EMSO: An Integrated Tool for Process Modeling, Dynamic Simulation and Optimization

R. de P. Soares[1] and A.R. Secchi[2*]

Departamento de Engenharia Química - Universidade Federal do Rio Grande do Sul

Rua Sarmento Leite 288/24 - CEP: 90050-170 - Porto Alegre, RS - Brasil

{rafael[1], arge[2]}@enq.ufrgs.br

Key words: High-index DAE systems; DAE initialisation; index reduction; dynamic process simulator.

---

[*] Author to whom correspondence should be addressed.

Abstract

The general purpose equation-based CAPE tool called EMSO (Environment for **M**odeling, **S**imulation and **O**ptimization), introduced in ESCAPE 13 is presented. In this tool complex process engineering flowsheets can be described by simply selecting and connecting existent model bocks or new models, developed in an own object-oriented modeling language. When solving a flowsheet description there is no translate, compile or link intermediate step avoiding the setup solution overhead. In addition, the system automatically checks the consistency of measurement units, system solvability and initial conditions. The solvability test is carried out by a structural index reduction method which requires time derivatives of the original equations that are provided by a built-in symbolic differentiation system. The partial derivatives required during the numerical solutions are generated by a built-in automatic differentiation system. Currently, the major development effort in EMSO is related with extension of the external interfaces support, regarding with the implementation of the thermodynamic and unit operations CAPE-OPEN interfaces together with the support for loading custom shared libraries.

1. Introduction

Simulator is a valuable tool for applications ranging from project validation, plant control and operability to production increasing and costs reduction. This facts among others has made the industrial interest in software tools for modeling, simulation and optimization to grow up, but this tools are still considered inadequate by the users (Che-Comp, 2002). The user dissatisfaction is mainly related with limited software flexibility, difficulty to use/learn and costly. Besides the lack of software compatibility and the slowness of inclusion of new methods and algorithms. Furthermore, the users have been pointed out some desired features for further development, like extensive standard features, *intelligent* interfaces among others (Hlupic, 1999).

In this context a new computer-aided process engineering (CAPE) tool, named EMSO (**E**nvironment for **M**odeling, **S**imulation and **O**ptimization) is presented. This tool aims to give the users more flexibility to use their available resources in an up to date system which provides features like automatic and symbolic differentiation, an object-oriented modeling language, among others. The big picture of the EMSO structure is shown in Figure 1, this figure demonstrates the modular architecture of the software.
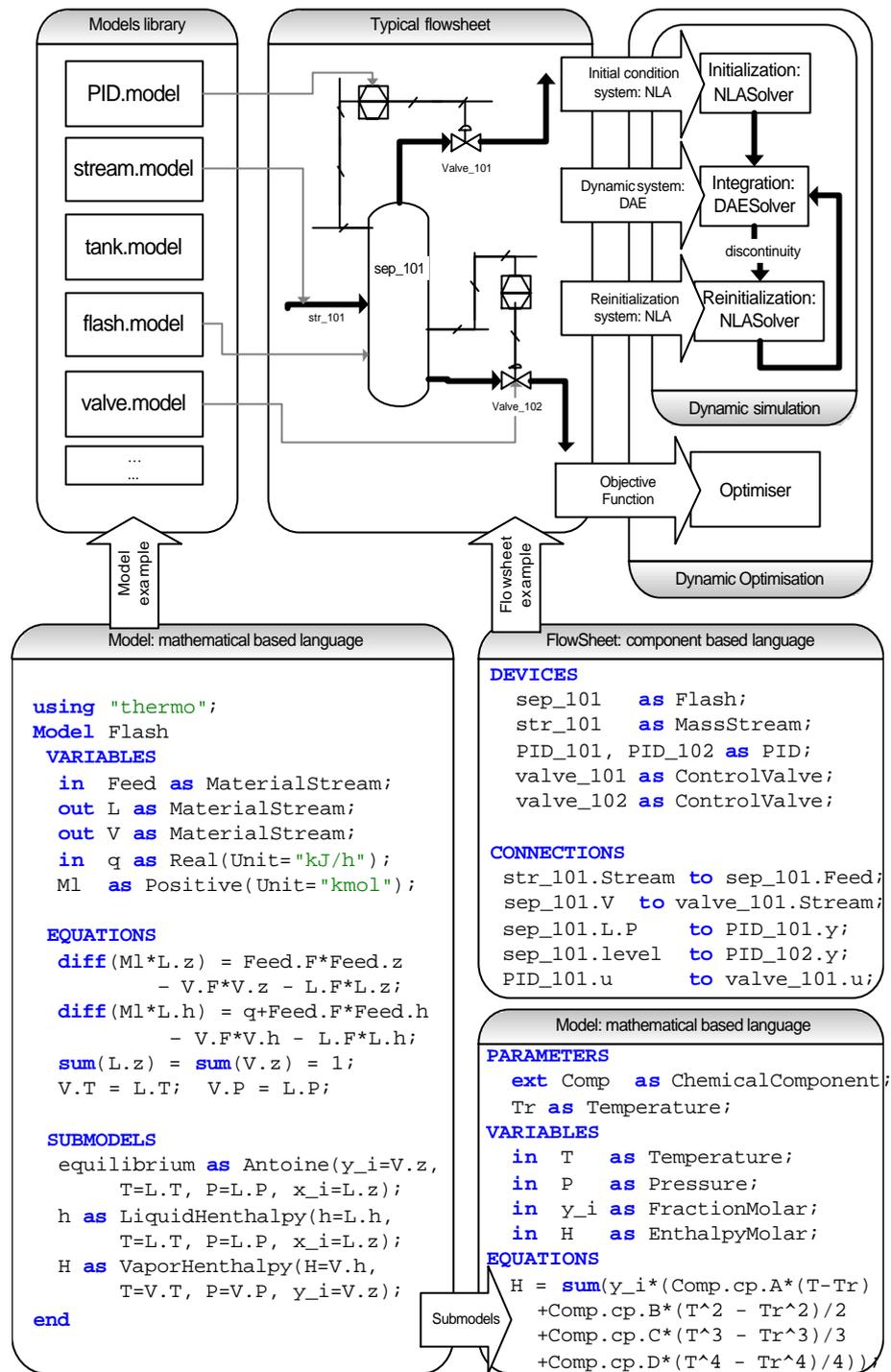
Figure 1. General vision of the EMSO structure and its components.

2. Process Model Description

In EMSO, the description of a process (a *flowsheet*) can be constructed simply creating *devices* (which are based in mathematical *model*s) and connecting them. Therefore, in the proposed modeling language there are three major entities: *models*, *devices*, and *flowsheets*. A *flowsheet* represents the process in analysis

which is composed by a set of *devices*. Each *device* has a mathematical description consisting in a *model*. At bottom of Figure 1 are given some pieces of code which exemplifies the language in use.

EMSO makes intensive use of automatic code generators and the object-oriented paradigm whenever is possible, aiming to enhance analyst and productivity.

2.1 Model

In the EMSO language, one *model* consists in the mathematical abstraction of some real equipment, process piece or even software. Examples of *models* are the mathematical description of a tank, pipe or a PID controller.

Each *model* can have parameters, variables, equations, initial conditions, boundary conditions and submodels that can have submodels themselves. *Models* can be based in pre-existing ones, and extra-functionality (new parameters, variables, equations, etc.) can be added. So, *composition* (*hierarchical modeling*) and *inheritance* are supported.

Every parameter and variable in a *model* is based in a predefined *type* and have a set of attributes like a *brief* description, *lower* and *upper* bounds, *unit* of measurement among others.

```
Fraction as Real(Lower=0, Upper=1);
Positive as Real(Lower=0, Upper=inf);
EnergyHoldup as Positive(Unit="J");
```

Figure 2. Examples of type declarations.

2.2 The Flowsheet and its Devices

In the proposed language a *device* is an instance of a *model* and represents some *real* device of the process in analysis. So, a unique *model* can be used to represent several different *devices* which have the same structure but may have different conditions (different parameters values and specifications). *Devices* can be connected each other to form a *flowsheet* (see Figure 1) which is an abstraction of the real process.

Although the language for description of *flowsheets* is textual (bottom right in Figure 1), it is simple enough to be entirely manipulated by a graphical interface. In this interface, *flowsheets* could be easily built by dragging *model* objects into it to create new *devices* that could be connected to other *devices* with the aid of some pointing unit (mouse).

Should be noted that in EMSO a connection between *devices* does not imply in the addition of an equality equation between the input and output because an input variable is only reference to the output variable connected to it.

3. Consistency Analysis

In solving the resulting system of differential-algebraic equations (DAE) of a *flowsheet*, prior analysis can reveal the major failure causes.

There are several kinds of consistency analysis which can be applied in the DAE system coming from the mathematical description of a dynamic process. Some of them are: measurement units, structural solvability and initial conditions consistency.

3.1 Measurement Units Consistency

In modeling physical processes the conversion of measurement units of parameters is a tiresome and error prone task. Moreover, a ill-composed equation usually leads to a measurement unit inconsistency. For this reasons, in EMSO the measurement units consistency and units conversions are automatically made for all equations, parameter setting and connections between *devices*.

Once all expressions are internally stored in a symbolical fashion and all variables and parameters have its measurement units as an attribute, the units of measurement consistency can be easily checked.

3.2 DAE Solvability

Soares and Secchi (2002) have proposed a structural method for index reduction and solvability test of DAE systems. With this method, structural singularity can be tested and the structural differential index can be reduced to zero by adding new variables and equations. Such variables and equations are the derivatives of the original ones with respect to the independent variable.

EMSO makes use of this method, allowing the solution of high-index DAE problems without user interaction. The required derivatives of the variables and equations are provided by a built-in symbolic differentiating system.

3.3 Initial Conditions Consistency

Once a DAE system is reduced to index zero the dynamic freedom degree is determined. So, the initial condition consistency can be easily tested by an association problem as described by Soares and Secchi (2002). This approach is more robust when compared with the index-one reduction technique presented by Costa et al. (2001).

4. External Interfaces

The ability to run external pieces of software at run time within a simulation tool is implemented by the main software vendors. Usually this is made out through some proprietary interfacing system, leading to heterogeneous mechanisms.

4.1 CAPE-OPEN Interfaces

Recently, the CAPE-OPEN project (CO-LAN, 2002) has published open standard interfaces for CAPE tools aiming to solve this problem. EMSO complies with this open pattern regarding to the numerical set of interfaces and the thermodynamic and unit operations set of interfaces are about to be implemented.

4.2 External Object Interfaces

The work of Soares and Secchi (2002b) showed that the CAPE-OPEN interfaces obligatory leads to some efficiency loss mainly because this interfacing system is designed to work transparently in networked and/or heterogeneous platforms. Because of this, besides the CAPE-OPEN interfaces an interfacing mechanism for loading external pieces of software at run time as shared objects (SO) in posix platforms or dynamic link libraries (DLL) in win32 is provided.

5. Graphical User Interface

The graphical user interface (GUI) of EMSO combines *model* development, *flowsheet* building, process simulation and results visualising and handling all in one. EMSO is entirely written in C++ and is designed to be very modular and portable. In running tasks there are no prior generation of intermediary files or compilation step, everything is made out at the memory. The software is multithread, allowing real-time simulations and even to run more than one *flowsheet* concurrently without blocking the GUI.

Furthermore, calculations can be paused or stopped at any time. The Figure 3 shows the EMSO GUI, it implements a Multiple Document Interface (MDI).
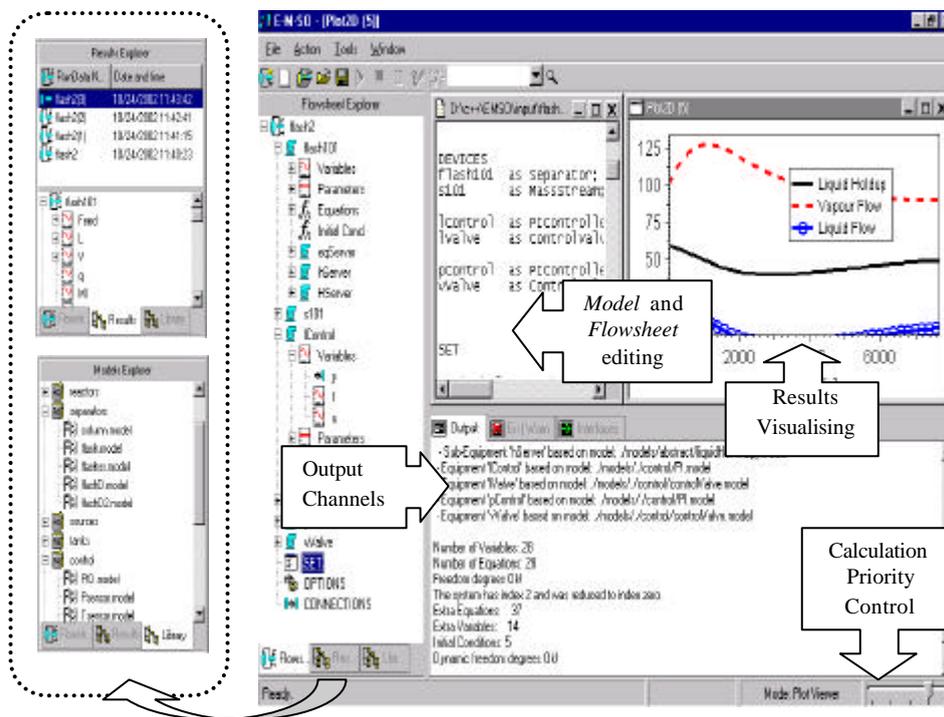

Figure 3. EMSO graphical user interface.

6. Applications

In this section examples related with the external interfaces of EMSO are presented. Examples regarding the direct solution of high-index problems with EMSO can be found at Soares and Secchi (2003).

6.1 CAPE-OPEN Merging of Models

The main purpose of the CAPE-OPEN project is to enable native components of a simulator to be directly replaced or plugged by those from another independent source. In this section an example of merging of models from different sources is presented.

Consider the flowsheet in *Figure 4*, broken in two pieces A and B. A CAPE-OPEN compliant software should be able to get each of this pieces from external and heterogeneous sources and simulate the entire flowsheet. This situation was successfully solved in an experiment with two instances of EMSO, a more detailed description of this experiment can be found at Soares and Secchi (2002b).
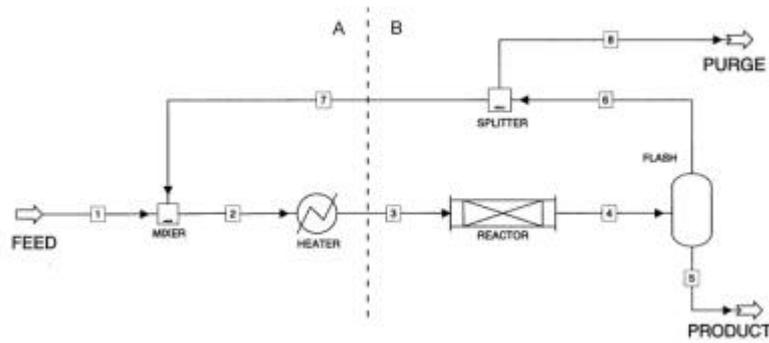
Figure 4. Flowsheet of a methanol plant.

6.1 Thermodynamic Properties from External Software

As cited before, in addition to the CAPE-OPEN interfaces EMSO provides an interfacing mechanism for loading at run time shared objects (SO) or dynamic link libraries (DLL). In order to test this capability a simple Rankine cycle process, as shown in *Figure 5* was used. The EMSO model of this process is presented in Appendix A. This process was successfully solved, resulting in a cycle efficiency of 26.5%, using a third party software (in a DLL) to calculate the steam/water thermodynamic properties. More complexes process, as an entire thermoelectrical power plant, were also solved.
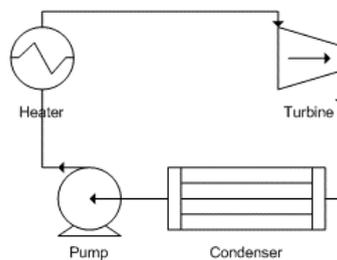

Figure 5. Rankine cycle process.

In this particular case the external piece of the software was built using the FORTRAN-90 programming language, but there is no language restriction. Actually template implementations in the major scientific programming languages (FORTRAN, C and C++) are available.

7. Conclusions

The integrated environment for modeling, simulation and optimization of general dynamic processes, named EMSO was presented. It implements an object-oriented modeling language and powerful methods

as consistency checks, index reduction, symbolic and automatic differentiation. The CAPE-OPEN numerical set of interfaces is already implemented and the thermodynamic and unit operations interfaces are the next targets.

Dynamic optimization and the solution of continuous-discrete systems are at design stage, but the modular internal architecture of EMSO allows to add it further without re-coding the other modules.

8. References

Che-Comp, 2002, The State of Chemical Engineering Softwares, www.che-comp.org .

CO-LAN, 2002, Conceptual Design Document for CAPE-Open Project, www.co-lan.org .

HLUPIC, V., 1999, Simulation Software: User's Requirements, Comp. Ind. Engrg, 37, 185-188.

LOGSDON, J. S. and BIEGLER, L. T., 1993, Ind. Eng. Chem. Res., v.32, n.4, 692-700.

OMG, 1999, The Common Object Request Broker, version 2.3.1, www.omg.org

SOARES, R. de P. and SECCHI, A. R., 2002, Direct Initialization and Solution of High-Index DAE Systems with Low-Index DAE solvers, Comp. Chem. Engrg (submitted 2002).

SOARES, R. de P. and SECCHI, A. R, Efficiency of the CAPE-OPEN Numerical Open Interfaces, Technical Reporting, UFRGS, Porto Alegre, Brasil (2002b).

SOARES, R. de P. and SECCHI, A. R., EMSO: a New Tool for Modelling, Simulation and Optimisation, ESCAPE 13 (2003).

Appendix A

It follows bellow the Rankine cycle written in the EMSO modeling language, using external thermodynamic package, called "thermo", written in FORTRAN-90. After the models are given some brief notes about the models.

```
# Type declarations
      Enthalpy        as      Real(Default=2,Lower=1e-3,Upper=10,Unit="MJ/kg");
      Entropy         as      Real(Default=5,Lower=1e-3,Upper=15,Unit="kJ/kg/K");
      Power           as      Real(Default=10,Lower=0,Upper=1e3,Unit="MW");
      Pressure        as      Real(Default=1,Lower=5e-4,Upper=100,Unit="MPa");
      Temperature     as      Real(Default=600,Lower=273.16,Upper=1073.15,Unit="K");
      Diff_Temp       as      Real(Default=0,Lower=-1073.15,Upper=1073.15,Unit="K");
      MassFlow        as      Real(Default=50,Lower=0,Upper=1e3,Unit="kg/s");
      SpecificVolume  as      Real(Default=1,Lower=1e-6,Upper=1e3,Unit="m^3/kg");
      Fraction        as      Real(Default=0.5,Lower=0,Upper=1);
      Efficiency      as      Real(Default=0.75,Lower=0,Upper=1);

Model Stream
 VARIABLES
      F as MassFlow;
      P as Pressure;
      T as Temperature;
      S as Entropy;
      H as Enthalpy;
end
```

```
Model Turbine
 PARAMETERS
  ext Prop      as ExternalObject;

 VARIABLES
  in  Fin       as Stream;
  out Fout      as Stream;
      H_IS      as Enthalpy;
      EF_T      as Efficiency;
      POT_TURB as Power;

 EQUATIONS

      H_IS = .Prop.propPS(Fout.P,Fin.S);

      Fout.H = (H_IS - Fin.H) * EF_T + Fin.H;

      [Fout.S,Fout.T] = Prop.propPH(Fout.P,Fout.H);

      Fout.F * (Fin.H - Fout.H) = POT_TURB;

      Fin.F = Fout.F;
end

Model Condenser
 PARAMETERS
  ext Prop      as ExternalObject;

 VARIABLES
  in  Fin       as Stream;
  out Fout      as Stream;
      Q_COND    as Power;
      G_S       as Diff_Temp;

 EQUATIONS

      Fout.P = Fin.P;

      Fout.T = Prop.Tsat(Fout.P) - G_S;

      [Fout.S,Fout.H] = Prop.propPTl(Fout.P,Fout.T);

      Q_COND = Fin.F * (Fin.H - Fout.H);
end

Model Bump
 PARAMETERS
  ext Prop      as ExternalObject;
      v_esp     as SpecificVolume;

 VARIABLES
  in  Fin       as Stream;
  out Fout      as Stream;
      H_IS      as Enthalpy;
      POT_BMB   as Power;
      EF_B      as Efficiency;

 EQUATIONS

      H_IS = Prop.propPS(Fout.P,Fin.S);

      (Fout.H - Fin.H) * EF_B = H_IS - Fin.H;

      [Fout.S,Fout.T] = Prop.propPH(Fout.P,Fout.H);

      POT_BMB * EF_B = Fin.F * v_esp * (Fout.P - Fin.P);

      Fin.F = Fout.F;
end
```

```
Model Boiler
 PARAMETERS
  ext Prop      as ExternalObject;

 VARIABLES
  in  Fin       as Stream;
  out Fout      as Stream;
      Q_GV       as Power;
      EF_GV      as Efficiency;

 EQUATIONS

      Fin.P = Fout.P;

      [Fout.S,Fout.H] = Prop.propPTv(Fout.P,Fout.T);

      Q_GV * EF_GV = Fin.F * (Fout.H - Fin.H);

      Fin.F = Fout.F;
end

Model Electric_Power
 PARAMETERS
  ext Prop       as ExternalObject;

      EF_GE as Efficiency;

 VARIABLES
      Pot as Power;
end

FlowSheet Rankine
 PARAMETERS
      Prop       as ExternalObject(File="thermo.dll");

 DEVICES
      Turb      as Turbine;
      Cond      as Condenser;
      Bump      as Bump;
      GV        as Boiler;
      GE        as Electric_Power;

 CONNECTIONS
      GV.Fout     to  Turb.Fin;
      Turb.Fout   to  Cond.Fin;
      Cond.Fout   to  Bump.Fin;
      Bump.Fout   to  GV.Fin;

 VARIABLES
      EF_CIC as Efficiency;

 EQUATIONS
      GE.Pot = Turb.POT_TURB * GE.EF_GE;
      EF_CIC * GV.Q_GV = (Turb.POT_TURB - Bump.POT_BMB);

 SET
      Bump.v_esp = 1.01e-3;
      GE.EF_GE = 0.92;

 SPECIFY
      GV.Fout.P = 11.3; # MPa
      GV.Fout.T = 530 + 273.15; # K
      GV.EF_GV = 0.8;
#     GV.Q_GV = 230; # MJ/s
      Turb.Fout.P = 0.007; # MPa
      Turb.EF_T = 0.8;
      Turb.POT_TURB = 63; # MW
#     Bump.Fout.F = 60; # kg/s
      Bump.EF_B = 0.7;
```

```
#      Bump.POT_BMB = 0.9; # MW
#      Cond.Fout.T = 34 + 273.15; # K
#      Cond.Q_COND = 125; # MJ/s
       Cond.G_S = 5.0; # K
#      EF_CIC = 0.275;

 OPTIONS
       mode = "steady";
       outputLevel = "high";
end
```

The model `Stream` has no equations and its function is only to hold the data that will be shared between the devices of the flowsheet. Therefore, variables deriving from `Stream` can be directly connected each other without the necessity of creation of special entities known as "streams" in the process simulators.

All models except `Stream` presents the parameter `Prop`, this parameter it the link with the external piece of the software which will calculate the steam/water thermodynamic properties. As can be seen, this parameter is declared in the models as an **ext**ernal parameter. This means that this parameter actually is only a reference to a parameter with the same name but declared in the flowsheet where the model will be instantiated as a device.

In the flowsheet `Rankine` the parameter `Prop` is not declared as an **ext**ernal parameter because it will be the source of the parameters with the same name of all its devices.