EcosimPro PROOSIS
Modelling and Simulation Software
EcosimPro/PROOSIS · Newsletter Nº 13 · Febrero 2017

EDITORIAL

A finales de 2016 sacamos al mercado las nuevas versiones EcosimPro 5.6 y PROOSIS 3.8 que incluyen nuevas capacidades muy potentes que tendrán sin duda un gran impacto en los trabajos de los usuarios.

Se ha creado una herramienta de testeo automático de librerías y modelos que puede revolucionar el uso de nuestras herramientas en un entorno industrial. El trabajo de los usuarios para validar cambios en sus modelos podía llevarles dias o semanas de trabajo. Ahora se puede automatizar y dejar una noche pasando cientos de tests automáticos y al dia siguiente tener la comparación exacta con los resultados de referencia.

Esta nueva versión incluye muchas novedades como por ejemplo una herramienta para simular directamente desde el esquemático, de esta manera, el usuario puede visualizar los resultados de la simulación sobre el mismo esquemático. Otra es la exportación de los modelos como caja negra usando los estándares internacionales FMI y ARP4868 que permite su uso desde otras herramientas. El lenguaje de modelado ha sido ampliado con clases template, punteros a funciones, etc. También tenemos nuevos resolvedores de transitorios más potentes que aceleran significativamente la simulación y mejoran la convergencia.

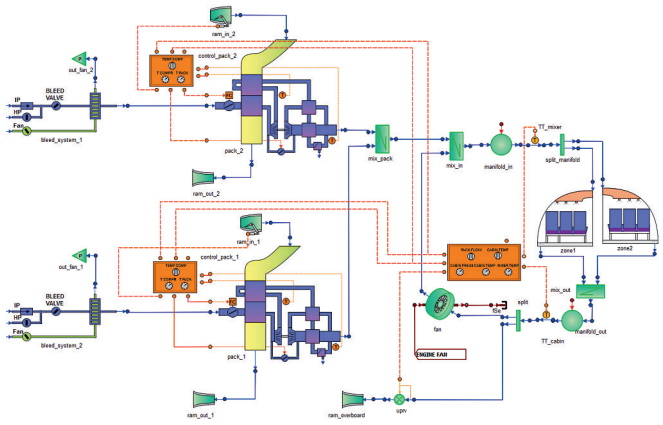
Es de destacar la cantidad de trabajos presentados por nuestros usuarios en las conferencias de Propulsión Espacial en Roma y Propulsión Aeronaútica en ASME Turbo en Corea.

La web se ha actualizado con más de 25 trabajos presentados con EcosimPro/ESPSS y PROOSIS/TURBO.

Un ejemplo interesante es el de la empresa india Team Indus preseleccionada por Google Lunar XPrize para enviar una nave espacial a la luna en 2017, posarla y enviar imágenes de alta precisión a la tierra. Team Indus usa para ello EcosimPro/FluidaPro.

Por delante tenemos nuevos retos como el de permitir la interconexión los modelos con otras herramientas y estándares usados en la industria, la simulación en tiempo real y otros que nos hacen mantener un espíritu continuo de creatividad y atención a las necesidades de los usuarios.

Pedro Cobas (pce@ecosimpro.com)
Jefe del Equipo de Desarrollo EcosimPro/PROOSIS
EA Internacional



ECS Model in PROOSIS

ÍNDICE

1. SIMULADOR DEL SISTEMA CRIOGÉNICO ITER

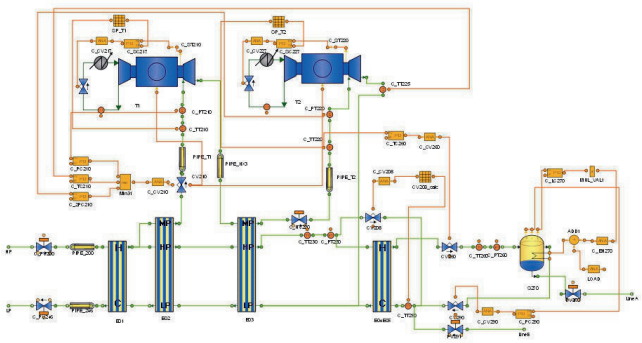
ANA VELEIRO, ECOSIMPRO/PROOSIS

Iter Organization (IO) va a desarrollar un simulador integrado de los distintos sistemas que conforman el reactor experimental ITER en Cadarache (Francia). El simulador pretende conectar e integrar los simuladores individuales de los sistemas. El objetivo final del simulador integrado es apoyar el comisionado de ITER, la ingeniería de apoyo durante la operación y mantenimiento de ITER, y el entrenamiento de los operadores.

En Criogenia, ITER viene trabajando desde hace tiempo en el desarrollo de modelos que permitan verificar el diseño, así como desarrollar algoritmos de control avanzado y probar el control con simulaciones hardware in the loop. Para ello EAI ha desarrollado modelos dinámicos de los circuitos que refrigeran los imanes de ITER y modelos iniciales para las bombas criogénicas y su distribución.

Dada la complejidad del sistema, IO ha decidido crear una plataforma de simulación distribuida que pueda integrar modelos de los distintos subsistemas y simularlos conjuntamente. EAI desarrollará ciertas funcionalidades específicas que hagan posible esta integración.

EAI dotará a EcosimPro de la capacidad de generar servidores OPC UA desde la herramienta. Esto permitirá conectar modelos desarrollados en EcosimPro con otras herramientas que tengan una interfaz OPC UA y crear una plataforma de simulación distribuida en base a esta tecnología. También se desarrollará un mecanismo de sincronización entre los distintos elementos y se trabajará en la optimización de los modelos para este tipo de aplicaciones.



2. LIBRERÍA EDUCACIONAL DE PROPULSIÓN ESPACIAL (LPRES)

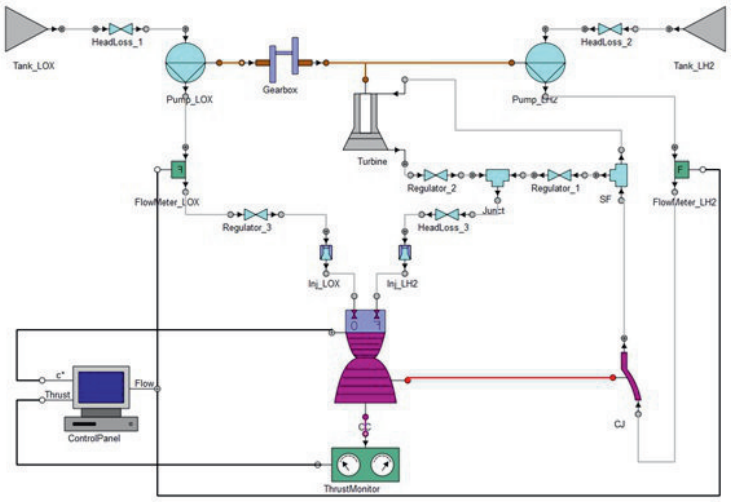
PABLO SIERRA, ECOSIMPRO/PROOSIS

Los motores cohete de propulsante líquido forman parte de los contenidos impartidos en ingenierías relacionadas con el sector aeroespacial. Las herramientas de cálculo que se emplean son muy sofisticadas, requieren una gran cantidad información de entrada y un usuario experto que las utilice, pero no son adecuadas para su uso en el aula. Por ello, EAI junto con la Universidad Politécnica de Madrid (UPM), han desarrollado la librería LPRES (Liquid Propellant Rocket Engine Simulation) en EcosimPro, que es una librería simplificada para la simulación de motores cohete de propulsante líquido.

LPRES trata de resolver las dificultades típicas en la enseñanza siendo una librería que se puede emplear en el aula. Es decir, con una curva de aprendizaje corta y que emplea modelos y conceptos similares a los que el alumno maneja en la resolución con “lápiz y papel” de los problemas habituales de clase.

LPRES emula los elementos de las librerías profesionales ESPSS, y contiene componentes para predecir el comportamiento estacionario de las diferentes configuraciones que puede presentar un motor cohete de propulsante líquido. Las simplificaciones principales utilizadas en LPRES son: gases y líquidos perfectos, cambio de fase limitado a algunos componentes, y modelos analíticos.

Además, varios ejemplos que utilizan la librería LPRE se incluyen en la librería LPRES_EXAMPLES. Estos ejemplos facilitan mucho el proceso de aprendizaje del usuario. Así se ha realizado el ciclo de un generador de gas, un ciclo expansor, un motor cohete presurizado e incluso un aerorreactor. Para validar los resultados se han efectuado comparaciones de algunos de los ejemplos con los resultados obtenidos mediante las librerías profesionales ESPSS y con valores obtenidos de bibliografía de motores cohete reales.



Modelo realizado con la librería LPRES. Simulación del motor cohete RL10

A modo de conclusión, la librería LPRES contiene componentes simples que hacen que sea sencilla de usar porque trabaja con un número pequeño de datos. Por lo tanto, su curva de aprendizaje es muy corta y el usuario no tiene que ser un experto para usarla. Además, los resultados de LPRES se obtienen de forma rápida y se pueden comparar exitosamente con valores reales. Por todas estas razones, LPRES es una herramienta muy útil para propósitos educacionales.

Finalmente, LPRES tiene potencial de crecimiento en un futuro cercano, añadiendo, por ejemplo, un modelo de masas, la capacidad de realizar simulaciones transitorias para lo que EcosimPro es el software ideal, o añadiendo más detalle en la descripción de los fluidos.

3. TEST CASE DE LA LIBRERÍA SMART_GRID & RENEWABLES

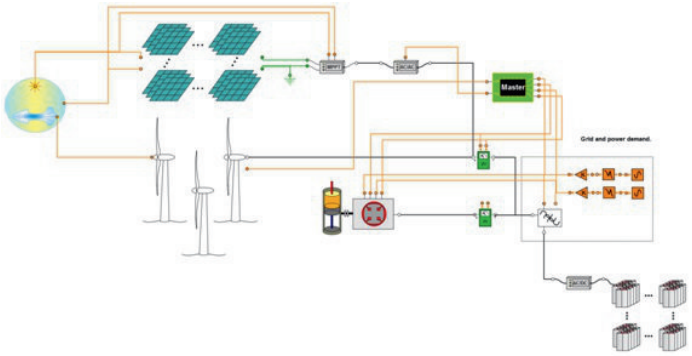
VÍCTOR PORDOMINGO, ECOSIMPRO/PROOSIS

Los sistemas de generación y transporte de energía se encuentran en un momento de profunda renovación, donde las redes inteligentes y la generación distribuida serán las grandes protagonistas del sector.

La nueva librería SMART_GRID & RENEWABLES de EcosimPro ofrece la capacidad de estudiar redes formadas por diversas plantas (actualmente: fotovoltaica, eólica y diesel y, en el futuro: hidroeléctrica, termo-solar, pilas de combustible,…) trabajando en isla o conectadas a red, para estudiar la producción diaria óptima en diversas condiciones meteorológicas, con diferentes perfiles de consumo y analizando distintas configuraciones del control local y central. Su capacidad puede ampliarse utilizando otras librerías disponibles en EcosimPro (ELECTRIC_SYSTEMS, CONTROL, MECHANICAL, THERMAL,…), así como los asistentes de cálculo en las fases de diseño y dimensionamiento (estudios paramétricos, optimización, simulación de Monte Carlo, …).

Se presenta aquí un primer test case de la librería (para más información ver paper correspondiente en nuestra WEB), que ha consistido en modelar un sistema en el que se gestionan tres fuentes de energía (solar, eólica y respaldo diesel) para alimentar una determinada demanda variable durante una jornada completa. La red cuenta, además, con un dispositivo de almacenamiento formado por un banco de baterías. El funcionamiento conjunto es gestionado por un controlador “máster” que distribuye la generación de la forma más eficiente posible para satisfacer la demanda en cada momento de acuerdo a los recursos de generación disponibles.

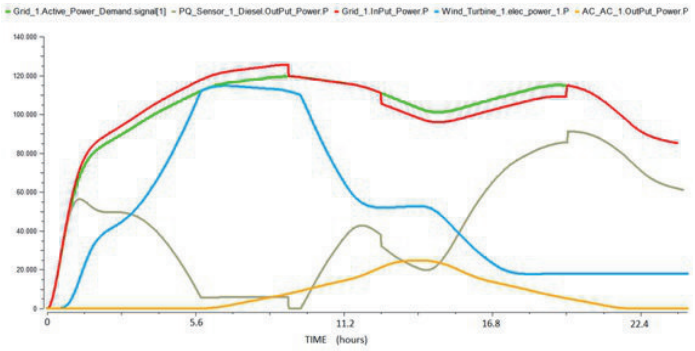
En el esquemático de la figura pueden observarse los bloques de generación solar fotovoltaica y eólica conectados a unas determinadas condiciones ambientales (radiación solar, temperatura y viento). En la parte inferior, completan el sistema el generador diesel y el banco de baterías. Finalmente, todos estos componentes se conectan a los controladores debiendo cubrir la demanda que el usuario también puede configurar mediante el ajuste de los bloques correspondientes.



Esquemático de la red analizada con EcosimPro

Las curvas muestran los resultados de la simulación para el balance diario de potencia activa. En verde puede verse la curva de demanda con dos horas punta. En rojo se muestra la generación total, que sigue a la demanda, y en la que se aprecian dos tramos en los que la generación está por encima

y por debajo de la demanda. Estos tramos corresponden a la carga y descarga de las baterías siguiendo el criterio establecido para los controladores. En azul se muestra la generación eólica y en naranja la solar fotovoltaica, que se emplean al máximo posible para reducir el consumo de combustible a lo estrictamente necesario. En gris se ve la generación diésel, muy variable ya que se emplea únicamente para cubrir la parte de la demanda que la generación renovable no puede alcanzar.



Balance diario de potencia activa del sistema

4. PROOSIS EN ASME TURBO, COREA

Alex Alexiou, Universidad Nacional de Atenas

Asia volvió a albergar la ASME TURBO EXPO (2016) en Seúl (Corea del Sur), que concentró a 3000 participantes tanto de la industria como del sector de la investigación gubernamental y académica de Turbinas de Gas de todo el mundo.

Se hizo gran hincapié en las amplias posibilidades de negocio y en las mejoras de fiabilidad y rendimiento presentes en esta industria, facilitadas por la confluencia de los mundos físicos y digitales. El modelado y la simulación juegan un papel crucial para aprovechar estas oportunidades y PROOSIS, como entorno de simulación puntero de turbinas de gas, está aportando una sorprendente contribución.

PROOSIS participó en esta conferencia a través de ocho publicaciones presentadas por la Universidad Nacional de Atenas (Grecia), la Universidad de Cranfield (Inglaterra), y la escuela ISAE-SUPAERO (Institut Supérieur de l'Aéronautique et de l'Espace) de Toulouse (Francia).

Concretamente, PROOSIS y su librería TURBO se utilizaron para modelar y simular motores aeronaúticos avanzados como el Open Rotor, considerado como futura planta de potencia para aviones de nueva generación (GT2016-56645, GT2016-57918 y GT2016-57921). PROOSIS se utilizó para desarrollar modelos de componentes de este novedoso motor, tales como propulsores y turbinas contra-rotativos y para realizar estudios de actuaciones una vez integrados en los motores. El abanico de posibilidades de simulación de PROOSIS se muestra en GT2016-56617 con la integración de un código in-house bidimensional de un fan en un modelo completo de propulsor turbofan con PROOSIS.

PROOSIS se utilizó también para los modelos de las turbinas de gas en una Planta de potencia combinada para análisis de performances y 'health-monitoring' (GT2016-57722). La capacidad de modelar sistemas transitorios con TURBO se describe en GT2016-57257, donde las predicciones del modelo digital se compararon satisfactoriamente con medidas reales del motor. Finalmente, la multi-disciplinariedad de PROOSIS se demuestra en GT2016-57700 y GT2016-57272 donde se analizaron modelos híbridos turbo-solares.

5. ECOSIMPRO/ESPSS EN LA CONFERENCIA DE PROPULSIÓN ESPACIAL

La European Space Propulsión Conference se celebró en Roma (2-6 Mayo) organizada por la Agencia Espacial Europea (ESA) y otros organismos europeos del sector.

Nuestros ingenieros de simulación participaron presentando varias ponencias así como atendiendo al Stand, donde se mostraron los distintos productos que usan la plataforma de simulación con EcosimPro, ESPSS (European Space Propulsion Simulation System), FluidaPro, Pipeliqtran, etc.

EcosimPro/ESPSS ha sido desarrollado por un consorcio europeo liderado por EAI y es la herramienta de la ESA para el modelado 0D-1D de sistemas de propulsión espacial. ESPSS permite el modelado de sistemas de propulsión de lanzadores y satélites, incluyendo los procesos de arranque, evolución de tanques de combustible, intercambiadores de calor, sistemas mecánicos o reguladores de presión electrónicos.

Entre las novedades presentadas en este foro cabe destacar la simulación acoplada del sistema de propulsión y la dinámica de un vehículo espacial, la inclusión de nuevos esquemas numéricos de gran precisión y muy eficientes en términos de velocidad de simulación, y la inclusión de un módulo para la definición de geometrías avanzadas de combustible sólido, y la validación de los combustores sólidos e híbridos frente a resultados experimentales.

EcosimPro/ESPSS se usa por las principales empresas aeroespaciales europeas para el diseño de nuevos sistemas de propulsión espacial, como se muestra en la gran cantidad de ponencias presentadas que se incluyen en la WEB tales como las aplicaciones para Exomars, la nave Orion, vehículos espaciales reusables de alta velocidad, etc. EAI presentó "ESPSS Model of a Simplified Combined-Cycle engine for supersonic cruise".

ESA y EAI organizaron en la misma feria el quinto workshop de usuarios EcosimPro/ESPSS con asistencia de más de 40 ingenieros de la industria. Se presentaron las últimas mejoras y los desarrollos en curso, y hubo una mesa redonda con propuestas de los usuarios respecto a nuevas mejoras en la herramienta.

6. TEAM INDUS USA ECOSIMPRO / FLUIDAPRO PARA EL GOOGLE LUNAR XPRIZE

Google Lunar XPRIZE es una competición que busca desarrollar un acceso eficiente y económico a la Luna promoviendo la innovación tecnológica de emprendedores de todo el mundo. El objetivo de la competición es aterrizar un vehículo en la superficie lunar, desplazarlo al menos 500 metros y transmitir imágenes y video en alta definición de vuelta a la Tierra. La competición ha entrado ya en su última fase, donde han sido preseleccionados 5 equipos que deben preparar su lanzamiento en 2017.

Uno de los equipos es Team Indus (Bangalore, India) que es una pequeña compañia india que está desarrollando un proyecto para ir a la luna y transmitir las imágenes de video. Team Indus hace uso de nuestro producto EcosimPro/FluidaPro para realizar los análisis fluidodinámicos del sistema de propulsión.

Desde EcosimPro nos alegramos de que sea uno de los equipos preseleccionados por Google para este importante premio y les deseamos suerte en la fase final del concurso.

Para más información ir a:
http://lunar.xprize.org/news/blog/meet-5-teams-are-launching-moon-year

7. HERRAMIENTA DE VALIDACIÓN DE LIBRERÍAS

Fernando Carbonero, EcosimPro/PROOSIS

Es una herramienta que permite la automatización de la validación de librerías y modelos, lo que puede marcar un antes y un después a los usuarios. La herramienta puede ahorrar dias de trabajo ya que muchas cosas que se hacían manualmente hasta ahora se automatizan y se pueden dejar ejecutándose una noche y a la mañana siguiente ver los resultados.

Al modificar componentes en EcosimPro siempre surge la misma necesidad: saber si con los cambios realizados, los resultados siguen siendo válidos. Esta tarea puede ser demasiado costosa, ya que cada modelo requiere varios pasos y si hubiera que repetirlos sobre un número grande de modelos el proceso global llegaría a ser inabordable.

EcosimPro 5.6 proporciona una nueva herramienta para realizar gran parte de los procesos de forma automatizada. Con unos clicks de ratón, sólo habrá que esperar un poco y obtener los resultados para analizarlos.

Se comprobará primero si la fase de modelado es aún válida. En primer lugar se comprobará el código de todas las librerías del workspace. Así podrá verse si algún cambio en el código fuente de un componente afecta a otros que heredan de él, o lo usan en su topología: cambios de nombres o supresión de alguna variable,etc.

También se verá si las particiones que se tenían siguen siendo válidas o si hay que editar algunas de ellas. Y finalmente se comprobará si los experimentos siguen compilando con dichas modificaciones.

En la fase de simulación el tiempo de ingeniero ahorrado puede ser bastante apreciable. Con decenas o cientos de modelos que correr, el usuario tendría que estar horas delante del ordenador hasta terminar la tarea a mano. Con esta herramienta se puede lanzarla y esperar a que todos los modelos hayan terminado.

Si algún modelo no ha dado los resultados esperados se puede acceder de una forma rápida y cómoda a una comparación de los resultados respecto a las referencias.

En el informe se tiene información de cuantos cálculos se han realizado (transitorios, estacionarios) y sus resultados. Igualmente, si existen variables que ya no están en el modelo de aquellas seleccionadas para comparar.

Se tienen dos modos de comparación. En primer lugar se tiene un modo estadístico, en el que se almacenan los valores estadísticos de las variables seleccionadas para la comparación. Este modo es especialmente útil cuando el número de puntos resultantes por variable es elevado, por ejemplo en un transitorio.

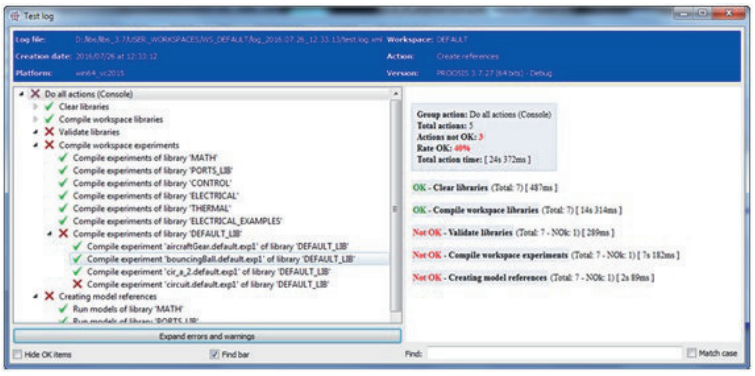
También puede irse más allá con una comparación exhaustiva de cada valor obtenido por variable. Por ejemplo, en un estudio paramétrico a partir de cálculos estacionarios.

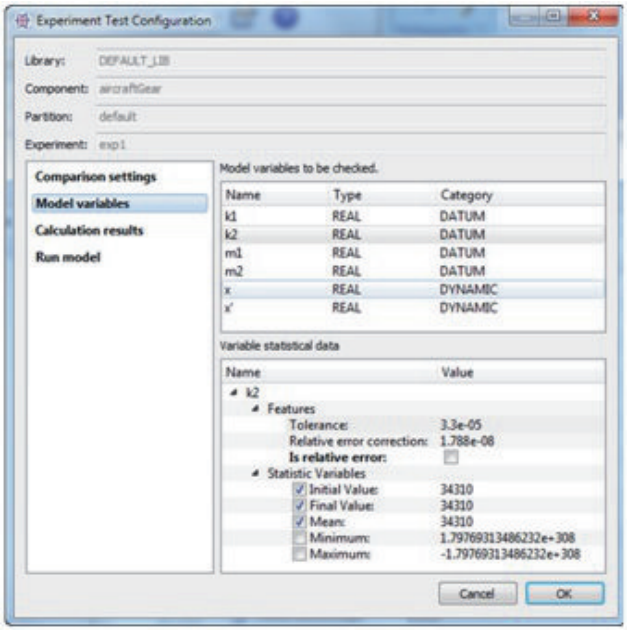
¿Y, contra qué referencias se compara? Este es uno de los procesos más sencillos. Partiendo de una librería en la que se tiene confianza. Es decir, se sabe que funciona, y que funciona bien.

Se seleccionan los modelos a validar, se configuran los parámetros necesarios y se deja que el programa obtenga todos los resultados que servirán de referencia tanto a nivel de modelado como de simulación.

Existe la posibilidad de realizar una configuración de los parámetros del workspace, incluso ir al detalle configurando un modelo individual hasta llegar a configurar una variable por variable si fuera necesario.

En resumen, se tiene una herramienta con la que validar los modelos o un subconjunto de ellos de una forma rápida y efectiva. Y se efectúa este proceso con sólo unos clicks tantas veces como se necesite ahorrando cientos de horas de ingeniero.





8. NUEVOS RESOLVEDORES TRANSITORIOS

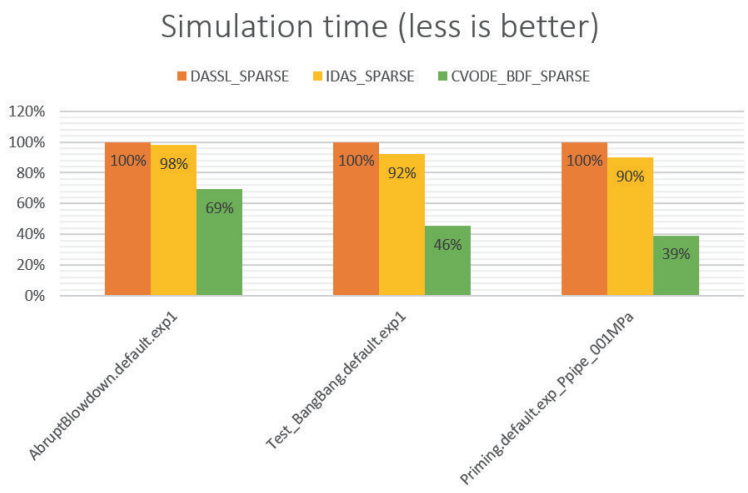
FERNANDO PUECH, ECOSIMPRO/PROOSIS

EcosimPro 5.6.0 incluye los nuevos resolvedores transitorios IDAS, IDAS_SPARSE, CVODE_AM, CVODE_BDF y CVODE_BDF_SPARSE. IDAS y CVODE pertenecen al paquete de resolvedores SUNDIALS (SUite of Nonlinear and DIfferential/ALgebraic equation Solvers) desarrollado por el "Center for Applied Scientific Computing Lawrence Livermore National Laboratory". IDAS es un resolvedor de propósito general para el problema del valor inicial de sistemas de ecuaciones algebraico diferenciales, también conocidos como DAEs.

IDAS está basado en el resolvedor DASPK y el método de integración que utiliza es un BDF de orden variable y de coeficientes variables. CVODE es un resolvedor de problemas de valor inicial rígidos (stiff) y no rígidos (nonstiff) de sistemas de ecuaciones diferenciales ordinarias basado en los resolvedores VODE y VODPK. En EcosimPro se encuentra disponible CVODE_AM, ideal para problemas no rígidos, y cuyo método de integración es de orden variable y paso variable basado en las fórmulas de Adams-Moulton mientras que CVODE_BDF, ideal para problemas rígidos, utiliza un método de orden variable y paso variable basado en BDF (Backward Differentiation Formulas).

EcosimPro incluye además las versiones dispersas (sparse) de los resolvedores anteriores y que suponen una mejora sustancial con respecto al resolvedor DASSL_SPARSE. Primero porque utilizan un método multi-hilo en las iteraciones internas y segundo porque utilizan una versión optimizada de la obtención del jacobiano disperso.

Las mejoras mencionadas significan un menor tiempo de simulación, con todos los beneficios que ello supone para el usuario. En el siguiente gráfico se comparan los tiempos de simulación utilizando EcosimPro 5.6.0 y la plataforma de compilación win64_vc2015 de diferentes modelos de la ESPSS 3.1.0. En los experimentos el error relativo (REL_ERROR) y el error absoluto (ABS_ERROR) se han establecido a 1.0×1E-6.



Los datos aportados en este gráfico demuestran que los nuevos resolvedores son una mejora muy sustancial respecto a la anterior generación (especialmente CVODES) y os animamos a probarlos.

9. EXPORTACIÓN DE MODELOS CON INTERFAZ FMI 2.0 PARA CO-SIMULACIÓN

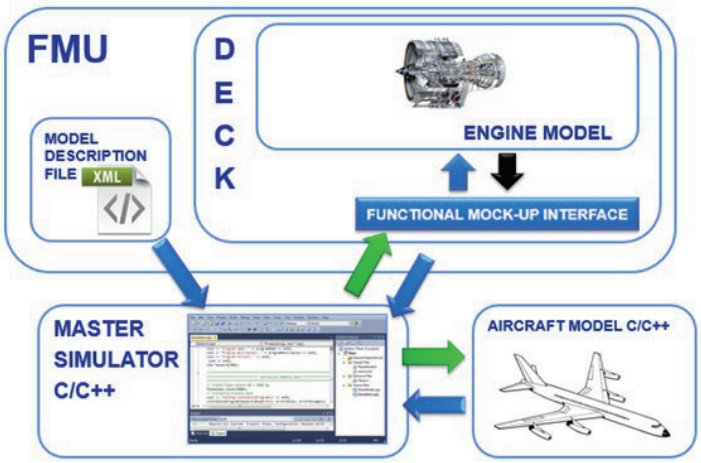
Fernando Puech, EcosimPro/PROOSIS

EcosimPro 5.6.0 introduce la capacidad de exportar un modelo utilizando el estándar FMI 2.0 para co-simulación (FMI 2.0 de ahora en adelante). FMI 2.0 es un estándar que busca facilitar la comunicación entre modelos de simulación desarrollados en diferentes herramientas y que incluye los siguientes conceptos básicos:

- FMI son las siglas de Functional Mock-up Interface, que es una interfaz ANSI C bien definida y se puede utilizar por un entorno de simulación para cargar uno o más modelos producidos por otras herramientas de simulación.
- FMU son las siglas de Functional Mock-up Unit, y es un fichero comprimido que contiene a su vez un fichero de descripción del modelo en formato XML y los ejecutables que implementan las funciones definidas en el FMI. Puede ser utilizado como un modelo esclavo en un entorno de simulación
- Existen dos tipos de FMUs: el FMI para co-simulación, en el que el modelo tiene su propio resolvedor/integrador (Este tipo de FMUs son los que EcosimPro genera), y FMI para intercambio de modelos, en el que el modelo espera que el simulador lleve a cabo la integración numérica.

Para exportar los modelos de EcosimPro utilizando FMI 2.0 como interfaz hay que preparar un experimento sencillo con unas reglas que se explican en la documentación y luego seleccionar en un wizard la opción de exportación a FMI.

Una vez que se tiene el experimento, y se ha ejecutado al menos una vez, se podrá lanzar el asistente para crear DECKs haciendo botón derecho en el experimento. El asistente, en su pantalla inicial permite habilitar la interfaz FMI 2.0. El siguiente paso es elegir las variables de entrada y salida que se quieren publicar y, por último, pulsar el botón de generación. En definitiva, con unos sencillos pasos el asistente habrá generado el fichero model_xxx.fmu que a su vez se puede cargar en cualquier herramienta que sea capaz de entender FMI 2.0 para co-simulación.



Lo mejor de esta capacidad es que cuando se tiene que compartir modelos con un cliente que no dispone de EcosimPro no hay que traducir el modelo a un lenguaje ni herramienta diferente, lo que supone un ahorro en costes y tiempos de desarrollo de modelos muy importantes.

10. NUEVO OPERADOR DE ASIGNACIÓN CASUAL

Hasta ahora EcosimPro/PROOSIS permitía introducir sólo ecuaciones acausales en el bloque CONTINUOUS de componentes y puertos. A partir de esta nueva versión también se permite escribir ecuaciones causales que imponen una manera de calcular una variable.

Ahora cuando se escriben ecuaciones en el bloque CONTINUOUS de un componente o puerto se puede usar indistintamente los operadores "=" y ":=". El operador "=" se usará siempre que se quiera expresar la ecuación en forma totalmente acausal (como se ha hecho hasta ahora), esto significa que EcosimPro/PROOSIS podrá transformar esta ecuación simbólicamente como estime conveniente. Así, por ejemplo dada la ecuación:

```
x= sin(y)
```

El algoritmo de ordenación podría finalmente escribir esta ecuación, dependiendo de la desconocida que se quiera calcular, en una de estas dos formas:

```
1  x=sin(y)
2  y=asin(x)
```

Lo cual está muy bien y da una gran flexibilidad a la herramienta para que el mismo modelado pueda ser usado en escenarios muy diferentes. Sin embargo, hay veces que el modelador no quiere que se transforme simbólicamente una ecuación y se mantenga el formato original, y para ellos se ha introducido el operador ":="

```
x := sin(y)
```

Se le está indicando al algoritmo de ordenación que no transforme simbólicamente la ecuación y mantenga las partes izquierda y derecha intactas. En este caso al final de la ordenación de ecuaciones esta ecuación o bien calcula x de forma explícita o bien se convierte en un residuo:

```
1    x=sin(y)
2    residue= (x) - (sin(y))
```

Este mecanismo evita en estos casos una transformación simbólica que puede ser peligrosa para la convergencia o simplemente se quiere que esta ecuación calcule siempre esta variable. Se está imponiendo por tanto causalidad a la ecuación.

Otra diferencia entre el operador "=" y el ":=", el primero admite tanto "expression= expression" como "variable= expresión", sin embargo el segundo sólo admite "variable= expression".

Así, por ejemplo esto es correcto:

```
x + y = 24*z
```

pero en cambio no es correcto escribir:

```
x + y = 24*z      -- Compilation error
```

Puesto que las ecuaciones causales deben usarse siempre para calcular una variable concreta.

También el operador ":=" es una alternativa más flexible al prefijo EXPL que busca una ecuación para calcular una variable. Por ejemplo si se tiene un componente como:

```
COMPONENT test
DECLS
      EXPL REAL x
      REAL y
CONTINUOUS
      x= 34.5*y + cos(TIME)
END COMPONENT
```

Hasta ahora este mecanismo era el que se usaba para buscar una ecuación explícita para calcular x. El problema venía si encontraba dos ecuaciones que calculen x, por ejemplo:

```
COMPONENT test
DECLS
      EXPL REAL x
      REAL y
CONTINUOUS
      x= 34.5*y + cos(TIME)
       x= 5.3*y
END COMPONENT
```

En este caso daba el siguiente error al hacer la partición:

```
*** Error 1 (code 925 ESI:99:925:01:98) ***
x = 5.3 * y
Variable "x" is already calculated in:
x = 34.5 * y + cos(TIME)
The equations system can be redundant.
```

Sería mejor hacerlo con el operador ":=":

```
COMPONENT test
DECLS
   REAL x
   REAL y
CONTINUOUS
   x := 34.5*y + cos(TIME)
   x= 5.3*y
END COMPONENT
```

En este caso fuerza a calcular x con la primera ecuación. Por tanto a partir de ahora no se recomienda usar el prefijo EXPL y usar este nuevo operador de asignación que es más intuitivo y robusto.

11. USO DE CLASES TEMPLATES EN EL

EcosimPro 5.6 permite definir nuevos tipos de datos con la sentencia TYPEDEF para ser usados posteriormente en la programación. Concretamente se permite definir nuevas clases heredadas de otras clases y definición de funciones.

Cuando se usan clases template en EL la sentencia TYPEDEF puede usarse para definir un nuevo tipo de clase heredado de la clase template aplicado a una clase concreta. Por ejemplo se puede usar la clase template EVector<className> para crear un vector de objectos del tipo "className" definiendo:

```
TYPEDEF CLASS newClassName IS_A
EVector<className>
```

La nueva clase "newClassName" puede ser usada a partir de ahora como cualquier otra clase y representará un vector de objetos de la clase "className". Puede ponerse un ejemplo con una clase sencilla llamada "Employee" con dos atributos "name" y "phone", y luego se crea un vector de objetos de este tipo:

```
CLASS Employee
DECLS
    STRING name
    INTEGER phone
END CLASS

TYPEDEF CLASS VectorEmployees IS_A
EVector<Employee>
```

A partir de este momento se puede usar la clase "VectorEmployees" como una más. Por ejemplo se puede añadir algún empleado tal como se vio cuando se describió la clase EVector:

```
OBJECTS
    VectorEmployees v
    Employee emp
BODY
    emp.name= "Jeff"
    emp.phone= 36474778
    v.append(emp)
```

Se ve, por tanto, cómo se ha definido una clase nueva heredada de una clase contenedora que puede contener objetos complejos. A partir de aquí se podrían incluso hacer nuevas clases template basadas en otras por ejemplo,

```
TYPEDEF CLASS dictionaryEmployees IS_A
EDictionaryString<VectorEmployees>
```

Ahora se tiene una nueva clase que puede usarse en nuestro código EL. Por ejemplo:

```
OBJECTS
    VectorEmployees v1, v2
    dictionaryEmployees dict
    Employee emp
BODY
    emp.name= "Jeff"
    emp.phone= 36474778
    v1.append(emp)
    emp.name= "Bill"
    emp.phone= 56363326
    v2.append(emp)
    dict.set("plant",v1)
    dict.set("office",v2)
```

Se ve cómo se puede ir sofisticando la creación de nuevas clases usando los templates y las clases contenedoras. Esto aporta un valor añadido muy grande a EL pues permite construir nuevos tipos de datos que representan estructuras complejas en memoria.

12. NUEVAS CLASES CONTENEDORAS

Las nuevas clases contenedoras son en general una alternativa al uso de arrays en EL. Actualmente un usuario puede usar un array para almacenar un vector de datos (e.g. REAL v[4]) o una matriz (e.g. REAL v[4,4]), pero tiene la limitación de que no pueden dimensionarse dinámicamente. Con estos contenedores se le dan al usuario unos contenedores más avanzados que permiten manejo de memoria automático a cambio de tener unas sintaxis menos intuitiva pues siempre se accede a ellos a través de métodos de clase (eg. v[3] vs v.at(3)). Será el usuario el que determine si es más conveniente usar un array o un contenedor de este tipo para una aplicación concreta.

Básicamente se han creado dos tipos de clases contenedoras:

- Contenedores secuenciales: Clases que almacenan la información de forma contigua en memoria. Son las clases EVector y EMatrix
- Contenedores asociativos: Clases que no almacenan la información de forma contigua en memoria y que siempre son ordenadas por alguna clave. Son las clases EDictionary y ESet.

Estas clases han sido diseñadas persiguiendo 3 objetivos:

- Simplicidad de uso
- Manejo inteligente de memoria
- Que sean lo más eficientes posibles en términos computacionales

Los nuevos contenedores están basados en la librería estándar STL de C++, con lo cual se benefician de los últimos avances en los compiladores de C++ y con millones de usuarios a nivel mundial que los usan en miles de proyectos de forma eficiente. No tienen exactamente la misma interfaz pues se han tratado de simplificar mucho para los usuarios de EL.

Todas las clases contenedoras puede contener tanto los tipos básicos de EL (REAL, INTEGER, STRING, BOOLEAN,ENUM) como objetos de cualquier clase creada en EL. Esto es importante pues pueden crearser estructuras de datos complejas de una manera eficiente.

Los vectores EVector (basada en la std::vector en C++) son contenedores secuenciales similares a los arrays unidimensionales de EL (eg REAL v[3]), sólo que tienen más inteligencia para manejar la memoria. Todos los objetos se almacenan contiguos en memoria. EVector maneja sus elementos de forma dinámica y permite acceso directo a cada elemento basado en un índice. Añadir elementos al final o borrarlos es muy rápido, sin embargo insertar objetos entre dos elementos no es óptimo puesto que internamente tiene que mover todos los objetos para hacer hueco a los nuevos. Un vector no requiere definir el tamaño inicial, cuando se inserta un elemento en una posición internamente se autodimensiona para que tenga al menos esos elementos, por ejemplo:

```
DECLS
  REAL value
OBJECTS
   EVectorReal v
BODY
   v.set(25, 3.1415)
   value= v.at(25)
```

En este ejemplo se introduce el valor 3.1415 en la posición 25 y luego se lleva este valor a una variable con el método at(). Si se quiere añadir un elemento al final simplemente se hace:

```
v.append(562.3)
```

La clase EMatrix (basada en un std::vector<std::vector>> de C++) representa una matriz bidimensional con un número de filas y columnas. Son contenedores secuenciales similares a los arrays bidimensionales (eg REAL v[4,5]) pero con importantes mejoras para el uso dinámico de memoria. Todos los objetos se almacenan contiguos en memoria. EMatrix también gestiona el tamaño en memoria de forma automática como EVector. Se pueden hacer cosas bastante sofisticadas y de una manera sencilla por ejemplo:

```
OBJECTS
    EMatrixReal mr
INIT
    mr.set(800,500,888)
    mr.clear()
    mr.assign(790,467,5)
    mr.replace(100,100,4)
    value= mr.at(400,400)
```

Este código hace lo siguiente:

- pone el valor 888 en el índice (800,500) con lo cual la matriz se autodimensiona en memoria a esa capacidad.
- Borra la matriz de memoria por completo
- Crea una nueva matriz asociada a mr con las dimensiones [790,567] e inicializa todos los valores de la matriz a 5.
- Reemplaza el valor en el índice (100,100) a 5
- Lleva el valor del índice (400,400) a la variable value.

La clase EDictionary (similar la std::map<key,value> en C++) representa un diccionario que es un contenedor asociativo (no secuencial) que ordena sus elementos de acuerdo a su clave. Cada elemento de un diccionario es un par <clave,valor> donde la clave puede ser de dos tipos: STRING o INTEGER, puesto que siempre vamos a acceder al diccionario por una de estas claves. El valor puede ser cualquier tipo básico (REAL, INTEGER, STRING, BOOLEAN), un enumerado o un objeto de otra clase definida en EL.

La mayor ventaja de los contenedores asociativo (EDictionary y de ESet) es que encontrar un elemento es muy rápido pues tiene complejidad logarítmica (en EVector y EMatrix es lineal). Así por ejemplo si se tuvieran 1000 elementos en un EDictionary y se quisiera buscar un elemento concreto nos llevaría de media 10 intentos hasta encontrar el objecto buscado, si se buscara en un Evector serían 500 búsquedas de media.

Se suministran varias clases de diccionarios dependiendo del tipo de clave y el valor que se quiere almacenar. Las claves son siempre o un STRING o un INTEGER. Por ejemplo el siguiente código permite crear un listín telefónico y luego hacer alguna búsqueda:

```
DECLS
    INTEGER phone
OBJECTS
    EDictionaryStringInt phones
INIT
    phones.set(“Josua”,35663678)
    phones.set(“Alan”, 37665667)
    phones.set(“Bill”,39683478)
    IF (phones.find(“Josua”,phone) == TRUE)THEN
        WRITE(“Found Josua\n”)
    END IF
```

La clase ESet (equivalente de la std::set<keyValue> en C++) representa un conjunto ordenado de objetos. EL permite tener listas ordenadas de valores de tipos básicos de EL tales como REAL, INTEGER, STRING o enumerados. Cuando se introduce un valor en un objeto ESet se ordena automáticamente internamente no permitiendo valores duplicados.

En este ejemplo se programa una función que usa un conjunto ordenado de strings en un ESet y realiza diversas operaciones:

```
FUNCTION NO_TYPE testSetString()
DECLS
    STRING v
OBJECTS
    ESetString esr
BODY
    esr.insert("set")
    esr.insert("dictionary")
    esr.insert("vector")
    esr.insert("matrix")
    WRITE("ESetString(size %d): %s\n",
          esr.size(),esr.asString() )
    IF ( esr.find("vector") == TRUE ) THEN
        WRITE("Found value vector in ESet\n")
    ELSE
        WRITE("Not found value")
    END IF
    WRITE("Erase item\n" )
    esr.erase("dictionary")
    WRITE("ESetString(size %d): %s\n",
        esr.size(),esr.asString() )
    FOR(i IN 1,esr.size())
        esr.get(i,v)
        WRITE("Pos=%d value=\"%s\"\n",i,v)
    END FOR
END FUNCTION
```

La salida de esta función es:

```
ESetString(size 4): {"dictionary", "matrix",
"set", "vector"}
Found value vector in ESet
Erase item
ESetString(size 3): {"matrix", "set", "vector"}
Pos=1 value="matrix"
Pos=2 value="set"
Pos=3 value="vector"
```

Se puede por tanto insertar valores, escribir el objeto entero, buscar elementos, borrar, listar en orden secuencial, etc. y lo mejor es que toda la gestión de la memoria se hace de forma transparente y muy optimizada.

No se incluye aquí la posibilidad de almacenar objectos complejos y todavía más importante un contenedor puede a su vez tener elementos contenedores, por ejemplo un EDictionary puede contener objetos del tipo EMatrix y a su vez cada elemento del EMatrix puede ser un EVector y así sucesivamente. Esto permite crear estructuras en memoria

muy sofisticadas y que hasta ahora era imposible realizar en EL.

13. CLASES PARA GENERACIÓN DE NÚMEROS ALEATORIOS

La nueva versión de EcosimPro/PROOSIS dispone de una nueva clase llamada EVectorRandom que permite generar números aleatorios de una manera muy sencilla. Por ejemplo:

```
FUNCTION NO_TYPE random1()
OBJECTS
    ERandomVector rn, rp, rc
BODY
    rn.populate(10, DISTR_NORMAL,7, 0.2)
    WRITE("NORMAL= %s\n",rn.asString(4))
    rp.populate(10, DISTR_POISSON,10)
    WRITE("POISSON= %s\n",rp.asString(4))
    rc.populate(10, DISTR_CHI_SQUARED ,6)
    WRITE("CHI_SQUARED= %s\n",rc.asString(4))
END FUNCTION
```

La salida de esta función es:

```
NORMAL=  7.223  6.945  7.054  7.126  7.101  7.135
7.11 6.833 7.052 7.015
POISSON= 7 11 11 10 13 15 12 7 11 9
CHI_SQUARED= 7.367 7.133 7.084 9.08 5.364 9.687
7.117 4.294 13.46 3.015
```

El método populate() es usado para generar N números usando distintas distribuciones con sus parámetros correspondientes. En este ejemplo hemos generado números usando las distribuciones Normal, Poisson y Chi-Squared. Las distribuciones disponibles son:

Normal, Log_Normal, Uniform, Uniform_Int, Bernoulli, Binomial, Geometric, Poisson, Exponential, Gamma, Chi-Squared, Cauchy, Fisher_F, Student_T y Weibull.

Si se quiere que los números generados se puedan reproducir se puede usar una semilla usando el método setSeed(), por ejemplo:

```
FUNCTION NO_TYPE random4()
OBJECTS
    ERandomVector rv
BODY
    rv.setSeed(456)
    rv.populateSorted(5, DISTR_NORMAL,7, 0.2)
    WRITE("rv= %s\n",rv.asString(4))
    rv.clear()
    rv.populateSorted(5, DISTR_NORMAL,7, 0.2)
    WRITE("rv= %s\n",rv.asString(4))
END FUNCTION
```

La salida es:

```
rv= 6.869 6.949 6.951 7.175 7.224
rv= 6.869 6.949 6.951 7.175 7.224
```

Vemos como el método setSeed() ha permitido introducir una semilla común que ha producido los mismos valores en dos llamadas distintas.

La clase también dispone de métodos que dan los valores típicos de cualquier distribución estadística: media, desviación típica, mediana, varianza, skewnednes, kurtosis, rango, valores mínimo y máximo, etc. Incluso hay un método que devuelve un resumen en modo string que puede imprimirse: statAsString(). También dispone de otros métodos como histogram() que dibuja la distribución por intervalos. Lo que se ve en el siguiente ejemplo:

```
FUNCTION NO_TYPE random7()
OBJECTS
    ERandomVector rv
BODY
    rv.populate(1000, DISTR_NORMAL,7, 0.2)
    WRITE("rv statistics= %s\n",
        rv.statAsString(4))
    WRITE("rv histogram=\n%s\n",
        rv.histogram(10,50,4))
END FUNCTION
```

Esta función genera 100 números aleatorios usando una distribución normal de media 7 y desviación típica 0.2. Luego imprime un resumen de las estadísticas típicas de la distribución generada y luego imprime un histograma que divide en 10 segmentos:

```
rv statistics= Mean: 7.002 Std Dev: 0.2009
MinVal: 6.353 MaxVal: 7.587 Range: 1.234
Skewness: -0.07264 Kurtosis: 3.023
Median: 7.006 Mode: 7 Variance: 0.04038
rv histogram=
* (0.6%)
***** (2.6%)
************ (6%)
*************************** (15.2%)
********************************** (21.1%)
************************************* (25%)
***************************** (17.2%)
***************** (9.2%)
***** ( 2.7%)
*( 0.4%)
```

A partir de ahora el usuario de EcosimPro/PROOSIS puede utilizar esta nueva clase para realizar múltiples tipos de cálculos basados en números aleatorios tales como estudios de Montecarlo.

También a través de los wizards paramétricos se usa esta clase para poder generar automáticamente estudios paramétricos basados en valores de entrada aleatorios. Y esto se hace ya de forma transparente al usuario.

14. USO SOFISTICADO DE PUNTEROS A FUNCIONES

Esta nueva versión permite un uso sofisticado de punteros a funciones que abre un nuevo mundo en la programación en EL. Se ha tratado de dotar de una interfaz muy sencilla para poder usar punteros a funciones de forma eficiente.

Lo mejor es que lo veamos con un ejemplo. Vamos a crear una función ptrFun() que a llamar a fCallre() y le va a pasar como argumento un puntero a función, que ésta a su vez va a usar para llamar a la función a la que apunta.

Primero tenemos que definir el tipo de la función que se va a apuntar, lo hacemos con la sentencia TYPEDEF que anteriormente usamos con las clases contenedoras template:

```
TYPEDEF FUNCTION REAL ftype2(REAL a, REAL b)
```

Con esto hemos creado un tipo de puntero a función que se llama ftype2 que tiene dos argumentos de tipo REAL y devuelve un REAL. Es importante pues si luego le pasamos una función que no tenga estos argumentos nos dará error de compilación.

Ahora creamos la función fCallre() pasándole el argumento de tipo puntero a función ftype2, lo hacemos con la sintaxis FUNC_PTR<ftype2>:

```
FUNCTION REAL fCallre(FUNC_PTR<ftype2> f2,REAL
a, REAL b)
DECLS
    REAL val2
BODY
    val2= f2(a,b)   --call the function f2 here
    RETURN val2
END FUNCTION
```

Esta función se limita a llamar a su vez a la función f2 que se le pasa como argumento.

Ahora definimos dos funciones que cumplen el prototypo <ftype2>, una que devuelve la suma de a y b y la otra la resta:

```
FUNCTION REAL fAdd(REAL a, REAL b)
BODY
    RETURN a+b
END FUNCTION

FUNCTION REAL fSubs(REAL a, REAL b)
BODY
    RETURN a-b
END FUNCTION
```

Por último definimos una función final ptrfun() que hace 2 llamadas a fCallre() la primera le pasa como argumento un puntero a la función de fAdd() y la otra le pasa fSubs(), e imprimimos los resultados:

```
FUNCTION NO_TYPE ptrfun()
DECLS
    REAL res=0
BODY
    res= fCallre(fAdd,50,20)
    WRITE("50+20= %g\n",res)
    res= fCallre(fSubs,50,20)
    WRITE("50-20= %g\n",res)
END FUNCTION
```

Como se esperaba la salida es:

```
50+20= 70
50-20= 30
```

Es decir la función fCallRe() dependiendo de los punteros que pasamos tendrá un comportamiento u otro. Se pueden hacer cosas mucho más sofisticadas con componentes y clases pero eso se explica en más detalle en la documentación del programa.

EA Internacional S.A.
Magallanes, 3 Madrid
28015 Spain
E-mail: info@ecosimpro.com
URL: http://www.ecosimpro.com
Phone: +34 91 309 81 42
Fax: +34 91 591 26 55

MAQUETADO POR: DAPHNE-DIANA JIMÉNEZ
REVISADO POR: ÁNGEL BARRASA

EcosimPro
Modelling and Simulation Software

PROOSIS
Propulsion Object Oriented Simulation Software