CAPE-OPEN OBJECT ORIENTED INTERFACES FOR THE DIVA SIMULATOR

Krasnik M.Y.
Donetsk National Technical University
krasnik@pisem.net

Abstract

Krasnik M.Y. CAPE-OPEN object oriented interfaces for the DIVA simulator. *Object oriented interfaces for the Diva simulator described in this article. There are model interfaces, equation solvers interfaces (differential-algebraic equation solvers, nonlinear algebraic equation solvers). Also included sequential diagrams for the full work cycle of the model and solver.*

Introduction

The scientific computing community's need for implementations of complex numerical algorithms has awakened interest in the potential benefits of object-oriented programming (OOP) compared to traditional approaches. These benefits include more flexible and extensible implementations of numerical algorithms, simplified user interfaces, and better readability from the developer's perspective.

As base of developing modeling system was chosen CAPE-OPEN standard [2]. It is a co-operative project sponsored by the European Union and aimed at defining software interfaces which allow plug-and-play simulation components within the various process simulators currently on the market (i.e. commercial, academic or "in house" simulators). The main features of the standard are:

- Steady-state and dynamic simulation for process design
- Training simulators for operators
- Data acquisition and reconciliation
- Advanced control, monitoring and diagnostic
- Optimization of processes
- Equipment design tools (heat exchanger, reactor, pipe network, pressure relief, distillation, etc.)
- Process design tools (process simulators, process network designers, etc.)

CAPE-OPEN interface classes CapeNumericESO and CapeNumericDAESO (see below) fully describe Diva models. Using CAPE-OPEN interface allows us to use models in different modeling systems, like gPROMS, HYSYS et al.

CAPE-OPEN interface classes

As base of modeling system was chosen Diva [1,8], which was developed at the Institut für Systemdynamik und Regelungstechnik (university of Stuttgart).

Figure 1. Multithreaded modeling system packages kit

On figure 1 presented diagram of developed packages:
Utility package. Basic defined classes:

- CapeUtilityComponent. Super class of the all CAPE subclasses, which holds information about name, description and version of the component.
- CapeException. Super class for the CAPE exceptions.
- CapePublicParameter. Holds information on any parameter in general.
- CapeVariant. Variant data type, which allows hold integer, real, string, pointer data types.

Matrix package. Defining the matrix types (see fig. 3):

- CapeNumericMatrix. Abstract super class for the matrix.
- CapeNumericFullMatrix. Dense matrix subclass.
- CapeNumericUnstructuredMatrix. Sparse matrix subclass.

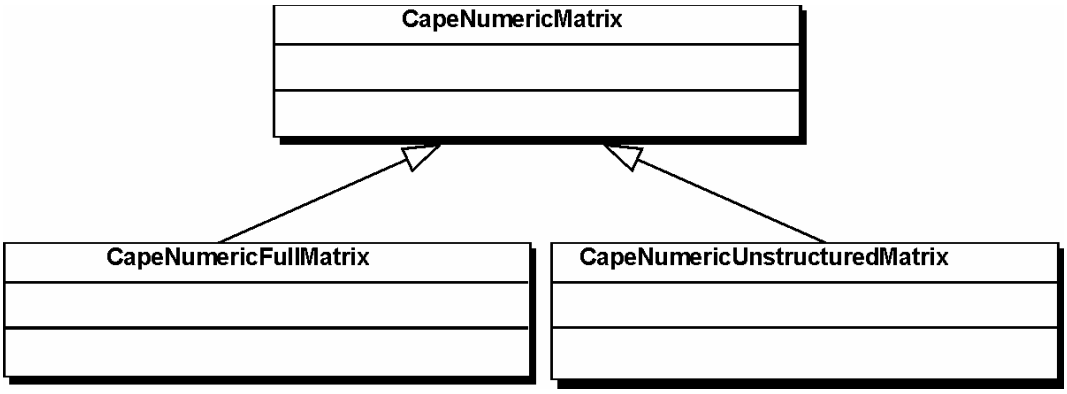

Figure 2. Class diagram of the Utility package

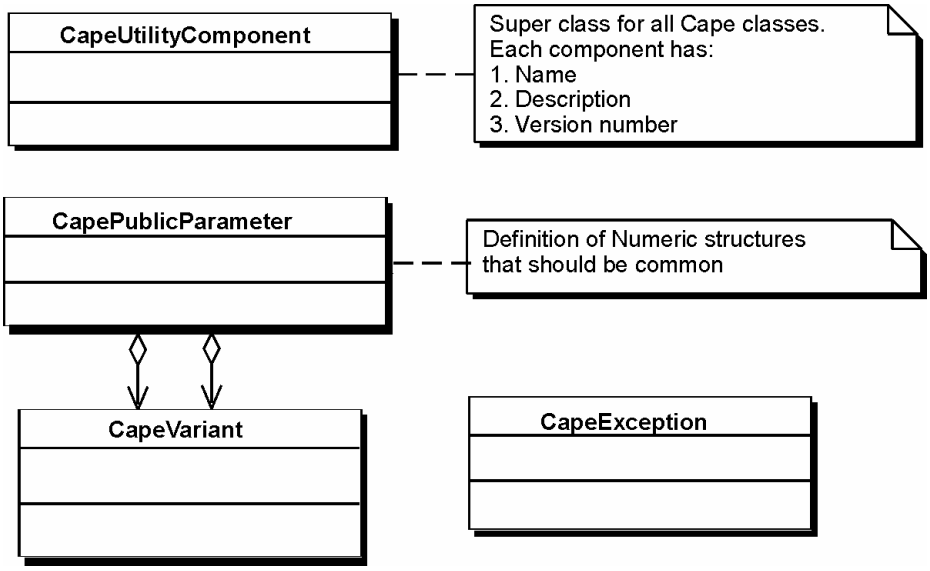

Figure 3. Class diagram of the Matrix package

Model objects

The base component of the modeling system is the model, which is the definition of large sets of nonlinear equations of any kind generally requires a large amount of relatively complex data. This has led us to introduce the concept of an Equation Set Object (ESO) as a means of defining this information in a way that can be accessed and used by instances of NLASolvers and DAESolvers (see [2]). The structure of the ESO is, therefore, central to the interface definitions which are the ultimate goal of this work.

The ESO is an abstraction representing a square or rectangular set of equations. These are the equations that define the physical behavior of the

process under consideration, and which must be solved within a flowsheeting problem. The interface to this object is intended to serve the needs of the various solver objects by allowing them to obtain information about the size and structure of the system, to adjust the values of variables occurring in it, and to compute the resulting equation residuals and, potentially, other related information (e.g. partial derivatives).

More specifically, an ESO will support a number of operations including the following:

- Obtain the current values of a specified subset of the variables.
- Alter the values of any specified subset of the variables.
- Compute the residuals of any specified subset of the equations at the current variable values.
- Obtain the partial derivatives of a specified subset of the equations with respect to a specified subset of the variables (at the current variable values of the object).

The ESO component mainly represents a rectangular system of P equations and N variables (P superior or equal to N). The ESO was built as an independent component with all the interfaces needed for the Solver Component.

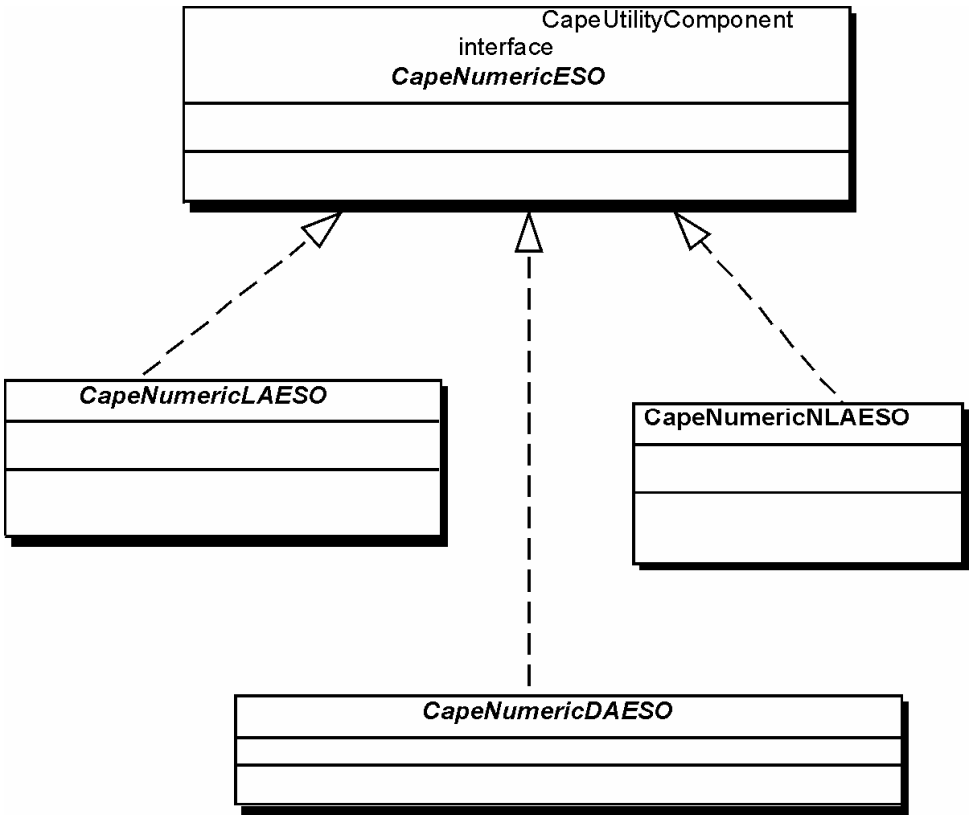

Figure 4. Class diagram of the ESO package

On figure 4 presented class diagram of the ESO package (CapeNumericESO, CapeNumericLAESO, CapeNumericNLAESO, CapeNumericDAESO classes).

The first type of classes is intended for encapsulation different model data types. The second fulfills library managing, which contains models. Below presented brief characteristic of the first type models.

CapeNumericESO (inherits from: CapeUtilityComponent)

This is the interface of the Equation Set Object which in the most general case represents a set of equations of the form f(x,x')=0. In general, a set described by an ESO can be rectangular, i.e. the number of variables does not have to be the same as the number of equations. The variables in an ESO are characterized by their current values (that can be changed via the provided interface), and also lower and upper bounds. Usually, these bounds relate to the domain of definition of the equations or physical reality. For this reason, any attempt to set one or more variables to values outside these bounds is considered to be illegal and will, therefore, be rejected.

The equations in an ESO are assumed to be sparse, i e. any given equation will involve only a subset of the variables in the ESO. Consequently, only a (usually small) subset of the partial derivatives $\partial f/\partial x$ are going to be nonzero for any set of values of the variables x. The sparsity pattern of the ESO refers to the number of such nonzero elements, and the row i (i.e. equation f_i) and column j (i.e. variable x_j) to which each such nonzero corresponds. The way in which information on this structure is defined is entirely analogous to that for linear systems.

The interface defined in this section provided mechanisms for obtaining information on the current values and bounds of the variables x, as well as the sparsity pattern of the ESO. It also allows the modification of the variable values, and the computation of the values ("residuals") of the equations f(x) for the current values of x and of the nonzero elements of the matrix $\partial f/\partial x$ (the so-called "Jacobian" matrix).

The methods of the CapeNumericESO interface are:

GetParameterList	gets the list of all the parameters
Get/SetParameter	gets/sets the current value of a specific parameter
GetNumVars	gets the number of variables of this ESO
GetNumEqns	gets the number of equations in this ESO
SetFixedVars	sets the value of fixed variables
SetAllVariables	sets the value of all variables of this ESO
Get/SetVariables	gets/sets the value of some variables
GetAllVariables	gets the value of all variables
GetVariables	gets the value of a subset of the variables
GetAllResiduals	gets the value of all the residuals
GetResiduals	gets the value of a subset of the residuals

GetJacobianStruct	returns a structure of the Jacobian matrix
GetAllJacobianValues	returns a Jacobian values
GetJacobianValues	gets the values of selected entries of the Jacobian
Destroy	deletes the ESO Component.

CapeNumericLAESO (inherits from: CapeNumericESO)

Describes linear system of equation:

$$A*x = B$$

where $x \in R^n$ solution vector, $A \in R^{n \times n}$ is a matrix of coefficients and $B \in R^n$ is a vector.

The methods of the CapeNumericLAESO interface are:

SetLHS	sets the left hand side values of the linear system
GetRHS	gets the left hand side values of the linear system
SetRHS	sets the values of the right hand side vector
GetRHS	gets the values for the right hand side vector

CapeNumericNLAESO (inherits from: CapeNumericESO)

Describes nonlinear implicit system of equation

$f(x) = 0$

where $x \in R^n$, $f: R^n \rightarrow R^n$. There is no special method right now for this interface.

CapeNumericDAESO (inherits from: CapeNumericESO)

This is the interface of the Differential-Algebraic Equation Set Object which represents a (generally rectangular) set of differential-algebraic equations of the form:

$$f(x, x', t) = 0,$$

where $t \in R$ is the independent variable, $x \in R^n$ is a vector of dependent variables and x' denotes the derivatives dx/dt. In general, the quantities dx/dt will appear in the system for only a subset of the dependent variables x. This subset of x are often referred to as the "differential variables" while the rest are the "algebraical variables".

The methods of the class CapeNumericDAESO functionality for accessing and altering information pertaining to f(x,x')=0. They also allows changing the value of the independent variable t.

The defined methods are:

SetAllDerivatives	sets the numerical value of all the derivatives
GetAllDerivatives	gets the numerical value of all the derivatives
GetDerivatives	gets the value of a subset of the derivatives

GetDiffJacobianStruct	gets the structure of the differential Jacobian matrix
GetDiffJacobianValues	gets the values of the differential Jacobian matrix
GetIndependentVar	gets the value of the independent variable
SetIndependentVar	sets the value of the independent variable.

Equation solvers

On figure 5 presented class diagram of the ESO package, which consist of classes description: CapeNumericSolver, CapeNumericLASolver, CapeNumericNLASolver, CapeNumericDAESolver.

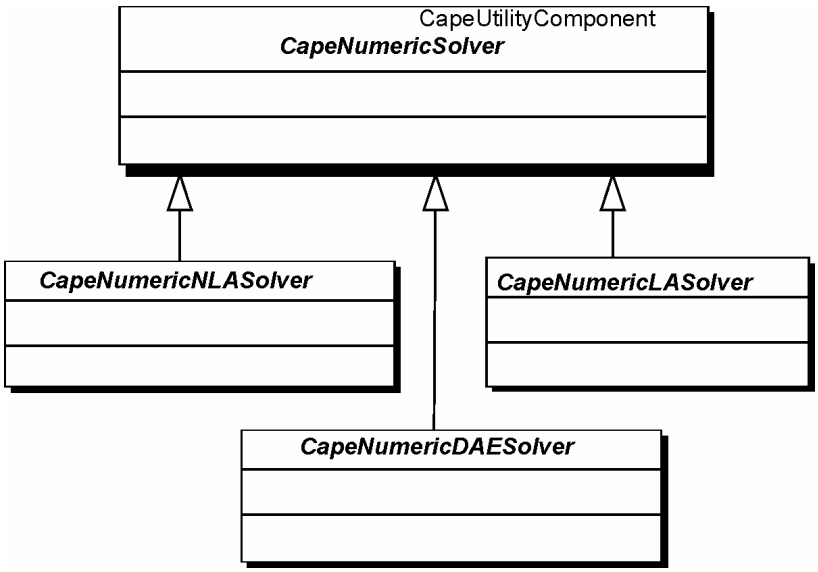

Figure 5. Class diagram of the Solver package

CapeNumericSolver (inherits from: CapeUtilityComponent)

This interface exists to provide facilities for identifying the various algorithmic parameters (e.g. convergence accuracy, integration error tolerances that are recognized by a numerical solver, and for altering their values if necessary.

Six methods have been defined, which are common to the different kinds of solvers:

GetParameterList	returns the list of all the parameters
SetParameter	sets the current value of a specific parameter
Solve	attempts to solve the system of equations
GetSolution	gets all the values of the variables
Destroy	deletes the solver component
SetReportingInterface	sets the reference to an object in charge of managing some reporting at each step of the process

CapeNumericLASolver (inherits from: CapeNumericSolver)

The Solve method get the A matrix and the B vector of the Ax=B system using the already defined methods in the CapeNumericLAESO. The A matrix is given by the GetJacobianValues method of the ESO and the B vector is equal to minus GetResiduals method with all the variables set to zero. The x vector result is given by the GetSolution method.

No specific methods have been defined for this kind of solver.

CapeNumericNLASolver (inherits from: CapeNumericSolver)

Such type of solvers allows to solve non-linear problems, which are defined by CapeNumericNLAESO, or steady-state problems which are defined by CapeNumericDAESO with all derivatives equal zero.

Methods of the class:

SetCvgTolerance	sets the convergence tolerance
GetCvgTolerance	gets information on the convergence tolerance
SetMaxIterations	sets the maximum number of iterations
GetMaxIterations	gets information on the maximum number of iterations
DoNIterations	perform N iterations on the nonlinear problem

All this functions supersedes parameters of particular solvers, which will be described later.

CapeNumericDAESolver (inherits from: CapeNumericSolver)

The class related to the solution of differential-algebraic equation systems and has additional methods:

SetRelTolerance	sets the relative tolerance values
GetRelTolerance	gets information on the relative tolerance
SetAbsTolerance	sets the absolute tolerance
GetAbsTolerance	gets information on the absolute tolerance

Below described realized nonlinear and differential-algebraic solvers. There are defined 4 classes:

- Class SdasacSolver (inherits from CapeNumericDAESO). This class based on the SDASAC solver written in FORTRAN, for more details see [3].
- Class NLEQ1Solver (inherits from CapeNumericDAESO). This class based on the NLEQ1S subroutine written in FORTRAN, which uses damped affine invariant Newton method [4,5].
- Class IDASolver (inherits from CapeNumericDAESO). This class based on the Implicit Differential Algebraic equation solver written in pure C, for more details see [6].

– Class KINSolver (inherits from CapeNumericNLAESO). This class based on the solver which uses Krylov Inexact Newton techniques and written in pure C language (see [9]).

On the figure 6 shown class diagram of the realized solver objects.

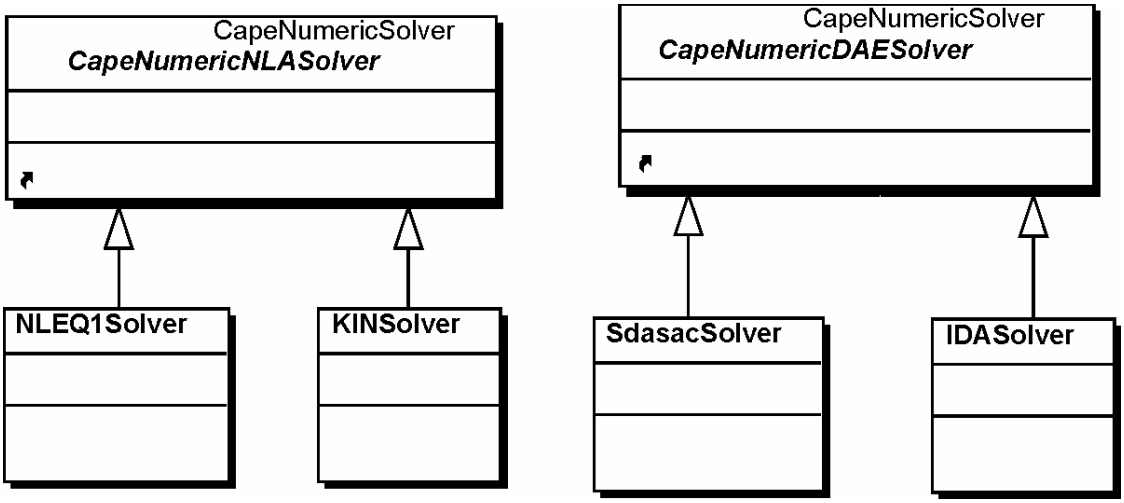

Figure 6. Sequential diagram of working with ESO and Solver classes

Conclusion

In the article described CAPE-OPEN interfaces for the different solvers. Also described models class, which definition can be generated using ProMoT with object oriented modeling conception.

The next step will be avoidance of the FORTRAN code in the models and maintaining full-featured object oriented models.

Acknowledgments

I am indebted to all researchers who have worked in the Diva project and have contributed to the new modelling system in one way or the other. Specially I would like to thank Martin Ginkel, Martin Häfele, Michael Mangold, Nikolai Tschebotarjov, Roland Waschler, Prof. Achim Kienle and Prof. Ernst D. Gilles from the Max-Planck-Institute of dynamic complex technical systems (Magdeburg, Germany), and also Andrew Chut and Konstantin Teplinsky from the Donetsk National Technical University (Ukraine).

The work was supported by PRO3 scholarship, and I would like to thank Beate Witteler-Neul, coordinator of the PRO3 project.

Also I am indebted to the computer engineering department of the Donetsk National Technical University, and head of the department Prof. Vladimir. A. Svjatnyj.

References

1. Wouwer A.V., Saucez Ph. , Schiesser W.E., Adaptive method of lines, Chapman & Hall/CRC Press, (2001), pp. 371-406.
2. Open Interface Specification Numerical Solvers, CO-NUMR-EL-03 Vers. 1
3. Mangold M., Short Dovumentation of SDASAC, Institut für Systemdynamik und Regelungstechnik Universität Stuttgart, April 3, 1998
4. Deuflhard P., Newton Techniques for Highly Nonlinear Problems - Theory and Algorithms, Academic press Inc.
5. Nowak U., Weimann L., A Family of Newton Codes for Systems of Highly Nonlinear Equations - Algorithm, Implementation, Application, ZIB, Technical Report TR 90-10 (December 1990)
6. Taylor Al. G. and Hindmarsh Al. C., User Documentation for IDA, A Differential-Algebraic Equation Solver for Sequential and Parallel Computers, Lawrence Livermore National Laboratory report UCRL-MA-136910, December 1999
7. Harwell Subroutine Library Specifications (Release 12), pp. 516-527, AEA Technology, Harwell Laboratory, December 1995.
8. Diva home page - http://www.isr.uni-stuttgart.de/diva/diva.html
9. Taylor Al. G., Hindmarsh Al. C., User Documentation for KINSOL, A Nonlinear Solver for Sequential and Parallel Computers, Lawrence Livermore National Laboratory report UCRL-ID-131185, July 1998.

Дата надходження до редколегії: 13.10.2003 р.