOSIMPRO/EL 3.2

En esta sección se realiza una comparación general entre ambos lenguajes. Una comparación exhaustiva que incluyese aspectos sintácticos generaría un documento tan extenso como pobre en información útil, por lo que se va a centrar en los aspectos diferenciadores entre ambos lenguajes a nivel conceptual. La comparación se realizará en el siguiente orden: tipos y propiedades de datos, sentencias, terminales, clases de componentes, funciones, librerías, particiones y definición de experimentos. Para cada una de las secciones se utilizarán tablas que permitan sintetizar y resumir propiedades y características en ambos lenguajes.

2.1 TIPOS Y PROPIEDADES DE LOS DATOS

2.1.1 Tipos de datos fundamentales

En la Tabla 1 se describen los tipos de variables fundamentales en ambos lenguajes.

Tabla 1: Tipos fundamentales de variables.

Tipo	Modelica	EL
Booleano	Boolean	BOOLEAN
Entero	Integer	INTEGER
Real	Real	REAL
Cadena	String	STRING
Tabla 1D	NO	TABLE_1D
Tabla 2D	NO	TABLE_2D
Tabla 3D	NO	TABLE_3D
Punteros a funciones	NO	FUNC_PTR

EL contempla los tipos Tabla y punteros a funciones como tipos base del lenguaje. Modelica no los incorpora como tales, aunque proporciona mecanismos para implementarlos como tipos compuestos. Por ejemplo, las tablas de datos para interpolación en Modelica se pueden implementar codificándolas o utilizando librerías ya existentes que ofrezcan esta funcionalidad, como la librería 'Modelica Additions'.

En cuanto a los tipos puntero a función de EL, Modelica no los contempla, aunque proporciona mecanismos suficientes para sustituir su funcionalidad. En general, ya sea en lenguajes de programación o de modelado, los punteros a funciones se utilizan para expresar un comportamiento paramétrico que se puede pasar como argumento a clases o funciones. Esta funcionalidad se puede implementar mediante la utilización de clases y/o componentes paramétricos en Modelica.

Para Modelica todos los tipos base se consideran clases preconstruidas compuestas por una serie de atributos que modelan su comportamiento. En [1] se especifican todos los atributos de los tipo base. Mediante la utilización de dichos atributos se puede:

- Asignar una cantidad física a la que el tipo está haciendo referencia, ej: voltaje, corriente, temperatura,...
- Definir las unidades que la variable está representando, así como las unidades en las que se mostrará esta variable, ej: las unidades representadas pueden ser m/s, mientras las variables en las que se visualizará la variable en algún gráfico puede ser Km/h.
- Definir intervalos de validez de la variable, fuera de los cuales la herramienta generaría una advertencia o error durante la simulación del modelo.
- Atributos para asignar el valor inicial y parámetros para condicionar a la herramienta en la asignación del valor inicial de dicha variable.
- En el caso de tipos reales, existen atributos que permiten definir valores nominales para la variable con que permitan normalizar el valor de las variables de este tipo ó adecuar los vectores de tolerancias de los resolvedores DAE. Otros atributos permiten a los tipos reales, o subtipos de éstos, condicionar a los manipuladores simbólicos en la elección de variables de estado en caso de problemas de índice superior.

Estos atributos están definidos en el lenguaje Modelica para que las herramientas que lo implementen puedan utilizarlos, aunque no necesariamente lo hagan. La herramienta utilizada con Modelica por los autores es Dymola. En su documentación ([3]) no se especifica que todas las funcionalidades que estos atributos ofrecen sean utilizados como en [1] se indica.
EL no trata los tipos fundamentales como clases, sino como tipos básicos del lenguaje que formarán parte de otros componentes.

2.1.2 Tipos de datos enumerados y derivados

En esta sección se presentan las posibilidades que ofrecen ambos lenguajes respecto los tipos enumerados, subconjuntos y arrays. En la Tabla 2 aparecen esquematizadas.

Tabla 2: Tipos enumerados y derivados.

Tipo	Modelica	EL
Enumeraciones	SI	SI
Subconjuntos	NO	SI
Arrays	SI	SI

Conviene comentar algunas diferencias respecto a los tipos tratados. En ambos casos están totalmente soportadas las enumeraciones. Los tipos subconjuntos sólo están definidos para EL.

La diferencia más importante, y quizás una desventaja clara, la muestra EL respecto a la limitación de los elementos que pueden estar contenidos en un array. EL restringe los tipos básicos y enumerados como elementos en un array. En cambio Modelica permite que además de los tipos básicos y enumerados, las clases o modelos puedan ser elementos de un array. Esta restricción en EL puede llegar a plantear limitaciones en el modelado de sistemas formados por estructuras regulares dispuestas en array e interconectadas entre sí. En general, modelos de parámetros distribuidos pueden ser modelados con arrays de componentes de forma muy modular.

2.1.3. Propiedades de los datos. Variabilidad.

La variabilidad de los distintos tipos de variables están presentes en ambos lenguajes. La única diferencia existente entre ambos lenguajes es a nivel sintáctico. Ambos permiten definir variables con distintas variabilidades:

- Constantes. Una vez declaradas e inicializadas nunca cambian su valor.
- Parámetros. Una vez declaradas e inicializadas no cambian su valor durante una simulación, aunque lo pueden hacer de una simulación a otra.
- Variables discretas. Cambian su valor en el transcurso de la simulación solamente en los instantes en los que se producen eventos discretos. En los intervalos de tiempo entre eventos mantienen su valor.
- Variables continuas. Cambian su valor en cualquier instante de la simulación

2.1 SENTENCIAS

En general, las sentencias de las que se compone el código de los lenguajes de modelado de sistemas físicos se pueden clasificar en sentencias secuenciales, continuas y discretas.

Las sentencias secuenciales se ejecutan en un orden predefinido dentro de la sección de código a la que pertenecen. Son las sentencias típicas en los lenguajes de programación como FORTRAN o C++.

Las sentencias continuas describen comportamientos de sistemas dinámicos continuos. Normalmente son relaciones matemáticas que ligan unas variables con otras. En general son conjuntos de ecuaciones no necesariamente continuas.
Las sentencias discretas describen discontinuidades explícitamente definidas mediante expresiones relacionales o lógicas. Este tipo de sentencias permiten declarar la existencia de discontinuidades en los modelos continuos, o modelar sistemas de eventos discretos directamente. Se ejecutan en cada paso de integración calculado por el resolvedor numérico.

Los conceptos de sistemas continuos, discretos e híbridos se exponen con bastante claridad en [4] y [8] .

En la Tabla 3 se puede encontrar un resumen de los tipos de sentencias que proporcionan ambos lenguajes. En dicha exposición se sigue el orden descrito en [2].

Respecto al tipo de sentencias, tanto Modelica como EL presentan construcciones similares que permiten implementar las funcionalidades descritas.

2.2 TERMINALES

Los terminales permiten comunicar unos componentes con otros dentro de un mismo sistema. En la Tabla 4 se muestran las propiedades que uno y otro lenguaje proporcionan en las implementaciones de este tipo de componentes.

En este tipo de componentes, EL muestra una mayor flexibilidad que Modelica. Se permite definir mediante las palabras reservadas ‘IN’ y ‘OUT’ en EL la direccionabilidad de las variables Through de los terminales.

Tabla 3: Tipos de sentencias

Tipo	Sentencias	Modelica	EL
Secuenciales	Asignación	SI	SI
	Llamada función	SI	SI
	If-then-else	SI	SI
	While	SI	SI
	For	SI	SI
	Aserciones	SI	SI
	Return	Implícito	SI
Continuas	Ecuaciones	SI	SI
	Expansión simbólica de ecuaciones	SI	SI
	Expansión simbólica de grupos de ecuaciones	SI	SI
	Ecuaciones definidas por zonas	SI	SI
Discretas	Detección de eventos por condición	SI	SI
	Aserciones discretas	SI	SI
	Expansión simbólica de sentencias discretas	SI	SI
	Expansión simbólica de bloques sentencias discretas	SI	SI

Tabla 4: Terminales

Característica	Modelica	EL
Parametros	SI	SI
Variables Across	SI	SI
Variables Through	SI	SI
Direccionabilidad programable	NO	SI
Sentencias continuas en el terminal	NO	SI
Restricción de conexiones por causalidad	SI	SI
Restricción adicional de conexiones	NO	SI

La definición de variables 'across' y 'through' se expone con claridad en [4], aunque se puede resumir que las variables 'across' son los potenciales asociados a los nodos de interconexión de componentes, y la 'through' son los flujos asociados a dichos potenciales y que entran o salen de los componentes a través de sus terminales.

EL permite introducir ecuaciones que relacionan las variables y restringir mediante el modificador 'SINGLE' el número de conexiones de un puerto.

En Modelica, la direccionabilidad está definida en el lenguaje por convenio, el flujo en las variables Through siempre entra en el componente.

Tampoco se permite que en los terminales existan ecuaciones. De hecho, todos los modelos en Modelica son clases, y los 'connector' son clases que están restringidas a no tener bloques de código que contengan sentencias continuas, ni secuenciales (ni, por tanto, discretas). Las únicas limitaciones en la conexión de terminales son las definidas por las causalidades de éstos.

2.3 CLASES DE COMPONENTES

Las clases o componentes son el elemento fundamental en los lenguajes de modelado orientados a objeto. Constituyen la herramienta básica para representar el comportamiento, desde sistemas simples e indivisibles, hasta sistemas muy complejos formados por otros subsistemas.

2.4.1 Bloques de código en las clases

En ambos lenguajes las clases se definen mediante un conjunto de secciones de código. Cada sección tiene un objetivo determinado en cada uno de los lenguajes. En la Tabla 5 se enumeran las secciones de las clases en ambos lenguajes.

Tabla 5: Secciones de código en Modelica y EL

Modelica	EL
declarativa	PORTS
protected	DATA
equations	DECLS
initial equations	TOPOLOGY
algorithm	INIT
	DISCRETE
	CONTINUOUS

La sección 'declarativa' en Modelica es la sección inicial que existe después de la definición del nombre de la clase. No tiene ninguna etiqueta que la identifique, y es el lugar en el que se declaran las variables y componentes de la clase.

No existe una correspondencia única entre secciones de ambos lenguajes. Para proporcionar una información clara de qué tipo de código se incluye en cada sección de forma sintética, se presenta la Tabla 6.

Tabla 6: Correspondencia entre secciones

Declaración de:	Modelica	EL
Parámetros	declarativa. ('parameter')	DATA
Terminales	declarativa	PORTS
Variables protegidas	protected	DECLS
Componentes	declarativa	TOPOLOGY
Conexión entre componentes	equation	TOPOLOGY
Inicialización de variables	initial equation o atributos 'start' de variables	INIT
Eventos	equation y algorithm	DISCRETE
Sentencias continuas, ecuaciones	equation	CONTINUOUS
Sentencias secuenciales	algorithm	INIT

Es necesario realizar algunas aclaraciones sobre la Tabla 6. En EL los parámetros se pueden declarar tanto en la sección DATA como en la definición del identificador del componente.

Respecto a la inicialización de variables, en Modelica se puede hacer mediante dos vías: asignación de atributos 'start' y 'fixed' de las variables a inicializar, o definiendo ecuaciones para el instante inicial en la sección 'initial equation'. En ambos casos las sentencias utilizadas son de carácter declarativo por lo que la herramienta realizará manipulaciones algebraicas necesarias para resolver el problema de valor inicial planteado en la simulación. Sin embargo, en EL, las inicializaciones se realizan en el bloque

INIT, ejecutándose secuencialmente las instrucciones de inicialización contenidas en este bloque.

En lo que respecta a la declaración de sentencias que definan eventos, en EL solamente aparecen en el bloque 'DISCRETE', mientras que en Modelica lo pueden hacer en el bloque 'equation' o 'algorithm'. De hecho, cuando aparecen en bloques 'algorithm' la definición de eventos se pueden priorizar de forma que ante una posible simultaneidad de eventos, siempre se podría tratar los eventos por un orden determinado. En [5] se detalla este tipo de construcciones.

Respecto a la utilización de sentencias secuenciales, en Modelica se destina una sección específicamente para componentes que incluyan este tipo de comportamientos. Esta sección es 'algorithm'.

2.4.2 Características avanzadas

Tanto EL como Modelica muestran ciertas características avanzadas que permiten parametrizar o variar el comportamiento entre clases. En la Tabla 7 se muestran estas características y cuales se incorporan en cada uno de los lenguajes.

Tabla 6: Correspondencia entre secciones

Característica	Modelica	EL
Herencia múltiple	SI	SI
Clases abstractas	SI	SI
Ecuaciones virtuales	NO	SI
Clases paramétricas	SI	NO

Tanto Modelica como EL soportan herencia múltiple y clases abstractas.

Las ecuaciones virtuales son un mecanismo muy flexible para modificar comportamientos de componentes que pertenecen al mismo grafo de herencia. En Modelica esta propiedad no está soportada y cuando una ecuación queda definida en una clase nunca podrá ser modificada en clases descendientes.

Las clases paramétricas son un mecanismo que permiten pasar como argumento, en tiempo de definición o instanciación, la clase a la que pueden pertenecer uno o varios componentes. Esta propiedad es de enorme potencia a la hora de reutilizar librerías que se hayan diseñado con intención de explotar esta propiedad. Por hacer una similitud con la programación orientada a objetos, las clases paramétricas son equivalentes a los 'templates' en C++ [6]. Esta característica no la tiene implementada EL.

2.5 FUNCIONES

En el caso de las funciones no existen diferencias conceptuales entre ambos lenguajes. Ambos permiten invocar funciones de lenguajes externos C y FORTRAN. En el caso de EL también se pueden invocar funciones de C++.

2.6 LIBRERÍAS

Las librerías son conceptualmente similares en ambos lenguajes. Su objetivo es el de almacenar modelos o clases ya desarrolladas para posterior utilización. Las diferencias se pueden encontrar en los detalles de implementación.

Una diferencia clara entre EL y Modelica es la inclusión de propiedades de representación gráfica de los componentes dentro de ellos mismos y en las librerías en los que se almacenan. Modelica permite a un componente asociar un icono gráfico dentro de la misma definición de éste mediante directivas 'annotation'. De esta forma, un usuario podría programar la información gráfica de su componente además de modelar su comportamiento. Por ejemplo, el elemento 'feedback' de la Figura 1 declara, en su código fuente completo, no sólo la operación de diferencia entre las dos variables de entrada, sino que añade información gráfica que le da el aspecto visual de la figura. Un diseñador de librerías podría incluso programar directamente en Modelica las primitivas gráficas que representan a la clase en un diagrama gráfico. Normalmente, la descripción gráfica del componente la realiza la misma herramienta de simulación mediante capturadores de esquemas.

Por ejemplo, en el caso de Dymola, la misma herramienta proporciona al usuario un capturador de esquemas propio y totalmente integrado en la herramienta.

EcosimPro no tiene un capturador de esquemas integrado en la herramienta. En su lugar se puede utilizar una herramienta externa, SmartSketch, que proporciona al usuario las posibilidades gráficas que EcosimPro no ofrece. Esta forma de trabajo no es tan dinámica como la ofrecida por Dymola.

2.7 PARTICIONES

La partición es un concepto solamente implementado en EcosimPro que permite dar una gran flexibilidad al diseño de experimentos y simulaciones. Es una característica implementada en la herramienta y no en el lenguaje. En la Tabla 7 se muestran las posibilidades que las particiones ofrecen.

Ninguna de las posibilidades que EcosimPro ofrece mediante las particiones se puede utilizar con Dymola/Modelica, al menos que conste en la información suministrada en la documentación del producto [3].

Tabla 7: Posibilidades basadas en particiones

Característica	Modelica	EL
Elección de condiciones de contorno	NO	SI
Elección de variables de tearing en Lazos Algebraicos	NO	SI
Selección de estados en problemas de indice superior	NO	SI
Cambiar variabilidad de parámetros	NO	SI

2.8 DEFINICIÓN DE EXPERIMENTOS

EL ofrece explicitamente la posibilidad de definir experimentos sobre los sistemas a simular mediante bloques de código secuencial que permite: declarar variables, inicializar variables algebraicas en los modelos, definir condiciones de contorno, cálculo de estacionarios, selección de métodos de integración, selección de tolerancias, generación de informes, invocación de funciones externas, ...

Todas estas posibilidades a la hora de realizarlas con Modelica dependen en gran medida de la herramienta con la que se trabaje. En el caso de Dymola, las funcionalidades que EL ofrece a través de los experimentos se puede alcanzar mediante los scripts de Modelica, aunque de manera más limitada.

3 EJEMPLO DE MODELADO Y SIMULACIÓN EN AMBOS LENGUAJES

3.1 INTRODUCCIÓN

Esta sección pretende presentar un ejemplo especialmente interesante a la hora de realizar comparaciones prácticas entre ambos lenguajes. El sistema propuesto ha sido extraido de [7], que propone el estudio de un sistema de control sencillo al que paulatinamente se va modificando el modelo de un componente para añadir comportamientos más reales, a la vez que se va observando cómo varía el comportamiento global del sistema.

El sistema propuesto es el de la figura 7.7 de [7] (pag. 107), titulado por el autor “Nuestro banco de pruebas de sensores”. Es un sistema que, pese a ser pequeño incorpora componentes continuos y discretos. Mediante la definición de componentes polimórficos y clases paramétricas es posible ir describiendo distintos modelos de componentes e ir paralelamente estudiando las variaciones en los resultados de la simulación.

3.2 DESCRIPCIÓN DEL SISTEMA A SIMULAR

El sistema a simular consiste en un sistema de control de velocidad angular (Figura 1). Está compuesto de un controlador PI, un actuador ideal y la planta a controlar. La planta está formada por una carga inercial conectada a una carcasa fija mediante una fricción. La variable a controlar es la velocidad de la carga. Dicha velocidad se obtiene a través de un sensor cuyo comportamiento influirá en el comportamiento global del sistema.

Se tratará de estudiar los resultados de las distintas simulaciones en las que se utilicen distintos modelos del sensor. Para ello se desarrollan tres modelos diferentes de sensor: ideal, con muestreador y retenedor de orden cero; y con muestreador-retenedor y convertidor A/D.

De todos los componentes del sistema el único que irá variando es ‘sensor’, que sucesivamente refinará su comportamiento añadiendo más realismo.

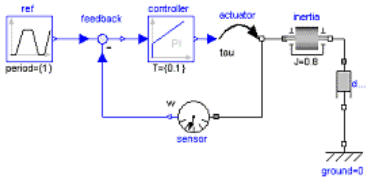

Figura 1: Banco de Pruebas

3.3 IMPLEMENTACIÓN EN MODELICA

Para implementar el sistema en Modelica todos los componentes utilizados serán los de la Modelica Standard Library (MSL). Las distintas implementaciones del sensor de velocidad están basadas en componentes de MSL aunque sus modelos varían conforme se van introduciendo características.

El código Modelica del sistema de la Figura 1 se reproduce a continuación:

```
model SensorBenchmark "Banco de pruebas"
  import Modelica.Mechanics.Rotational;
  import Modelica.Blocks;
  Blocks.Continuous.PI controller(k={100},T={0.1});
  Blocks.Math.Feedback;
  Blocks.Sources.Trapezoid ref(offset={50}, rising={0.2},
width={0.25}, falling={0.2}, amplitude={50});
  Rotational.Inertia inertia(J=0.8);
  Rotational.Damper damper(d=2);
  Rotational.Fixed ground;
  Rotational.Torque actuator ;
  replaceable Rotational.Sensors.SpeedSensor sensor
extends Rotational.Interfaces.AbsoluteSensor;
equation
  connect(ref.outPort, feedback.inPort1)
  connect(feedback.outPort, controller.inPort)
  connect(controller.outPort, actuator.inPort)
  connect(actuator.flange_b, inertia.flange_a);
  connect(inertia.flange_b, damper.flange_a);
  connect(damper.flange_b, ground.flange_b);
  connect(actuator.flange_b, sensor.flange_a);
  connect(sensor.outPort, feedback.inPort2);
end SensorBenchmark;
```

Cabe destacar en este listado la declaración del componente parametrizable ‘sensor’, precedida del especificador ‘replaceable’. Para definir distintas

clases de sistema de control con distintos modelos de sensor, será necesario crearlas como clases derivadas de 'SensorBenchmark' redeclarando el componente 'sensor'. Las clases a las que pueden pertenecer los sensores deben ser siempre subclases de 'Modelica.Mechanics.Rotational.Interfaces.Absolute Sensor'.

A continuación se definen nuevos modelos para los distintos sensores a implementar:

3.3.1 Sensor Ideal

Este sensor mide la velocidad angular real de la carga sin introducir ningún tipo de error en la medida.

El código Modelica de este componente es:

```
model SpeedSensor  "Ideal sensor to measure the absolute
flange angular velocity"
   extends Interfaces.Absolute.Sensor;
   SIunits.AngularVelocity w "Absolute angular velocity of
flange";
equation
   w = der(flange_a.phi);
   w = outPort.signal[1];
   0 = flange_a.tau;
end SpeedSensor;
```

Este componente pertenece a la MSL. Se reproduce para comentarlo brevemente. Es un modelo derivado de la clase base 'Modelica. Mechanics.Rotational.Interfaces.AbsoluteSensor'.

Añade una variable continua del tipo velocidad angular (subtipo de real) 'w' y la define como derivada de la posición angular 'phi' contenida en el conector de la clase base (flange_a). La clase base aparece en el fragmento de código que aparece a continuación:

```
partial model AbsoluteSensor  "Base class to measure a
single absolute flange variable"
  extends Modelica.Icons.RotationalSensor  "(left) flange to
be measured (flange axis directed INTO cut plane)";
  Modelica.Blocks.Interfaces.OutPort outPort(final n=1);
end AbsoluteSensor;
```

Esta clase está compuesta por dos conectores, uno mecánico (Flange_a) y otro de tipo señal de salida (OutPort).

3.3.2 Sensor con Muestreador y Retenedor

El código se muestra a continuación:

```
model SampleHoldSensor
import Modelica.Mechanics.Rotational;

   extends Rotational.Interfaces.AbsoluteSensor;
   Modelica.SIunits.AngularVelocity w;
   parameter Modelica.SIunits.Time sample_interval=0.1;
equation
   w = der(flange_a.phi);
   flange_a.tau = 0;
algorithm
   when sample(0, sample_interval) then
      outPort.signal[1] := w;
   end when;
end SampleHoldSensor;
```

Este sensor deriva de 'Modelica.Mechanics.Rotational.Interfaces.Absolute Sensor' para cumplir la restricción impuesta en el sistema de control. Además añade el código necesario para modelar el comportamiento de un muestreador y retenedor de orden cero. Dicho código fuerza a un evento cada 'sample_interval' segundos mediante la función interna de Modelica 'sample()', y asigna a la señal de salida el valor de la velocidad angular del conector mecánico. Debido a que la señal de salida está siendo asignada dentro de una sentencia 'when', su variabilidad queda implicitamente convertida a discreta, manteniendo su valor hasta el próximo evento que cumpla la condición en la sentencia when (su próxima muestra). El periodo de muestreo se define como parámetro del modelo.

3.3.3 Sensor con Muestreador, Retenedor y convertidor A/D

Este sensor modela el comportamiento adicional de sistema de adquisición de datos, basado en la función A/D. De esta forma incorpora una mayor realismo al sensor permitiendo estudiar los errores cometidos por comportamientos no modelados en los modelos anteriores. En este caso el ruido de cuantificación se añade a la medida proveniente del muestreador y retenedor. Se muestra el código del nuevo modelo:

```
model QuantizedSensor
   import Modelica.SIunits;
   import Modelica.Mechanics.Rotational;
extends Rotational.Interfaces.AbsoluteSensor;

   parameter Integer bits=4;
   parameter SIunits.Time sample_interval=0.02;
   parameter SIunits.AngularVelocity min=-150;
   parameter SIunits.AngularVelocity max=150;
   SIunits.AngularVelocity w;
protected
   parameter Real delta=(max - min)/2^bits;
   Integer level;
equation
   w = der(flange_a.phi);
   flange_a.tau = 0;
algorithm
   when sample(0, sample_interval) then
      level := integer((w - min)/delta);
   end when;
   if level < 0 then
      outPort.signal[1] := min;
   elseif level >= 2^bits then
      outPort.signal[1] := max;
   else
      outPort.signal[1] := level*delta + min;
   end if;
end QuantizedSensor;
```

Este modelo deriva de nuevo de 'Modelica.Mechanics.Rotational.Interfaces.Absolute Sensor' para cumplir la condición impuesta por el

sistema contenedor. Tiene un parámetro, el número de bits del A/D, para la cuantificación de forma que en distintas instanciaciones del componente con distintos valores del número de bits se puede observar la influencia de este parámetro en el comportamiento final del modelo.

En el código se puede seguir facilmente la implementación del funcionamiento de un convertidor A/D, en el que se definen extremos en la señal analógica a convertir (max y min) y la variable entera interna 'level' que almacena el valor de la conversión. A partir de este valor, la variable continua de salida 'outPort.signal[1]' se asigna en función la variable 'level'.

3.4 IMPLEMENTACIÓN EN ECOSIMPRO/EL

En EL no existen dos propiedades de Modelica que son utilizadas en el código anterior: componentes sustituibles y la función interna 'sample()'. Esto no supone un obstáculo sino una diferencia en el diseño del componente. Para implementar la funcionalidad de los tres sensores es necesario sustituir el modelo del sensor para cada simulación del sistema. Para que el sensor pueda muestrear se crea un componente que implemente el muestreo mediante la generación de eventos discretos. El código EL del sistema de la Figura 1 se describe a continuación:

```
USE MECH_ROTATIONAL
USE TILLER_C04

COMPONENT SensorBenchmark
TOPOLOGY
  MECH_ROTATIONAL.RotationalInertia inertia (J=0.8)
  MECH_ROTATIONAL.RotationalFixed ground
  MECH_ROTATIONAL.RotationalDamper damper (d=0.2)
  MECH_ROTATIONAL.RotationalTorque actuator
  MECH_ROTATIONAL.RotationalSpeedSensor sensor
  TILLER_C04.PIController controller (Kp = 100, Ti = 0.1)

  CONNECT ground.p TO damper.flange_b
  CONNECT damper.flange_a TO inertia.flange_b
  CONNECT actuator.flange_b TO inertia.flange_a
  CONNECT sensor.flange_a TO inertia.flange_a
  CONNECT controller.driver TO actuator.inPort
  CONNECT sensor.outPort TO controller.sensor
END COMPONENT
```

3.4.1 Sensor Ideal

Se reproduce su código EL a continuación obtenido de la librería MECH_ROTATIONAL, desarrollada por EA International:

```
ABSTRACT COMPONENT RotationalAbsoluteSensor
  PORTS
    OUT MECH_ROTATIONAL.Flange flange_a
    OUT CONTROL.analog_signal outPort
END COMPONENT

COMPONENT RotationalSpeedSensor IS_A
RotationalAbsoluteSensor
  DECLS
    REAL w
  CONTINUOUS
    w = flange_a.phi'
    w = outPort.signal
    0 = flange_a.tau
END COMPONENT
```

Siguiendo la metodología de la implementación en Modelica, el componente 'RotationalSpeedSensor' se ha desarrollado basándolo en la clase base abstracta 'RotationalAbsoluteSensor'.

3.4.2 Sensor con Muestreador y Retenedor

Para la implementación de este componente en EL es necesario modelar el comportamiento del muestreador mediante la generación de periódica de eventos. Se implementa el componente 'sampler' como clase base de otros componentes que necesiten incorporar este comportamiento, que lo podrán hacer mediante el mecanismo de herencia del que están dotados ambos lenguajes.

Código EL de componente 'sampler':

```
COMPONENT Sampler
DATA
  REAL sample_interval = 0.1
DECLS
  BOOLEAN sample = FALSE
INIT
 sample = TRUE AFTER sample_interval
DISCRETE
  WHEN(sample==TRUE) THEN
    sample = FALSE AFTER 0
    sample = TRUE AFTER sample_interval
  END WHEN
END COMPONENT
```

El componente a implementar para sustituir en el banco de pruebas es 'SampleHoldSensor'. Debe basarse en 'sampler' además de incluir código para mantener el valor de la variable continua muestreada hasta el próximo instante de muestreo.

Código EL de 'SampleHoldSensor':

```
COMPONENT SampleHoldSensor IS_A Sampler,
MECH_ROTATIONAL.RotationalAbsoluteSensor
DECLS
  REAL w
DISCRETE
  WHEN(sample==TRUE) THEN
    outPort.signal = w
  END WHEN
CONTINUOUS
  w = flange_a.phi'
  0 = flange_a.tau
END COMPONENT
```

Mediante herencia múltiple el componente incorpora el comportamiento del 'Sampler' que en el componente original 'sensor' no existía.

3.4.3 Sensor con Muestreador, Retenedor y convertidor A/D

El componente es el mismo que el codificado en la sección de Modelica. Se muestra el código EL de este componente:

```
USE MECH_ROTATIONAL

COMPONENT QuantizedSensor IS_A
MECH_ROTATIONAL.RotationalAbsoluteSensor, Sampler
DATA
  INTEGER bits = 4
  REAL min = -150
  REAL max = 150
DECLS
  REAL delta
  REAL w
  INTEGER level
DISCRETE
  WHEN (sample) THEN
    level = (w - min) / delta
    IF (level < 0) THEN
      outPort.signal = min
    ELSEIF (level >= 2**bits) THEN
      outPort.signal = max
    ELSE
      outPort.signal = level*delta + min
    END IF
  END WHEN
CONTINUOUS
  delta = (max - min) / 2**bits
  w = flange_a.phi'
  flange_a.tau = 0
END COMPONENT
```

3.5 SIMULACIÓN CON DYMOLA/ MODELICA

Para realizar la comparación del comportamiento del sistema respecto a los tres modelos de sensor se realiza una simulación con los tres componentes simultaneamente. El código Modelica del sistema a simular se muestra a continuación:

```
model Comparacion

model SamplingCase = SensorBenchmark (redeclare
SampleHoldSensor sensor(sample_interval=0.01));
 model QuantizedCase=SensorBenchmark(redeclare
QuantizedSensor sensor(sample_interval=0.01, bits=8));

  SensorBenchmark ideal;
  SamplingCase sampling(sensor(sample_interval=0.015));
  QuantizedCase quantized(sensor(sample_interval=0.015));
end Comparacion;
```

Para realizar la simulación de los tres componentes simultáneamente se define una clase que contenga a los tres. Previamente se ha hecho uso de la capacidad de Modelica de definir nuevas clases 'SamplingCase' y 'QuantizedSensor', basadas ambas en el modelo inicial 'SensorBenchmark' pero pasando como argumento la clase del sensor a utilizar. 'SamplingCase' es una clase derivada de 'SensorBenchmark' en la que la clase del sensor es un muestreador y retenedor. 'QuantizedCase' es una clase derivada de 'SensorBenchmark' en la que la clase del sensor es un muestreador, retenedor y función A/D.

Posteriormente se declaran tres componentes: 'ideal', 'sampling' y 'quantized' de los tipos 'SensorBenchmark', 'SamplingCase' y 'QuantizedCase' respectivamente. A los dos últimos componentes se les pasan parámetros en tiempo de instanciación para fijar el periodo de muestreo a 15 ms, en lugar de los 10 ms definidos por las clases a las que pertenecen. En las Figuras 2 y 3 se muestran los resultados de la simulación.

Se puede observar cómo se van degradando las prestaciones del sistema de control de velocidad conforme se añaden el muestreador, retenedor y convertidor A/D. El comportamiento al principio es muy oscilatorio debido a que inicialmente el sistema está en reposo (w=0), la referencia parte con un valor de 50, y el periodo de muestreo es excesivamente alto. Disminuyendo el periodo de muestreo y aumentando el número de bits del convertidor A/D el comportamiento se aproxima más al del sistema de control basado en un sensor ideal.

3.6 SIMULACIÓN CON ECOSIMPRO/EL

El código EL necesario para realizar la simulación es:

```
USE MECH_ROTATIONAL
USE TILLER_C04
COMPONENT SamplingCase
TOPOLOGY
  MECH_ROTATIONAL.RotationalInertia inertia (J=0.8)
  MECH_ROTATIONAL.RotationalFixed ground
  MECH_ROTATIONAL.RotationalDamper damper (d=0.2)
  MECH_ROTATIONAL.RotationalTorque actuator
  SampleHoldSensor sensor(sample_interval = 0.015)
  TILLER_C04.PIController controller (Kp = 100, Ti = 0.1)
  CONNECT ground.p TO damper.flange_b
  CONNECT damper.flange_a TO inertia.flange_b
  CONNECT actuator.flange_b TO inertia.flange_a
  CONNECT sensor.flange_a TO inertia.flange_a
  CONNECT controller.driver TO actuator.inPort
  CONNECT sensor.outPort TO controller.sensor
END COMPONENT

COMPONENT QuantizedCase
TOPOLOGY
  MECH_ROTATIONAL.RotationalInertia inertia (J=0.8)
  MECH_ROTATIONAL.RotationalFixed ground
  MECH_ROTATIONAL.RotationalDamper damper (d=0.2)
  MECH_ROTATIONAL.RotationalTorque actuator
  QuantizedSensor sensor(sample_interval = 0.015, bits=8)
  TILLER_C04.PIController controller (Kp = 100, Ti = 0.1)
  CONNECT ground.p TO damper.flange_b
  CONNECT damper.flange_a TO inertia.flange_b
  CONNECT actuator.flange_b TO inertia.flange_a
  CONNECT sensor.flange_a TO inertia.flange_a
  CONNECT controller.driver TO actuator.inPort
  CONNECT sensor.outPort TO controller.sensor
END COMPONENT

COMPONENT Comparacion
TOPOLOGY
        SensorBenchmark ideal
        SamplingCase sampling
        QuantizedCase quantized
```

```
END COMPONENT
```

La librería TILLER_C04 ha sido desarrollada por EA International. En este caso, debido a que no existen componentes sustituibles en EL, es necesario definir explicitamente las clases 'SamplingCase' y 'QuantizedCase'. En cada una de ellas se incluye el tipo de sensor que interesa.

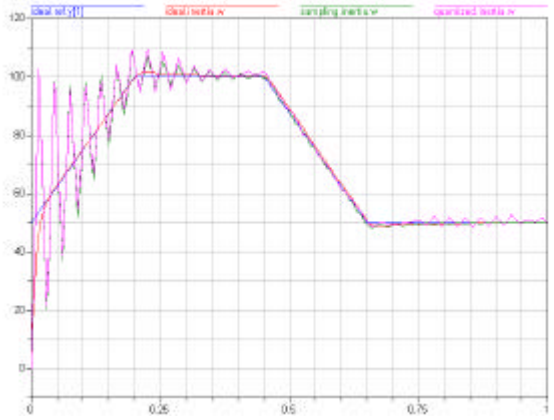

Figura 2: Simulación de los tres sistemas

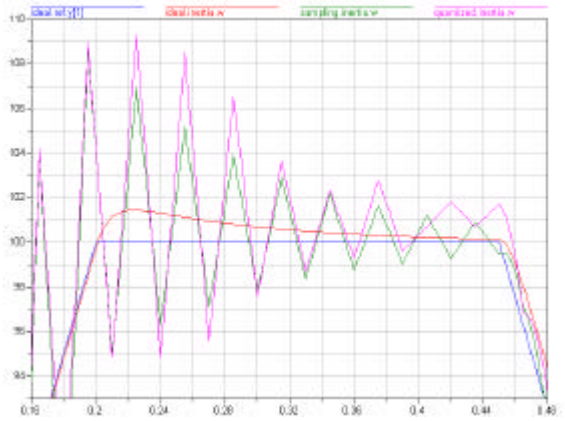

Figura 3: Detalle de la simulación

Después de generar la partición por defecto desde EcosimPro, se crea el siguiente experimento:

```
EXPERIMENT exp1 ON  Comparacion.default

DECLS
   TABLE_1D trapezoid = {{0., 0.2, 0.45, 0.65, 1},{0., 1., 1.,
0., 0.}}
INIT    -- set initial values for variables
    -- Dynamic variables
   ideal.inertia.phi = 0
   ideal.inertia.w  = 0
   ideal.controller.IntegratorBlock.s_out.signal = 0
   sampling.inertia.phi = 0
   sampling.inertia.w  = 0
   sampling.controller.IntegratorBlock.s_out.signal = 0
   quantized.inertia.phi = 0
   quantized.inertia.w  = 0
   quantized.controller.IntegratorBlock.s_out.signal = 0

BOUNDS    -- set expressions for boundary variables: v =
f(t,...)
   ideal.controller.command.signal = 50 + 50 *
periodTimeTableInterp(TIME, trapezoid, 1., 0)
   quantized.controller.command.signal = 50 + 50 *
periodTimeTableInterp(TIME, trapezoid, 1., 0)
   sampling.controller.command.signal = 50 + 50 *
periodTimeTableInterp(TIME, trapezoid, 1., 0)

BODY
   REPORT_TABLE("reportAll", " * ")
   TIME = 0
   TSTOP = 1
   CINT = 0.002
   INTEG()
END EXPERIMENT
```

Con este experimento se pretenden definir las excitaciones al sistema e inicializar el sistema en reposo inicialmente . Los resultados de la simulación obtenidos mediante EcoMonitor se muestran en las Figuras 4 y 5.

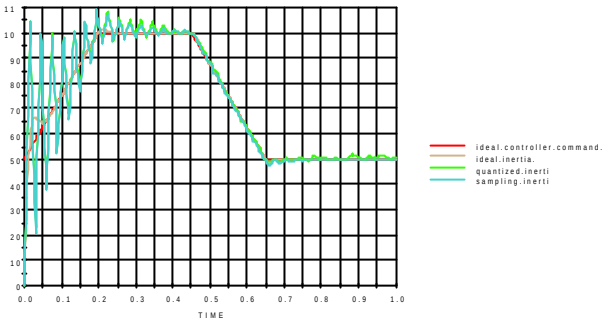

Figura 4: Resultados de simulación

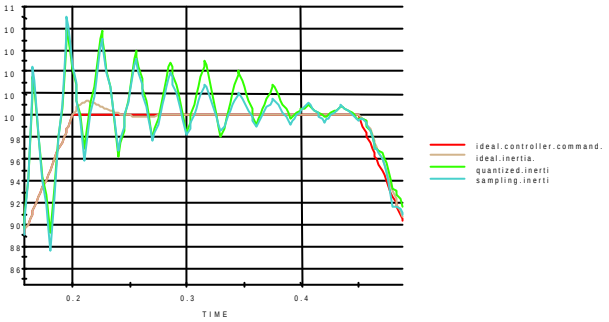

Figura 5: Detalles de la simulación

Se puede observar que los resultados de la simulación mostrados en las Figuras 4 y 5 no son iguales a los de las Figuras 2 y 3. Esto se debe a que en la implementación de sistema de control en cada uno de los lenguajes se ha utilizado un controlador PI con distintas realizaciones internas. En el caso de Modelica se ha utilizado el controlador PI de la MSL y en el caso de EL se ha utilizado el diseñado en el capítulo 4 de [7]. En ambos casos se aprecia claramente la influencia del muestreo y de los convertidores A/D utilizados.

4 CONCLUSIONES

En general, ambos lenguajes están perfectamente dotados de construcciones que permiten abordar la metodología de modelado orientada a objeto de sistemas continuos, discretos e híbridos.
La opinión de los autores respecto a posibles extensiones en ambos lenguajes son:

1 EL aumentaría sus posibilidades de modelado si los arrays pudiesen contener componentes, además de tipos simples y derivados en EL.
2 EL aumentaría sus posibilidades de Modelado con la introducción de clases paramétricas.
3 Modelica aumentaría su flexibilidad en el modelado de componentes de un mismo grafo de herencia, si permitiera la utilización de ecuaciones virtuales que especializaran el comportamiento en sucesivas generaciones de componentes.
4 El interfaz para simular experimentos de modelos con EcosimPro mediante lenguajes de programación (C++) es una capacidad muy flexible no ofrecida por Dymola/Modelica.
5 La utilización de herramientas gráficas complementarias por parte de EcosimPro/EL, como SmartSketch, hace más incómodo la labor de modelado que si dicha funcionalidad estuviera incorporada en la misma herramienta. Debido a ésto, modelar gráficamente en Dymola/Modelica es mucho más cómodo que en EcosimPro.

Agradecimientos

Los autores agradecen a la CICYT y a los fondos FEDER la financiación en parte la investigación realizada en el ámbito de los proyectos QUI99-0663-C02-02, DPI2000-1218-C04-01, DPI2001-2380-C02-02 y DPI2002-04375-C03-02/03.

Referencias

[1] Modelica Association, (2002): "Modelica Language Specification, Version 2.0". http://www.modelica.org.

[2] EA International. EcosimPro 3.2. Documentación.

[3] Dynasim AB. Dymola 5.0a. Documentación.

[4] Andersson, M. (1994). "Object-Oriented Modeling and Simulation of Hybrid Systems". PhD thesis. ISRN LUTFD2/TFRT—1043—SE, Departament of Automatic Control, Lund Institute of Technology, Lund, Sweden.

[5] Modelica Association, (2002): "Modelica Tutorial, Version 1.4".http://www.modelica.org.

[6] Stroustrup, B., (1997): "The C++ programming language. Third Edition". Addison-Wesley.

[7] Tiller, M.M. (2001): "Introduction to Physical Modeling with Modelica". Kluwer Academic Publishers

[8] Urquía Moraleda, A. (2000). "Modelado Orientado a Objetos y Simulación de Sistemas Híbridos en el ámbito del Control de Procesos Químicos". Tesis doctoral. Depto. De Informática y Automática. Facultad de Ciencias. U.N.E.D.