EcosimPro PROOSIS

Modelling and Simulation Software

EcosimPro/PROOSIS · Newsletter Nº 13 · February 2017

FROM THE EDITORS

Topping the "what's new" list in this new issue of our newsletter are the new versions of our products EcosimPro 5.6 and PROOSIS 3.8 released in December 2016. They are a great leap forward from the earlier versions, since they provide new tools and language capabilities than can have a great impact on the users of our products.

It has been developed a tool to test libraries and models automatically, which may revolutionize how our tools are used in an industrial setting. Until now, it took users days or weeks of work just to validate changes to their models, but now they can automate the task to run hundreds of automatic tests all night and have the exact comparison with the reference results the very next morning. This way the impact of any change can be spotted quickly.

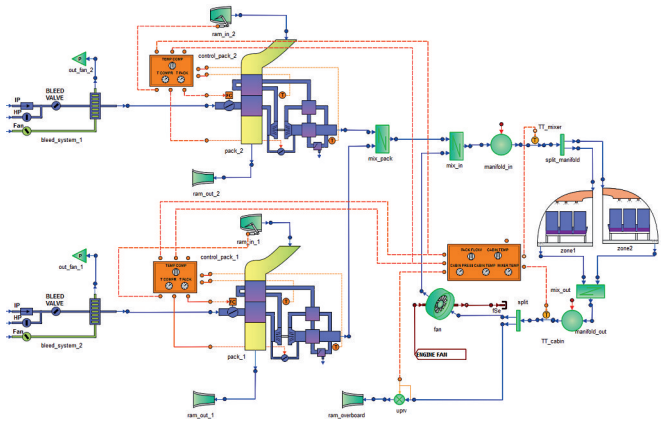
Other important improvements are: a new tool to simulate directly from the schematic without to create neither a partition nor a experiment; an exporting tool to the international standards FMI and ARP4868 for connecting our models to other tools; new transient solvers that speed up the simulations up to 50%; and new modelling language features such as container classes (e.g. dictionaries), acausal equations, pointers to functions, etc. that makes the modelling work more flexible.

Also worth noting is the great number of papers given using our tools presented by our users at the Space propulsion conference in Rome and the Aeronautical Propulsion conference ASME TURBO in Korea. We've updated our website with more than 25 papers using EcosimPro / ESPSS and PROOSIS / TURBO.

An interesting example to mention is the Indian company Team Indus, which has been pre-selected for the prestigious Google Lunar XPrize for sending a spacecraft to the moon in 2017, landing it on the moon and sending high-def images back to Earth. Team Indus uses EcosimPro/FluidaPro in designing such a sophisticated project.

This new year, we face important challenges, such as allowing our models to interconnect with other tools and standards used in the industry, real-time simulation, and others that infuse our team with an ongoing spirit of creativity.

Pedro Cobas (pce@ecosimpro.com)
Head of the Development Team of EcosimPro/PROOSIS
EA Internacional



ECS Model in PROOSIS

CONTENTS

1. ITER CRYOGENIC SYSTEM SIMULATOR

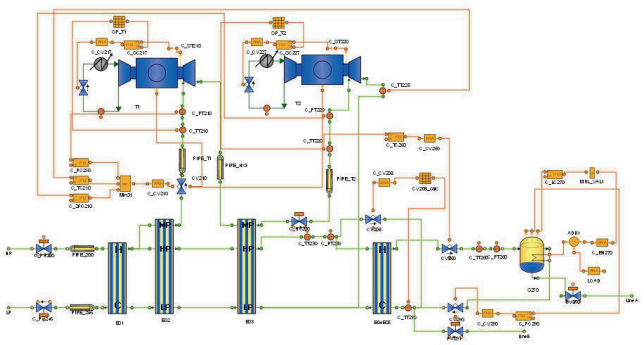
Ana Veleiro, EcosimPro/PROOSIS

One of ITER Organization's goals is to develop an integrated simulator of the different systems making up the ITER experimental reactor under construction in Cadarache (France). The simulator is meant to bring together the individual simulators developed in the different systems and integrate them. The final purpose of the integrated simulator is to support the commissioning, the engineering support during its operation and maintenance, and the training of operators.

In Cryogenics, ITER has been working for some time on developing models that can verify the design, design advanced control algorithms and test the control with hardware simulations in the loop. With this aim, the simulation department at EAI has developed dynamic models of the circuits that cool the ITER magnets and initial models for the cryogenic pumps and their distribution.

Because of the complexity of the system, IO has proposed creating a distributed simulation platform that can integrate models from the different subsystems and simulate them jointly. To achieve this goal, EAI has been entrusted with developing some specific features during 2017 to make this integration possible.

In the framework of this project, EAI will endow EcosimPro with the capability of generating OPC UA servers from the tool itself. This will let models developed in EcosimPro connect to other tools that have an OPC US interface and create a distributed simulation platform based on this technology. Similarly, a synchronization mechanism will be developed among the different elements and will work on optimizing the models for this sort of applications.



2. SPACE PROPULSION EDUCATIONAL LIBRARY (LPRES)

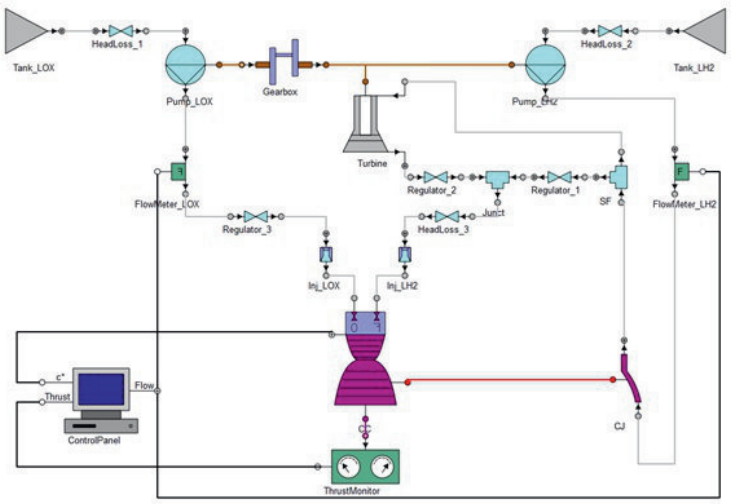
Pablo Sierra, EcosimPro/PROOSIS

Liquid propellant rockets engines are efficient systems of aerospace propulsion that are part of the content taught at engineering degrees related to the aerospace sector. The calculation tools used for it are very sophisticated, require a large amount of input data and a skilled user to use them, so they are not suitable for use in a classroom. Therefore, EAI, together with the Technical University of Madrid (UPM), have developed the LPRES library (Liquid Propellant Rocket Engine Simulation) in EcosimPro, which is a simplified library to simulate liquid propellant rockets engines.

LPRES aims to overcome typical training troubles by creating an EcosimPro library that can be used in the classroom. That is, with a short learning curve, and using models and concepts similar to the ones the student uses in solving the class work problems on "paper and pencil".

LPRES emulates the elements of the ESPSS professional libraries, containing components for predicting the steady-state behaviour of the different possible configurations of a liquid propellant rocket engine. The main simplifications included in the LPRES library are: perfect gases and liquids, phase change limited to some components and analytical models.

Furthermore, several examples of using the LPRES library are included in the LPRES_EXAMPLES library. These examples greatly facilitate the user's learning process. A gas generator cycle has been made, an expander cycle, a pressurized rocket engine, and even the cycle of a jet engine. To validate the results, a comparison of results with the ESPSS toolkit or with measured values has been made in some examples.



Model with the LPRES library. Simulation of RL10 rocket engine

In conclusion, the library has simple components that make it easier to use because they work with a small number of data. Therefore, its learning curve is short and the user does not have to be an expert to use it. Moreover, LPRES can be successfully compared with real data and reliable results are obtained fast. For all these reasons, LPRES is a very useful tool for educational purposes.

Finally, LPRES has the potential to grow in the next future, by adding, for example, a mass model, the capability to perform transient simulations, for which EcosimPro is the ideal software, or by adding more detail to the description of the substances.

3. SMART_GRID & RENEWABLES LIBRARY TEST CASE

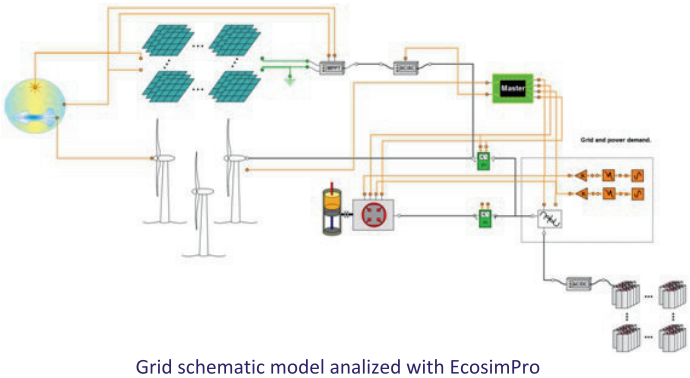
VÍCTOR PORDOMINGO, ECOSIMPRO/PROOSIS

Energy generation and transport systems are nowadays in an important renewal process. Smart grids and distributed energy resources will play the main role in the sector.

The new SMART_GRID & RENEWABLES Library provides the user with the necessary tools to study energy grids formed by different plants (at the moment photovoltaic panels, wind turbines, and diesel generator and, in the future, hydroelectric, thermosolar, fuel cells,...) working in stand-alone or grid-connected mode. The user will be able to study optimum daily generation under different meteorological conditions, demand behaviour or local and central control configurations. The library capabilities can be extended by using other EcosimPro libraries, especially ELECTRIC_SYSTEMS, CONTROL, MECHANICAL or THERMAL. Also complex calculations may be created effortless using the EcosimPro assistants (parametric studies, optimization, Monte Carlo simulation, etc).

The first test case of the library is already available in EcosimPro web and is briefly described in the following lines. The system simulated includes three different energy sources (photovoltaic panels, wind turbines and diesel backup) which must satisfy a certain demand which varies during a 24 hours period. The grid includes, also, a battery bank for energy storage. The whole system is managed by the “master” controller, which distributes the generation in the most efficient way in order to satisfy the demand in every moment considering the available resources.

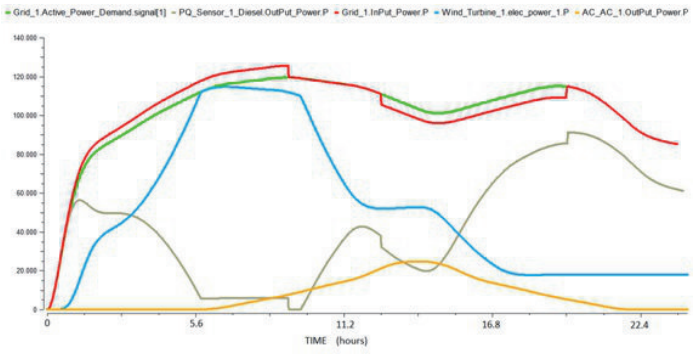
The schematic in the figure shows the photovoltaic and wind generation blocks connected to a certain environmental conditions: solar radiation, temperature and wind speed. At the bottom of the schematic, the diesel generator and battery bank complete the system. All the components are connected to the correspondent controllers, which warranties that enough energy is generated so that the demand is satisfied. This demand profile can be also configured by the user in an intuitive way.



Grid schematic model analized with EcosimPro

The simulation results show the daily active power balance. In green colour appears the demand behaviour with two maximum points during the day. In red colour the global power generation has been plotted. It is possible to see how the power generation follows the demand. There can be seen two periods of over-generation and under-generation which corresponds to battery charge and discharge processes according to the controllers criterion. In blue colour appears

the wind power generation and in orange colour the photovoltaic power generation, which are used as much as possible in order to minimize the fuel consumption to the strictly necessary levels. That is why the diesel power generation, plotted in grey colour, is so varying. It is only used to cover the part of the demand beyond the renewable generation.



Daily active power balance in the system

4. PROOSIS IN ASME TURBO, KOREA

Alex Alexiou, National University of Athens

The ASME TURBO EXPO 2016 was held in Seoul, Korea. It gathered 3000 participants from industry, research, government and academi in Gas Turbines from all over the world.

It emphasized the significant opportunities for profit, reliability and performance increases in the turbomachinery industry which are created from the intersection between the physical and digital worlds. Modelling and Simulation capabilities play a central role in exploiting these opportunities and PROOSIS, as a state-of-the art gas turbine modelling and simulation environment, is already making a significant contribution.

PROOSIS participated in this conference through eight (8) technical papers from the National Technical University of Athens in Greece, Cranfield University in the UK and ISAE Toulouse in France.

Specifically, PROOSIS and its TURBO library were used to model and simulate advanced engine concepts such as the Open Rotor which is considered the powerplant for next generation single aisle aircraft. Three papers were presented on this subject (GT2016-56645, GT2016-57918 and GT2016-57921) in which PROOSIS was used to develop performance models of the novel engine components such as the contra-rotating propellers and turbines and perform studies at component and engine levels. PROOSIS multi-fidelity simulation capabilities were exemplified in paper GT2016-56617 that demonstrated the integration of an in-house two-dimensional fan code into a PROOSIS turbofan engine model. In the spirit of the conference's keynote theme.

PROOSIS was also used to develop the digital 'twins' of two gas turbines in a Combined Heat and Power plant for health and performance monitoring purposes (GT2016-57722). The ability of the TURBO library in PROOSIS to deal with transient phenomena was showcased in GT2016-57257 where model predictions are compared with engine measurements. Finally, PROOSIS potential for multi-disciplinary simulations was highlighted in papers GT2016-57700 and GT2016-57272 that deal with different aspects of gas turbine solar hybridization.

5. ECOSIMPRO/ESPSS IN THE SPACE PROPULSION CONFERENCE

The European Space Propulsion Conference 2016 was held in Rome during the first week of May. It was organised by the European Space Agency (ESA) and other bodies involved in the space sector.

A team from EAI participated in this conference in order to present details of the products available for space propulsion simulation, that were displayed at the stand: EcosimPro, ESPSS (European Space Propulsion Simulation System), FluidaPro, Pipeliqtran, etc.

The EcosimPro/ESPSS toolkit was developed by a European consortium of companies and universities led by EAI and is currently the official ESA tool for 0D-1D modeling of space propulsion systems. ESPSS models propulsion systems for spacecraft and satellites, including firing processes, fuel tank evolution, heat exchangers, mechanical systems and electronic pressure regulators.

Among the new features developed in the libraries over the last year and presented in this forum are the simulation of a propulsion system coupled with the dynamics of a spacecraft, the inclusion of new high precision numerical schemes which are very efficient in terms of simulation speed, the inclusion of a module for the definition of the advanced geometry of solid fuel, the validation of solid and hybrid combustors compared to experimental results, and the inclusion of new components

and new options.

EcosimPro/ESPSS is currently being used by the majority of European aerospace companies for the design of new space propulsion systems. This can be seen from the large number of papers presented by companies during the conference that included models developed using this software and that are avaible in the Web. EAI presented a paper entitled "ESPSS Model of a Simplified Combined-Cycle Engine for Supersonic Cruise".

ESA and EAI organised the fifth workshop for EcosimPro/ESPSS users which hosted some 40 engineers from the European industry. Apart from presenting the latest improvements and developments underway, a round table was also held at which users could propose new improvements for future versions.

6. TEAM INDUS USES ECOSIMPRO / FLUIDAPRO FOR THE GOOGLE LUNAR XPRIZE

Google's Lunar XPRIZE competition seeks to develop efficient, economical access to the Moon by promoting technological innovation by entrepreneurs the world over. The goal of the competition is to land a vehicle on the Moon's surface, move it at least 500 metres and transmit high-definition photos and video back to Earth. The competition has now entered its final phase, with 5 teams pre-selected to prepare for launch in 2017.

One of them is Team Indus (from Bangalore, India). This small Indian startup is developing a project to go to the Moon and transmit back video images. Team Indus is using our EcosimPro/FLUIDAPRO tool to perform the fluid-dynamics analysis of its propulsion system.

We here at EcosimPro are very pleased that they are one of the teams that has been preselected by Google to compete for this important prize, and we wish them all the best in this final phase of the competition.

For more information:
http://lunar.xprize.org/news/blog/meet-5-teams-are-launching-moon-year

7. LIBRARY VALIDATION TOOL

FERNANDO CARBONERO, ECOSIMPRO/PROOSIS

The latest version of our products comes with a new tool that is bound to take out users by storm: a tool that can automate the validation of libraries and models. This new tool can save days of work by automating a lot of what had to be done by hand before. Now they can run automatically all night and have the results waiting for you the next morning.

The same need always arises whenever components are modified in EcosimPro: knowing if the results are still valid after making the changes. This task can quickly become too costly. Each model requires several steps, and if they had to be repeated on a large number of models, the overall process would become unwieldy.

EcosimPro 5.6 boasts a new tool for running most of the processes automatically. With a few clicks of the mouse, we need only sit back and wait for the results for our analysis.

Let's start by checking if the modeling phase is still valid. First we'll check the code of all the workspace libraries. That way we will see if any change in a component's source code affects other that inherit from it or use it in their topology: name changes, suppression of a variable, etc.

We'll also find out if our old partitions are still valid or if some of them need editing. And finally, we'll check to see if the experiments are still compiling with those changes.

In the simulation phase, the engineer time saved is appreciable. With dozens if not hundreds of models to run, the user would have to spend hours in front of the computer until the task is finished by hand. With this tool we can start it up and kick back until all the models have finished.

If any model doesn't give the expected results, we can quickly and comfortably access a comparison of the results with respect to the references.

The report gives us information on how many calculations were done (transient, stationary) and their results. Similarly, it says if any variables chosen for the comparison are no longer in the model.

There are two types of comparisons. First there is a statistical mode that stores the statistical values of the variables selected for comparison. This mode is especially useful when the number of resultant points per variable is high, such as in a transient.

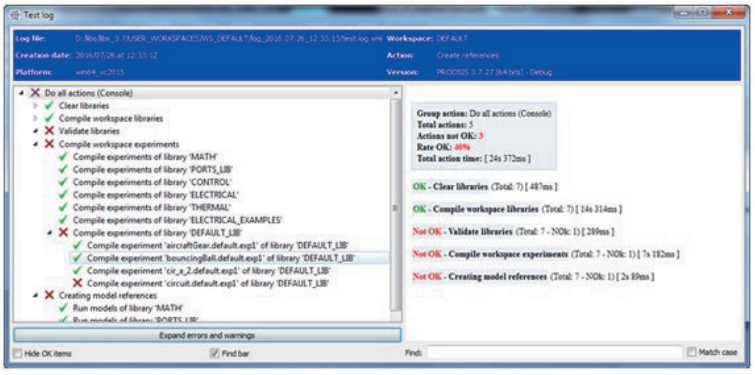
We can also go further with a thorough comparison of each value obtained per variable. For instance, in a parametric study based on stationary calculations.

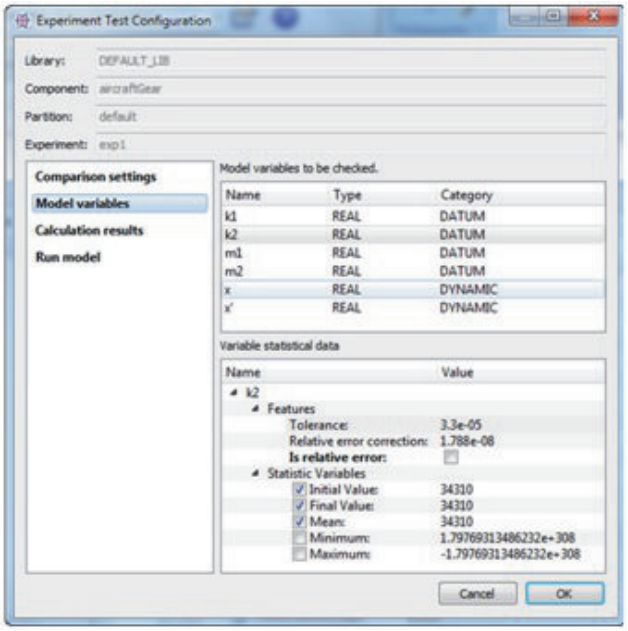
So what references do we compare against? This turns out to be one of the simplest processes. We start off with a library we trust. In other words, we know it works and works well.

We choose the models to validate, we set the parameters and let the program churn out all the results we will need as reference for both modeling and simulation.

There is a possibility of configuring the parameters of the workspace, even getting an itemized breakdown by configuring an individual model, variable by variable if necessary.

In short, we have a tool we can use to validate our models or a subset of them quickly and efficiently. And we can do this process with just a few clicks as many times as we need to, saving hundreds of hours of engineering.





8. NEW TRANSIENT SOLVERS

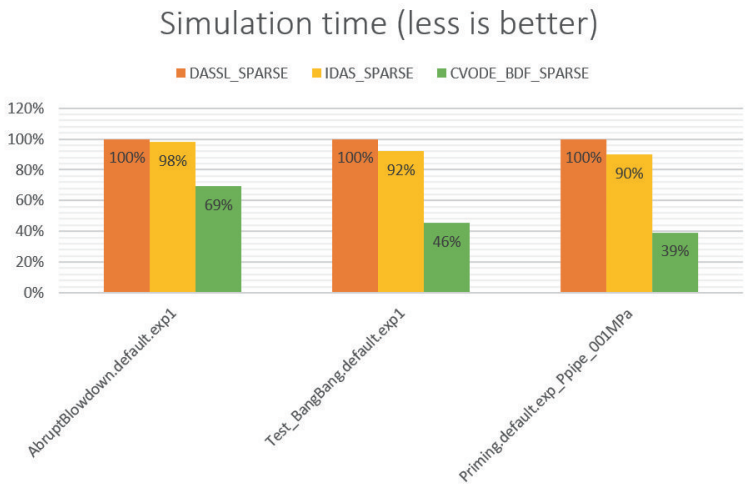
FERNANDO PUECH, ECOSIMPRO/PROOSIS

EcosimPro 5.6.0 now includes the new transient solvers IDAS, IDAS_SPARSE, CVODE_AM, CVODE_BDF and CVODE_BDF_SPARSE. IDAS and CVODE belong to be SUNDIALS package of solvers (SUite of Nonlinear and DIfferential/ALgebraic equation Solvers) developed by the "Center for Applied Scientific Computing Lawrence Livermore National Laboratory". IDAS is a general-purpose solver for the initial value problem in systems of differential algebraic equations, also known as DAEs.

IDAS is based on the DASPK solver and uses a BDF integration method of variable order and variable coefficients. CVODE is a problem solver of stiff and nonstiff initial values of ordinary differential equations and is based on the VODE and VODPK solvers. Available in EcosimPro is CVODE_AM, ideal for nonstiff problems, and whose integration method is variable order and variable step based on the Adams-Moulton formulas, whereas CVODE_BDF, ideal for stiff problems, uses a variable order, variable step method based on BDF (Backward Differentiation Formulas).

EcosimPro also includes the sparse versions of the previous solvers as a major improvement over the DASSL-SPARSE solver. First because they use a multi-thread method in the internal iterations and second because they use an optimized version to obtain Jacobian scattering.

What do all these improvements mean? Shorter simulation times, with all the benefits for the user accordingly. The following graph compares the simulation times using EcosimPro 5.6.0 and the compiling platform win64_vc2015 from different models of ESPSS 3.1.0. In the experiments the relative error (REL_ERROR) and absolute error (ABS_ERROR) are set at 1.0×1E-6.



The data in this chart show that the new solvers are a substantial improvement over the previous generation (especially CVODES). Give them a try!

9. EXPORTING MODELS WITH AN FMI 2.0 INTERFACE FOR CO-SIMULATION

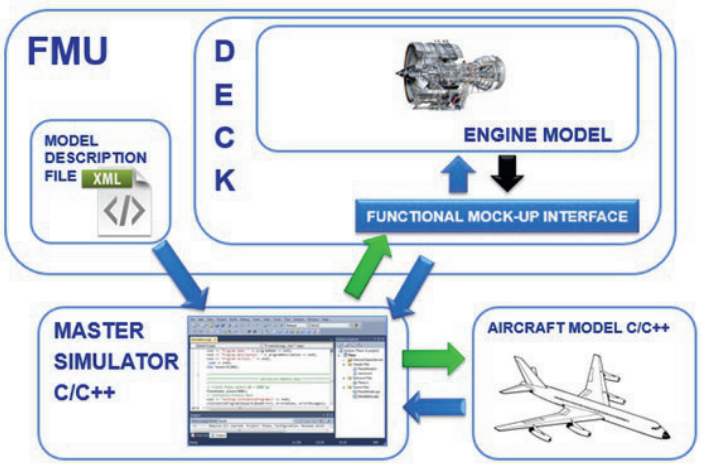
Fernando Puech, EcosimPro/PROOSIS

EcosimPro 5.6.0 introduced the ability to export a model using the FMI 2.0 standard for co-simulation. What is FMI 2.0? A standard meant to facilitate communication between simulation models developed in different tools. The standard has the following basic concepts:

- FMI, which stands for Functional Mock-up Interface, this is a well-defined ANSI C interface. It may be used by a simulation environment to load one or more models made by other simulation tools.
- FMU, which stands for Functional Mock-up Unit, this is a compressed file that in turn contains a model description in XML format and the executable files that implement the functions defined in the FMI. It can be used as a slave model in a simulation environment.
- There are two types of FMUs: FMI for co-simulation: the model has its own solver / integrator. This is the type of FMUs that EcosimPro generates. FMI to exchange models: the model waits for the simulator to carry out the numerical integration.

To export the EcosimPro models using FMI 2.0 as the interface, we set up a simple experiment with rules that are explained in the documentation and then we use the wizard to select the option to export to FMI.

Once we have the experiment and have run it at least once, we can launch the wizard to create DECKs by right-clicking on the experiment. As the startup screenshot shows, the wizard lets you enable the FMI 2.0 interface. The next step is to choose the input and output variables to be published, and lastly, to press the "generate" button. Clearly, in a few simple steps the wizard generates the file model_xxx.fmu which in turn can be uploaded into any tool that can read FMI 2.0 for co-simulation.



The best thing about this new capability is that when you have to share models with a client who does not have EcosimPro, you don't have to transpose the model to a different language or tool, which thus means major savings in time and money when developing models.

10. NEW CAUSAL ASSIGNMENT OPERATOR

Until now, EcosimPro/PROOSIS let only non-causal equations be entered in the CONTINUOUS block of components and ports. But with this new version, you can write causal equations that dictate how a variable is to be calculated.

From now on, when you write equations in the CONTINUOUS block of a component or port, the "=" and ":=" operators can be used indistinctly. Use the "=" operator whenever you want to express the equation in totally non-causal format (as always); this means that EcosimPro can transform this equation symbolically as it sees fit; for example, given the following equation:

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

The sorting algorithm can write this equation (depending on the unknown being calculated) in one of two ways:

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

This is very advantageous and gives great flexibility to the tool so that the same model can be used in very different scenarios. However, sometimes the modeler does not want an equation to be transformed symbolically but to keep the original format, which is why we have added the ":=" operator

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

We are instructing the sorting algorithm not to symbolically transform the equation and to keep the left-hand and right-hand parts intact. In this case, at the end of the sorting of the equations, this equation either calculates x explicitly or it converts it to a residue:

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

In these cases, this mechanism prevents a symbolic transformation that could jeopardize convergence or we may simply always want this equation to calculate this variable. We are imposing causality on the equation.

Another difference between "=" and ":=" is that the former operator allows both "expression= expression" and also "variable= expression" while the latter only allows "variable= expression".

For example, this is correct:

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

but it is not correct to write:

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

That is because causal equations must always be used to calculate a specific variable.

The operator ":=" is a more flexible alternative to the prefix EXPL that searches for an equation to calculate a variable. For example, if we have a component like:

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

Until now, this mechanism was the one used to find an explicit equation for calculating x. The problem was when there were two equations, such as:

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

In this case it gave the following error when making the partition:

```
*** 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.
```

It is best to use ":=":

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

In this case it forces us to calculate x with the first equation. Therefore, from now on, we recommend not using the EXPL prefix and using this new assignment operator instead, which is more intuitive and robust.

11. USING TEMPLATES IN EL

A new sophistication in EcosimPro 5.6/PROOSIS 3.8 is the use of templates. They allow to define new data types base on some predefined template classes provided in the tool (container classes).

When template classes are used in EL, the TYPEDEF statement can be used to define a new type of class inherited from the template class applied to a specific class. For example, we could use the template class EVector<className> to create a vector of objects of type "className" and define:

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

The new "newClassName" class can be used from this point on as any other class and represents a vector of objects of the "className" class. Following is an example with a simple class called "Employee" with two attributes "name" and "phone", and with a vector of objects of this type:

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

TYPEDEF CLASS VectorEmployees IS_A
EVector<Employee>
```

From this moment on, the class "VectorEmployees" can be used just like the others. For instance, we can add a new employee the same way as in the EVector class:

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

We see how a new class has been defined inherited from a container class that can contain complex objects. From here on, it is even possible to make new template classes based on others, such as

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

Now we have a new class we can use in our EL code. For example:

```
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)
```

We see we can create more and more sophisticated classes using the templates and container classes. This gives a huge added value to EL by being able to create new types of data that represent complex structures in memory.

12. NEW CONTAINER CLASSES

EcosimPro 5.6/PROOSIS 3.8 provides new powerful container classes. The new container classes are generally an alternative to using arrays in EL. At the moment, a user can use an array to store a data vector (for example REAL v[4]) or a matrix (e.g. REAL v[4,4]), but has the limitation of not being dimensionable dynamically. Users can rely on these more advanced containers that allow the automatic handling of memory. On the other hand, their syntax is less intuitive, since access to them is done through class methods (e.g. v[3] vs v.at(3)). Users will need to determine whether it is better to use an array or a container of this type for a specific application.

There are essentially two types of container classes:

- Sequential containers: Classes for the adjacent storage within the memory. These are the EVector and EMatrix classes.
- Associative containers: Classes with no adjacent storage of information in the memory and that are always arranged with a key. These are the EDictionary and ESet classes.

These classes have been designed with three goals in mind:

- Simplicity of use
- Smart handling of memory
- Maximum efficiency in computational terms

The new containers are based on the standard STL library of C++, so they benefit from the latest advances of C++ compilers and from the millions of users throughout the world who efficiently use them on thousands of projects. The interface is not exactly the same because the one for EL has been greatly simplified.

All the container classes can contain the basic types of EL (REAL, INTEGER, STRING, BOOLEAN, ENUM) as objects of any class created in EL. This is important because it lets us create complex data structures easily and efficiently.

EVector vectors (based on the std::vector in C++) are sequential containers that are similar to the one-dimensional arrays of EL ((eg REAL v[3]), but they are smarter to handle the memory. Adjacent storage of objects in the memory shall be used. EVector handles its elements dynamically and allows direct access to each element based on an index. Elements can be added at the end or deleted quickly. However, inserting objects between two elements is not optimum because, internally, it is necessary to move all the objects to make way for the new ones. A vector does not need to define the initial size; when we insert an element into a position, it resizes internally to have at least those elements, for example:

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

In this example, the value 3.1415 is inserted into position 25 and then this value is taken to a variable with the at() method. Adding an element at the end is also easy using:

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

The EMatrix class (based on std::vector <std::vector>> from C++) represents a two-dimensional matrix with several rows and columns. They are sequential containers that are similar to the two-dimensional arrays (eg REAL v[4,5]), but with significant improvements for the dynamic use of memory. Adjacent storage of objects in the memory shall be used. EMatrix also automatically manages the size in memory as EVector. This makes it simple to do quite sophisticated things, such as:

```
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)
```

This code does the following:

- it set the value 888 in the index (800,500), so the matrix resizes in memory to that capacity.
- It deletes the matrix from the memory completely.
- It creates a new matrix associated with mr with the dimensions [790,567] and initializes all the values of the matrix to 5.
- It replaces the value in the index (100,100) to 5
- It takes the index value (400,400) to the value variable.

The EDictionary class (similar to std::map<key,value> in C++) represents a dictionary that is an associative (non-sequential) container that orders its elements according to its key. Each dictionary element is a pair <key,value> where the key can be of type types: STRING or INTEGER, as users will always access the dictionary via one of these keys. The value can be of any basic type (REAL, INTEGER, STRING, BOOLEAN), enumeration or object of another class defined in EL.

The main advantage of associative containers (EDictionary and ESet) is that finding an element is very fast, as it has logarithmic complexity (while in EVector and EMatrix it is linear). For example, if there were 1000 elements in an EDictionary, it would take an average of 10 attempts to find a given element, while searching in EVector would take an average of 500.

Various classes of dictionaries are predefined depending on the type of key and the value we wish to store. The keys are always either a STRING or an INTEGER. For example, the following code lets us create a telephone list and then search it:

```
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
```

The ESet class (equivalent to std::set<keyValue> in C++) represents an sorted set of objects. EL allows sorted lists of values of basic EL types such as REAL, INTEGER, STRING or enumeration. When a value is entered into an ESet object it is automatically sorted internally and no duplications are allowed.

In this example a function is programmed that uses a sorted set of strings in an ESet and carries out different operations:

```
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
```

The output of this function is:

```
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"
```

As you see, you can insert values, write the entire object, search for elements, delete, list in sequential order, etc. and the best of all is that all the memory management is done transparently and fully optimized.

In this article we have not gone into the possibility of storing complex objects, and even more importantly, a container can in turn have container elements, for example an EDictionary can contain EMatrix type elements and each EMatrix element can in turn be an EVector and so on. With this, you can create highly sophisticated structures in memory that up to now had been impossible to do in EL. Please refer to the Modelling Language Manual for more detailed information and examples.

13. CLASS FOR GENERATING RANDOM NUMBERS

The new version of EcosimPro/PROOSIS has a new class called EVectorRandom that makes it very simple to generate random numbers. For example:

```
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
```

The output of this function is:

```
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
```

The populate() method is used to generate N numbers using different distributions with their corresponding parameters. In this example, we have generated numbers using the Normal, Poisson and Chi-Squared distributions. The distributions available are:

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

If you want the generated numbers to reproduce, you can use a seed, such as:

```
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
```

The output is:

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

We see how the setSeed() method lets us add a common seed that yields the same values in two different calls.

The class also has methods that give typical values of any statistical distribution: average, standard deviation, median, variance, skewness, kurtosis, range, minimum and maximum values, etc. There is even a method that returns a summary in string mode that we can print out: statAsString(). It also has other methods such as histogram() that outlines the distribution by interval. Let's see an example:

```
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
```

This function generates 100 random numbers using a normal distribution with an mean of 7 and standard deviation of 0.2. It then prints a summary of the standard statistics of the distribution generated and then prints a histogram that splits it into 10 segments:

```
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%)
```

From now on, EcosimPro/PROOSIS users can use this new class to perform a number of types of calculations based on random numbers such as Montecarlo studies.

The parametric wizards also make use of this class to automatically generate parametric studies based on random input values. This is done in a way that is transparent to users.

14. SOPHISTICATED USE OF FUNCTION POINTERS

EcosimPro 5.6/PROOSIS 3.8 introduces the sophisticated use of function pointers that open up a new world in EL programming. We have kept the interface as simple as possible for using function pointers efficiently.

The best is to see it in an example. Let's create a function ptrFun() that calls fCallre() and sends it a function pointer as its argument, which the latter will in turn use to call the function it points to.

First we have to define the type of function to be pointed to:

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

With this we have created a type of function pointer called ftype2 that has two REAL-type arguments and returns a REAL. This is important because if we then give it a function that does not have these arguments, it will give us a compilation error.

Now we create the function fCallre() giving it the function pointer type argument ftype2, which we do with the syntax 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
```

This function is limited to calling the f2 function it was given as an argument.

Now we define two functions that fulfill the prototype <ftype2>: one that returns the sum of a+b and the other the returns the difference of a-b:

```
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
```

Lastly, we define a final function ptrfun() that makes 2 calls to fCallre(): the first one gives it a function pointer fAdd() as the argument and the other is passed to fSubs() and we print out the results:

```
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
```

As expected, the output is:

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

In other words, the function fCallRe() behaves differently depending on the pointers we give it. Much more sophisticated things can be done using components and classes, but that is explained in more detail in the Modelling Language with many examples.

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

EDITED BY: DAPHNE-DIANA JIMÉNEZ
REVIEWED BY: ÁNGEL BARRASA

EcosimPro
Modelling and Simulation Software

PROOSIS
Propulsion Object Oriented Simulation Software