CAPE-OPEN

Delivering the power of component software
and open standard interfaces
in Computer-Aided Process Engineering

Open Interface Specification:

Partial Differential Algebraic Equations

Interface

www.colan.org

ARCHIVAL INFORMATION

Filename	Partial Differential Algebraic Equations Interface Specification.doc
Authors	CO-LaN consortium
Status	Public
Date	August 2003
Version	version 4
Number of pages	71
Versioning	version 4, reviewed by Jean-Pierre Belaud, August 2003
	version 3, December 2001
Additional material	
Web location	www.colan.org
Implementation specifications version	CAPE-OPENv1-0-0.idl (CORBA)
Comments	

IMPORTANT NOTICES

Disclaimer of Warranty

CO-LaN documents and publications include software in the form of *sample code.* Any such software described or provided by CO-LaN --- in whatever form --- is provided "as-is" without warranty of any kind. CO-LaN and its partners and suppliers disclaim any warranties including without limitation an implied warrant or fitness for a particular purpose. The entire risk arising out of the use or performance of any sample code --- or any other software described by the CAPE-OPEN Laboratories Network --- remains with you.

CO-LaN is a non for profit organization established under French law of 1901.

Trademark Usage

Many of the designations used by manufacturers and seller to distinguish their products are claimed as trademarks. Where those designations appear in CO-LaN publications, and the authors are aware of a trademark claim, the designations have been printed in caps or initial caps.

Microsoft, Microsoft Word, Visual Basic, Visual Basic for Applications, Internet Explorer, Windows and Windows NT are registered trademarks and ActiveX is a trademark of Microsoft Corporation.

Netscape Navigator is a registered trademark of Netscape Corporation.

Adobe Acrobat is a registered trademark of Adobe Corporation.

SUMMARY

This document aims at defining CAPE-OPEN standard interfaces for the definition and solution of Systems of Partial Differential Algebraic Equations (PDAEs). It is an extension of the existing CAPE-OPEN Numerics specification for Linear Algebraic, Non-Linear Algebraic and Differential Algebraic Equation Solvers (LAEs, NLAEs and DAEs).

First, an object model covering all necessary elements to generally describe PDAEs has been developed. Based on this the PDAESO interfaces have been developed, which provide all information required by a PDAE solver for their solution. For the solver interface itself the approach of the existing GCO Numerics specification has been used. This means, the solver can be provided with any parameters via the generic parameter list.

ACKNOWLEDGEMENTS

CONTENTS

LIST OF FIGURES

1. Introduction

The models under consideration in this document consist of Systems of Partial Differential Algebraic Equations (PDAEs).

This interface specification aims at extending the existing CAPE-OPEN (CO) Numerics specification for Differential- Algebraic Equation Systems (DAEs) towards PDAEs. In PDAEs the dependent model variables depend on one or more independent variables. Independent variables are for instance spatial coordinates, particulate coordinates (in case of population balance models) or time (in case of dynamic models). Thus, models of computational fluid dynamics are also included in this class of problems. Examples include packed bed tubular reactors, packed bed absorption and distillation columns, pipelines, etc. In other types of units, some of the properties of the material are characterised by probability density functions instead of single scalar values. Examples include crystallisation units and polymerisation reactors, in which the size of the crystals and the length of the polymer chains respectively are determined by population balances and described in terms of distribution functions. Of course, the latter may also vary with both time and spatial position.

Notice that the new requirements which extend the ones of the existing Numerics specification for DAEs can be stated quite straight forward. This is done in the following requirements chapter. Further, more general requirements are given in the CO Numerics specification. However the conceptual extensions in the interface design in order to cover PDAEs are quite complex. Therefore these are explained in detail in the Analysis and Design chapter.

2. Requirements

This chapter introduces the requirements developed by the project team. It contains a textual description followed by use cases, other diagrams and scenarios.

2.1 Textual requirements

In CO we were concerned with the solution of three different types of mathematical problem:

(i) The solution of square systems of linear algebraic equations.

(ii) The solution of square systems of nonlinear algebraic equations.

(iii) The solution of mixed square systems of ordinary differential and algebraic equations (DAEs) over time or another independent variable.

In this document the range of problems is extended to:

(iv) The solution of mixed square systems of partial differential algebraic equations (PDAEs) over *n* independent variables including time.

2.2 Use cases

2.2.1 Actors

2.2.2 List of Use Cases

Put the full list of use cases of section 2.2.4 here.

- **UC-001**: Unit defines PDAEs to be solved
- **UC-002**: Advance PDAE solution

2.2.3 Use Cases Maps

2.2.4 Use Cases

SYSTEM INITIALISATION

UC-001 UNIT DEFINES PDAES TO BE SOLVED

Actors: Flowsheet Unit
Priority: medium
Classification: System Initialisation
Context:
Pre-conditions: A PDAE system has been selected and an instance of it created and configured

Flow of events: The Flowsheet Unit identifies a subset of its equations and variables as a differential-algebraic problem, and sets up a DAE system to handle this problem during execution. It also generates or otherwise obtains initial guesses for all unknowns.

Post-conditions:

Errors:

Uses:

Extends:

SOLVER EXECUTION

UC-002 ADVANCE PDAE SOLUTION

Actors: DAE Solver

Priority: medium

Classification: Solver Execution

Context:

Pre-conditions:

Flow of events: The PDAE Solver carries out steps in the independent variables. It interacts with the global differential-algebraic equation and variable sets as follows :

(v) it changes the variable values

(vi) it requests residual values corresponding to the latest variable values it has supplied

(vii) it requests Jacobian information of the equation Solvers

The termination condition will be provided by the Simulator Executive. It will consist either of an explicit target value of the independent variable, or a condition on a particular variable value.

The numerical method will involve an iterative procedure.

Note : this use case is written only for EO simulators

Post-conditions:

Errors:

Uses:

Extends:

2.3 Sequence diagrams

At the requirements stage no Sequence diagrams are required.

3. Analysis and Design

3.1 Overview

3.1.1 Description of Partial Differential Algebraic Equations (Analysis)

In the following the concepts used to represent distributed Solvers are presented.

3.1.1..1 Domain

Prior to establishing any equation, it needs to be clear which set of domains (for instance spatial directions or particle size distributions) is to be involved. These domains define an independent variable by specifying its name (e.g.'AXIAL') and a lower and upper bound (e.g. 0..1). All domains are continuous, i.e. a closed domain that consists of a subset of R. Discrete domains are not considered as such, because they can be dealt with sufficiently by simply introducing further variables.

Once the domains are established, it is possible to devise variables that are distributed on these domains.

3.1.1..2 Variable

Somewhat different to the (independent) variables, which are described by domains, are the dependent variables. They denote a state of the equation under consideration. A variable is identified by a name, a lower and upper bound for its value and a preset value. Most important for the purpose of the document is the distribution of the variable that is represented by a number of associated domains. The variable is interpreted as being distributed over the Cartesian product of all associated domains. The assumption limits the scope of the object model to simple geometries, but avoids for example the need to describe geometries of a Solver.

Due to its distribution, a variable will have a separate value for each grid point.

3.1.1..3 Equation validity domains

An equation constrains the values of occurring variables. As such, it is defined for the whole interior part of a domain, and can be extended to the boundary as well. Thus, equation validity domains are a subset of variable domains. The domain boundaries of an equation can either be of type CLOSED or OPEN.

In case domain boundaries are *open*, boundary conditions have to be specified at the corresponding boundaries. These in turn are treated as equations placed on point domains, i.e. domains with the same lower and upper bound and only one domain boundary open.

3.1.1..4 Projected Domains

Projected Domains are required if an equation involves dependent variables with one or more of the independent variables (or time) fixed at a specific value, for instance $\underline{x}_P(z_1, z_2, z_3, z_4) = \underline{x}_P(z_1, z_2, 0, z_4)$. All dependent variables $\underline{x}$ other than $\underline{x}_P$ are assumed to be always distributed over the full extent of associated domains, i.e. no domain with $\underline{LB} \leq \underline{z} \leq \underline{UB}$ with fixed at one single value.

3.1.1..5 Equations

Having defined domains, variables and equation validity domains, it is now possible to pose equations involving defined variables on defined equation validity domains. Because the goal of this work is to develop standards for engineering applications mainly involving balance equations, and thus, considering convective,

diffusive and source terms, only, the Partial Differential Algebraic Equation (PDAE) under consideration has been assumed to be of the following quite general, transparent and flexible superstructure.

$$\underline{F}\left[\frac{\partial}{\partial t}\underline{f}(\underline{x},\underline{x}^{P},\underline{z},t),\frac{\partial}{\partial \underline{z}}\underline{g}\left(\frac{\partial \underline{x}}{\partial \underline{z}},\underline{x},\underline{x}^{P},\underline{z},t\right),\underline{k}(\underline{x},\underline{x}^{P},\underline{z},t)\right]=\underline{0} \qquad \forall \underline{L}<\underline{x},x^{P}<\underline{U}$$

The symbols have the following meaning:

$\underline{F}$ is the vector of equations making up the overall PDAEs,

$\underline{f}$ is the convective term,

$\underline{g}$ is the diffusive term,

$\underline{k}$ is the source term,

$\underline{x}$ is the vector of state variables,

$\underline{x}^{P}$ is the vector of state variables on projected domains[1],

t is the independent variable time,

$\underline{z}$ is the vector of independent variables (other than time),

$\underline{f}$, $\underline{g}$, $\underline{k}$ are function of the state variables $\underline{x}$ and $\underline{x}_{\underline{z}}$ and the independent variables $\underline{z}$ and t.

The terms $\frac{\partial \underline{x}}{\partial \underline{z}}$ denote the partial derivatives of $\underline{x}$ with respect to independent variables,

The terms $\frac{\partial}{\partial t}\underline{f}(\underline{x},\underline{z},t)$ denote the partial derivatives of $\underline{f}$ with respect to time,

The terms $\frac{\partial}{\partial \underline{z}}\underline{g}\left(\frac{\partial \underline{x}}{\partial \underline{z}},\underline{x},\underline{z},t\right)$ denote the partial derivatives of $\underline{g}$ with respect to independent variables,

$\underline{L}$ is the vector of lower bounds of the (open or closed) interval on which an equation is defined, and

$\underline{U}$ is the vector of upper bounds of the (open or closed) interval on which an equation is defined.

The formulation does not make any assumptions on the structure of the equation Solver, i.e. the linear as well as the nonlinear case are supported. However, restriction is to explicitly include derivatives of order two or less. Higher order derivatives, though, can always be included using dummy derivatives. Further, apart from transparency reasons, the reason for explicitly introducing time as a distinct domain is that there is exactly one time for the overall equation Solver which is the same for all equations. Spatial or particulate coordinates on the other hand are usually accounted for separately in distinct process models.

The independent variables $\underline{z}$ occurring in the Solver formulation are subsequently called *domains*, each defined by a name, a lower and an upper bound. Equation validity domains are established using domains plus additional information on whether they are regarded to have closed or open boundaries on its two ends.

---

[1] Projected domains are necessary if equations involve dependent variables with one or more of the independent variables (or time) fixed, e.g. $\underline{x}(z)=\underline{x}(0)$.

Each of the variables $\underline{x}$ occurring in the PDAEs can (but do not have to be) distributed on any number of domains and are then interpreted as a function of the independent variables (cf. Hackenberg, et al., 2000).

3.1.1..6 The Partial Differential Algebraic Equation Set Object

The Partial Differential Algebraic Equation Set Object (PDAESO) concept defines a partial differential algebraic equation system from a mathematical/numerical point of view. It defines a number of domains, variables, equation validity domains and equations, which are denoted with *PDAESODomain, PDAESOVariable, PDAESOEqDomain,* and *PDAESOEquation,* respectively. Note that besides the problem-specific independent variables defined, a domain time is always present in the PDAESO.

Based on this information, it is assumed that the PDAE Solver under consideration is of the general form presented above.

The following object model explains the relationships of the concepts introduced:

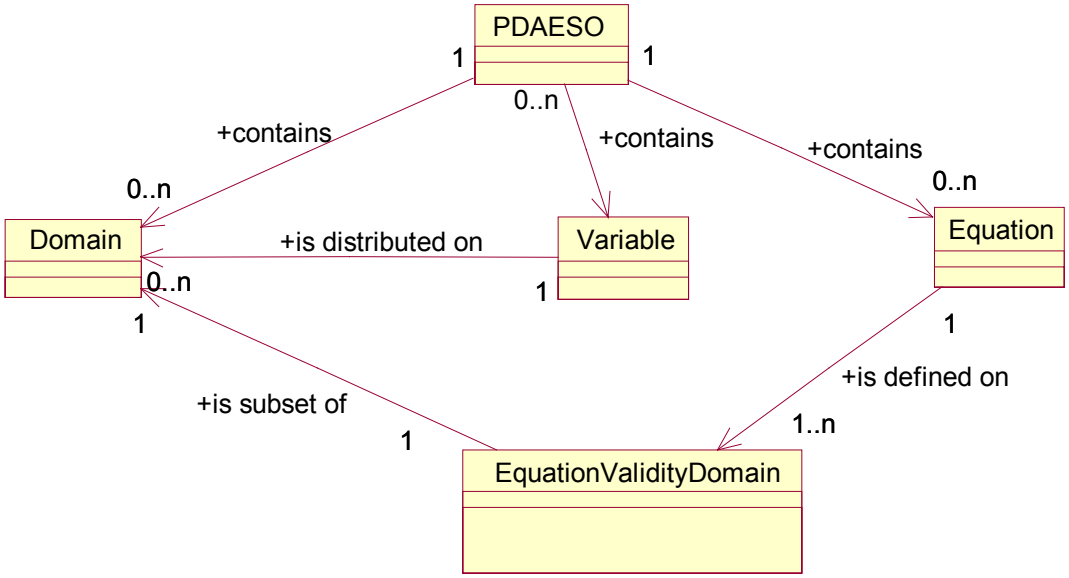


Figure 1 (High-level) Object model for partial differential algebraic equation Solvers

3.1.2 Design

3.1.2..1 Relation to previously defined interfaces

In CAPE-Open, the interfaces for the solution of linear algebraic (LA), non-linear algebraic (NLA) and differential algebraic (DA) systems have been specified. There the interface architecture is based on three main concepts, called Equation Set Object (ESO), Solver and Solver Manager.

The ESOs represent models, which are defined as a set of algebraic equations in case of steady state models and as a mixed set of differential and algebraic equations in case of dynamic models. The interface to this object provides the information required by solver objects.

The Solver is concerned with the solution of the ESO equations and the Solver Manager with the creation of instances of Solvers. Conceptually the Solver contains both the ESO and the numerical algorithms for the solution. On the interface level this is reflected as follows. The interface to the solver gives access to the

solver object, i.e. allows its configuration and the invocation of the "solve" command. Therefore the solver contains the numerical algorithm.

Further, a solver is created by the solver manager for a specific ESO. On the interface level this is reflected by a method of the solver manager interface "CreateSolver", which creates an instance of a solver. A parameter, which has to be specified as input to this method, is the address of the ESO for which the solver is to be created. Therefore the solver contains as well the ESO (via its address). If the "solve" method is invoked on the solver, the numerical algorithm will use the ESO address to obtain the necessary information.

In the present document the interfaces are specified which extend the framework outlined above to the solution of partial differential algebraic equations.

3.1.2..2 Solvers

In the CAPE-OPEN project, the following classes of solver objects had been introduced and defined. They contain both the data that characterise the mathematical problem *and* the numerical algorithm that solves this problem.

- The Linear Algebraic Solver(*LASolver*) object.
- The Nonlinear Algebraic Solver (*NLASolver*) object.
- The Differential-Algebraic Equation Solver (*DAESolver*) object.

Here one additional class of Solver object is introduced to consider the problem of PDAES.

- The Partial-Differential-Algebraic Equation Solver (*PDAESolver*) object.

In the rest of this document, we will generically refer to these objects as "*Solvers*".

3.1.2..3 Solver Manager

Solver Manager objects are used to create instances of the corresponding Solver using information that defines the structure of each such instance (i.e. ESO address and algorithmic parameters of solver object).

Recall, that in the CAPE-OPEN project, the following *Manager* classes were introduced:

(i) The Linear Algebraic Solver Manager (*LASolverManager*).

(ii) The Nonlinear Algebraic Solver Manager (*NLASolverManager*).

(iii) The Differential-Algebraic Equation Solver Manager (*DAESolverManager*).

Here we introduce the SolverManager for PDAESolvers:

(iv) The Partial-Differential-Algebraic Equation Solver Manager (*PDAESolverManager*).

3.1.2..4 The Equation Set Object

In Keeping and Pantelides (1999), the concept of an *Equation Set Object* (ESO) has been introduced as means of defining information concerning large sets of nonlinear equations of any kind in a way that can be accessed and used by instances of NLASolvers and DAESolvers.

The Equation Set Object is an abstraction representing a square or rectangular set of equations. These are the equations that define the physical behaviour of the process[2] 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 Solver, 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). Hence, this interface requires standardisation as part of CAPE-OPEN. However, the *construction* of such an object is a proprietary matter for individual vendors of flowsheeting packages and is *not* standardised as part of CAPE-OPEN.

More specifically, an ESO supports 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.
- Get the structure[3] of the sparse matrix representing the partial derivatives of a specified subset of the equations with respect to a specified subset of the variables.
- Compute the residuals of any specified subset of the equations at the current variable values.
- Get a sparse matrix containing the values of the partial derivatives of a specified subset of the equations with respect to a specified subset of the variables (at the object's current variable values).

A more complete definition is given in Keeping and Pantelides (1999).

The information associated with an ESO differs depending on whether the set of equations being described is purely algebraic (as is the case with the NLASolver class mentioned above), mixed differential and algebraic (as in the case of DAESolver) or mixed, partial differential and algebraic (as in the case of PDAESolver). For this reason, Keeping and Pantelides (1999) introduce a hierarchy of ESOs, which comprises two classes:

(i) Class *AlgebraicESO* defines a purely algebraic set of equations.

(ii) Class *DifferentialAlgebraicESO* inherits from class AlgebraicESO and refines it to define a mixed set of differential and algebraic equations.

In the present specification we introduce the ESO which defines a mixed set of differential algebraic equations:

(iii) Class *PartialDifferentialAlgebraicESO.*

3.1.2..5 The Partial Differential Algebraic Equation Set Object

As introduced above, a PDAESO contains a number of domains, variables, equation validity domains and equations. The equations are of the following form,

$$\underline{F}\left[\underbrace{\frac{\partial}{\partial t}\underline{f}(\underline{x},\underline{x}^P,\underline{z},t)}_{\underline{w}^f},\underbrace{\frac{\partial}{\partial \underline{z}}\underline{g}\left(\frac{\partial \underline{x}}{\partial \underline{z}},\underline{x},\underline{x}^P,\underline{z},t\right)}_{\underline{w}^g},\underbrace{\underline{k}(\underline{x},\underline{x}^P,\underline{z},t)}_{\underline{w}^k}\right]=\underline{0}$$

[2] Here, the term "process" may mean the entire plant being modelled, a plant section or, indeed, a single unit operation or part thereof.

[3] *i.e.* a list of the partial derivatives which will *not* be identically zero for all values of the variables.

where $\underline{w}^f$, $\underline{w}^g$ and $\underline{w}^k$ denote the terms of the system as given in the formula.

In order to treat this Solver effectively in the concept of an *Equation Set Object* (ESO), we distinguish between an *inner* ESO and an *outer* ESO.

The inner ESO is concerned with the evaluation of equations $\underline{f}$, $\underline{g}$, $\underline{k}$ as a function of the state variables $\underline{x}$ and $\underline{x}_z$ (the latter being the derivative of x with respect to z, which is derived by the PDAE solver by means of discretisation) and the independent variables $\underline{z}$ and t. In fact, due to the distributed nature of the problem, the inner ESO is evaluated at *a certain point* (given by the independent variables) on the grid. As such, it will have to be evaluated a number of times to cover all possible grid points.

In order to treat the Solver using an Equation Set Object, the equations and variables need to be placed into one contiguous vector each. This leads to the introduction of $\underline{E}$, the set of functions $\underline{f}$,$\underline{g}$,$\underline{k}$ and $\underline{v}$, the set of variables $\underline{x}$, $\underline{x}^P$, $\underline{x}_z$, $\underline{z}$, t,, where $\underline{x}_z = \dfrac{\partial x}{\partial z}$

$$\underline{E} \equiv \begin{pmatrix} \underline{f} \\ \underline{g} \\ \underline{k} \end{pmatrix}$$

$\underline{E}$ is called the vector of inner equations and is of dimension NIE:

$$NIE = dim(\underline{f}) + dim(\underline{g}) + dim(\underline{k}).$$

The variables $\underline{x}$, $\underline{x}^P$, $\underline{x}_z$, $\underline{z}$, t are mapped into the vector $\underline{v}$:

$$\underline{v} \equiv \begin{pmatrix} \underline{x} \\ \underline{x}^P \\ \underline{x}_z \\ \underline{z} \\ t \end{pmatrix}$$

These variables are called inner variables $v_1,..,v_{NIV}$, where

$$NIV = dim(\underline{x}) + dim(\underline{x}^P) + dim(\underline{x}_z) + dim(\underline{z}) + 1.$$

Altogether, they constitute the inner ESO called innerESO, which is of type ICapeNumericESO.

$$\underline{E}(\underline{v}) \equiv \begin{pmatrix} \underline{f} \\ \underline{g} \\ \underline{k} \end{pmatrix} \left(\underline{x}, \underline{x}^P, \underline{x}_z, \underline{z}, t \right)$$

In order to do a proper mapping, interfaces are defined which allow the unambiguous distinction of inner variables by their kind. The inner variable can be one of five kinds:

- Kind: **IDVariable** ($\underline{x}$)

The inner variable is a dependent variable, and the InnerDependentVariable interface provides means to access the corresponding PDAESOVariable.

- Kind: **IDPVariable** ($\underline{x}^P$)

The inner variable is a dependent variable on a projected domain, and the InnerDependentProjectedVariable interface provides means to access the corresponding PDAESOVariable as well as the information required due to the projection of domains.

- Kind: **IDDVariable** ($\underline{x}_{\underline{z}}$)

The inner variable is a partial differentiated dependent variable, and the InnerDependentVariableDerivative interface provides means to access the corresponding PDAESOVariable as well as the corresponding PDAESODomain.

- Kind: **IIVariable** ($\underline{z}$)

The inner variable is an independent variable (other than time), and the InnerIndependentVariable interface provides means to access the corresponding PDAESODomain.

- Kind: **ITVariable** *(t)*

The inner variable is the somewhat special independent variable time.

Apart from the information about the kind, the InnerVariable interface also has to provide means to access the index of the corresponding PDAESOVariable or PDAESODomain in their corresponding lists. This assigns an inner variable used in the innerESO to exactly one PDAESOVariable or PDAESODomain as used by the PDAE solver and establishes the required link.

The outer Solver is concerned with the evaluation of the overall equations $\underline{F}$ as a function of the expressions $\underline{w}$, which previously have been introduced and distinguished by $\underline{w}^f$, $\underline{w}^g$ and $\underline{w}^k$ for transparency reasons. Subsequently, we call this the outer ESO.

The outer ESO is an Equation Set Object (ICapeNumericESO) on the set of outer functions $\underline{F}$.

$$\underline{F}(\underline{w}^f, \underline{w}^g, \underline{w}^k) = \underline{F}(\underline{w}) \equiv 0$$

where $\underline{w}$ is called the vector of outer variables, $w_1,..,w_{NOV}$,

$$NOV = dim(\underline{w}^f) + dim(\underline{w}^g) + dim(\mathrm{w}^k).$$

$$\underline{w} \equiv \begin{pmatrix} \underline{w}^f \\ \underline{w}^g \\ \underline{w}^k \end{pmatrix}$$

From the outer variables, $\underline{w}^f$ and $\underline{w}^g$ denote the partial derivatives of $\underline{f}$ with respect to t and $\underline{g}$ with respect to $\underline{z}$, respectively. The term $\underline{w}^k$ is equal to $\underline{k}$ and summarises all (nonlinear) terms, which cannot be classed into one of the other groups.

For instance, $\underline{w}^f = \dfrac{\partial}{\partial t}\underline{f}(\underline{x}, \underline{x}^P, \underline{z}, t)$.

The outer variables can be obtained from the inner equations for instance by discretisation and/or quadrature formulae. This is the task of each PDAE Solver internally. Some ways of solving it will be discussed in Section 6.2.

The following figure presents the concepts introduced above.

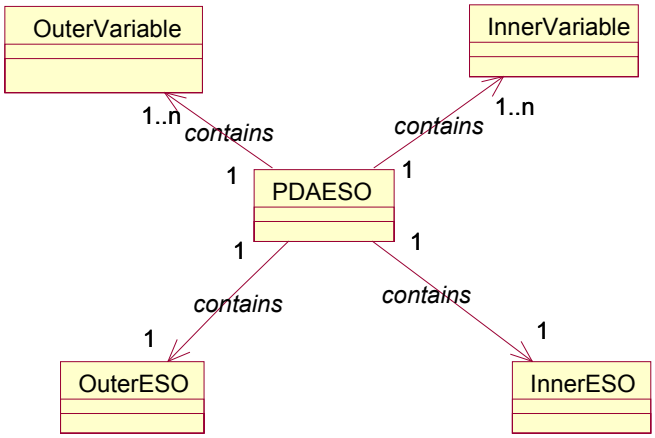


Figure 2 Concepts

3.2 Sequence diagrams

This section lists the sequence diagrams. To facilitate a better understanding, in the present case they are given in algorithmic form. The solution process is described using the interfaces presented in Section 3.3 and defined in Chapter 3.6.

3.2.1 Solution Process

In the following, it is explained how a solver could use the PDAESO to solve the problem contained in it. After a very brief introduction to the set-up, the step AdvanceSolution is considered in great detail, in order to build up the understanding step-by-step. For this purpose, it turned out to be advantageous to consider a simple DAE case initially, before introducing further concepts and steps to deal with the more complex case of PDAE systems appropriately.

3.2.1..1 Set-Up

Assume a particular model with domains, variables, equation validity domains and equations (incl. boundary conditions) to be given. Now, the solver can be set up using the following calls to the PDAESO:

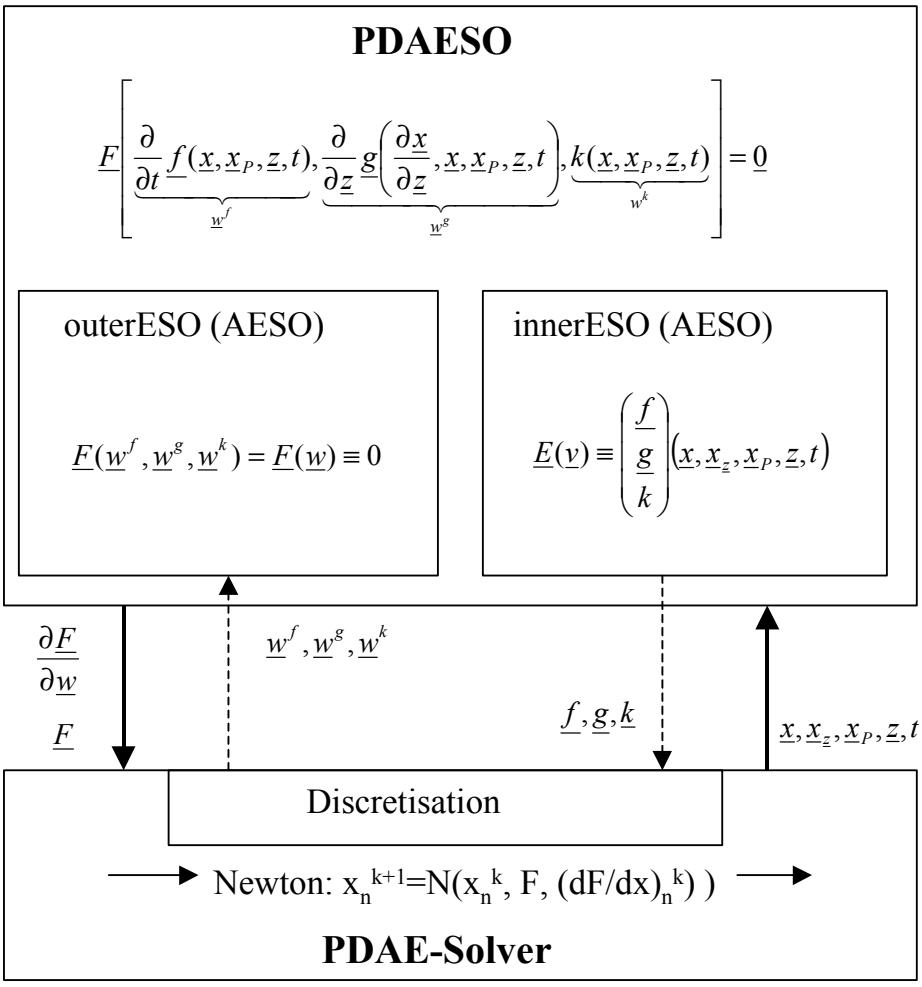
- PDAESO -> GetDomains
- PDAESO -> GetVariables
- PDAESO -> GetEquations

These three calls return lists of interfaces, one each for PDAESODomain, PDAESOVariable and PDAESOEquation. Interfaces are established independently of the actual distribution of a variable over a (number of) domain(s), i.e. any kind of discretisation is done internally in the solver and as such, unimportant for the interface specification.

3.2.1..2 Advance Solution – DAE case

Let us assume at this stage that we are dealing with a DAE Solver, i.e. outer variables $\underline{w}^g$ as appearing in Figure 3.2 are assumed to be zero, and differentials $\frac{\partial \underline{x}}{\partial \underline{z}}$ to be non-existing.

Overall, we are faced with two nested loops or iterations. One to proceed in time and one to decrease the residuals of the equations within that particular time step (close enough) to zero.



For the moment we assume we want to integrate the DAE Solver from an already established time step t_{n-1} to time step t_n. This means we want to receive the values of all inner variables v at time t_n, i.e. v_n. Further we assume that we use a Newton Algorithm in order to receive a good approximation for v_n. Let's denote the value of v in the Newton iteration step k within time step t_n as v_n^k. Lets assume that the value of v_n^k has been established and now we want to get the next, better approximation v_n^{k+1}.

To summarise, the solver knows:

- v_n^k, the value of the last iteration step within the present time step
- s, where s is an array containing the values of vector v at the previous time steps, i.e. $s(t_{n-1}, t_{n-2},....) = (v_{n-1}, v_{n-2}, ...)$.

Our goal is to calculate the values of the next iteration step *k+1* of *v* by using a Newton type algorithm:

$v_n^{k+1} = v_n^k - (F|_{t=tn,\ iteration\ step\ k} * (dF/dv)^{-1}|_{t=tn,\ iteration\ step\ k})$.

Remember that v_n^k is known from the previous iteration step. The values for $F|_{t=tn,\ iteration\ step\ k}$ and $(dF/dv)^{-1}|_{t=tn,\ iteration\ step\ k}$ will have to be calculated by using the inner and outer ESO.

Calculating $F|_{t=tn,\ iteration\ step\ k}$ means to evaluate $F(w_n^k)$. For this purpose w_n^k has to be constructed. This can be done as follows. First, the addresses of the outer variable interfaces have to be received by the solver. In the following the calls of the solver to the PDAESO interfaces will be written in a pseudo algorithmic way and in courier new font.

```
// get references to outer variable interfaces
OuterVariableRef := PDAESO -> GetOuterVariables
```

Then the solver has to ask the PDAESO for the outerESO in order to ask it for the number of outerVariables.

```
// receive reference of outer ESO
OuterESO := PDAESO -> GetOuterESO

// receive number of variables w in outer ESO
NOV := OuterESO -> GetNumVars
```

Now the outerVariable array w_n^k has to be constructed in a way which depends on the type of the outer variable. In the time dependent DAE case this means simply constructing the time derivatives df/t and receiving the source terms k (see figure). This is written down in a pseudo algorithmic way below in Procedure 1.

```
// Procedure 1: "Construct w_n^k and receive corresp. Residuals F(w_n^k)"
// from outer ESO,
// v_n^k is the result of the last Newton iteration within the
// calculation of time step t_n

// go through all outer variables
FOR i:=1 TO NOV DO

OuterVariableKind:=OuterVariableRef(i)->GetKind

IF OuterVariableKind="fFunction" THEN
// case partial derivative wrt time
// w(i)= δf_j/δt = δE(i)/δt

// This is an example of how the derivative could be constructed,
// see how v_n^k is constructed below in Procedure 2, v_n is the known
// result from the previous time step t_n.
      w(i)_{n+1}^k= (E(i, v_n^k) – E(i, v_{n-1})) / (t_n-t_{n-1})

ELSEIF OuterVariableKind="kFunction" THEN
// case source term, no changes required to Residuals E
     w(i)_n^k= E(i, v_n^k)

END
```

Notice that the approach shown above to approximate the derivative df/dt in the solver can be improved by using the Jacobian of the InnerESO as follows:

$$\frac{\delta f(x(t))}{\partial t} = \frac{\partial f(x(t))}{\partial x} * \frac{dx(t)}{dt} \quad ,$$

where the first factor of the right hand side df/dx is an element of the Jacibian provided by the InnerESO and the second factor dx/dt would have to be approximated by the solver.

Finally the outerVariable values can be set and the residuals received.

```
OuterESO -> SetVariableValues(w_n^k)
// the 0 as argument of GetResiduals denotes that all residuals are requested.
F(w_n^k):=OuterESO -> GetResiduals(0)
```

However, in the above procedure the residuals of the innerESO E are required for $v=v_n^k$ and $v=v_{n-1}^k$, i.e. the calls

```
InnerESO -> SetVariableValues(v_n-1)

E(i, v_n-1):=InnerESO -> GetResiduals(i)
```

and

```
InnerESO -> SetVariableValues(v_n^k)

E(i, v_n^k):=InnerESO -> GetResiduals(i)
```

have to be issued to the InnerESO. Notice, that v_{n-1} is known from the previous time step, and v_n^k is the value of the last iteration step within time step t_n.

Now the Jacobian $(dF/dv)\,|_{t=tn,\ iteration\ step\ k}$ has to be constructed. This could be done for example as follows:

$(dF/dv)\,|_{t=tn,\ iteration\ step\ k} = (F(w_n^k) - F(w_n^{k-1}) / (v_n^k - v_n^{k-1})$

where w_n^k is constructed as shown in Procedure 1, w_n^{k-1} will be known from the last iteration step and v_n^k, v_n^{k-1} are the values of the last two iteration steps within time step t_n . However in terms of computation cost and numerical stability this may not be a satisfactory approach.

Alternatively, if the OuterESO provides all Jacobian entries, $(dF/dw)\,|_{t=tn,\ iteration\ step\ k}$ can be obtained from the OuterESO as follows:

$(dF/dw)\,|_{t=tn,\ iteration\ stepk}$ = OuterESO->GetJacobianValues

The Jacobian $(dF/dv)\,|_{t=tn,\ iteration\ step\ k}$ could be obtained by evaluating $(dw/dv)\,|_{t=tn,\ iteration\ step\ k}$ (as described for an example in the Appendix xxx) and

$(dF/dv)\,|_{t=tn,\ iteration\ step\ k} = (dF/dw)\,|_{t=tn,\ iteration\ step\ k} * (dw/dv)\,|_{t=tn,\ iteration\ step\ k}$.

Now all information is available to perform the next Newton step as described above.

So far we assumed that we know the value of v of the previous iteration step k-1, v_n^{k-1}. However for the first iteration (k=0) of a new time step n we have to provide the first guess for v, v_n^0 which includes to set the time $t = t_n$. This is described for the DAE case below in Procedure 2.

Before that we have to ask the PDAESO for the InnerVariables, the InnerESO itself and the number of its variables NIV.

```
InnerVariableRef := PDAESO->GetInnerVariables
InnerESO:= PDAESO->GetInnerESO
NIV := InnerESO->GetNumVars

// Procedure 2: Construct inner ESO variable vector v_n^0.

FOR i:=1 TO NIV DO

InnerVariableKind:=InnerVariableRef(i)->GetKind

IF InnerVariableKind="IDVARIABLE" THEN
// case InnerDependentVariable
v(i)_n^0 := guess from(s,(ds/dt))

ELSEIF InnerVariableKind="IIVariable" THEN
// case InnerIndependentVariable: do nothing

ELSEIF InnerVariableKind="ITVariable" THEN
// case InnerTimeVariable
v(i)_n^0 := t_n

END
```

The corresponding Residuals E_n^0 and the Jacobian can be obtained by setting the variable values

```
InnerESO -> SetVariableValues(v_n^0)
```

And make the following calls

```
E(v_n^0) :=InnerESO -> GetResiduals

(dE/dv) |t=tn, iteration step 0 = InnerESO->GetJacobianValues.
```

3.2.1..3 Advance Solution – PDAE case

Having understood the DAE case, we are now ready to proceed with the PDAE case. Once more, we are faced with two nested loops or iterations. One to proceed in time (called time stepping) and one to decrease the residuals of the equations within that particular time step (close enough) to zero (called residual iteration). The overall process is once more demonstrated in Figure 3.2.

Assume that we have just finished to determine the dependent variables at time step t_{n-1} and are now looking to proceed to time step t_n. Without going into too much detail here, it can be assumed that the solver has been able to provide guesses (for instance, using a Newton-type algorithm) for the dependent variables $\underline{x}$ at time t_n.

However, this estimate most likely is not going to be good enough to satisfy all equations $\underline{F}$ straight away without need for further modification. Thus, we need to iterate the values of $\underline{x}$ within time step t_n using another Newton-type iteration. In order to distinguish time stepping from residual iteration, we denote a residual iteration by superscript k and time stepping by subscript n. Thus, $\underline{x}_n^k$ indicates the dependent variable vector $\underline{x}$ at residual iteration step k within time step n.

The distinction of variables into dependent, independent, projected and so on arises from their physical meaning. The solver, however, simply works on **one** long array comprising **all** variables regardless of their physical meaning, called the inner variable vector $\underline{v}$. Thus, if we say that the solver provides guesses for $\underline{x}_n^k$, it is rather particular entries of the vector $\underline{v}$ that are known.

Note, that there is one vector of inner variables for **each** point on the grids for $\underline{z}$ and t, i.e. more appropriately, it should be $\underline{v}(\underline{z},t)$, even though the values of $\underline{z}$ and t are also stored inside the vector itself. Due to the design of the innerESO, which can handle only **one** vector v at a time and not an array of vectors $\underline{v}$ (i.e. a tensor) for all $\underline{z}$ at a given t_n, it is once more the task of the PDAE solver to keep track of all vectors $\underline{v}$

corresponding to a particular $\underline{z}$ and t. This is achieved by introducing the tensor *IVV(i,*$\underline{z}$*,t)*, which stores all inner variable vector entries i for each value of independent variables $\underline{z}$ and time t..

At this stage, the assembly of the inner variable vector needs to be considered in more detail. Apart from the dependent variables $\underline{x}$ and independent variables $\underline{z}$ and t, the inner variable vector consists further of the projected dependent variables $\underline{x}_P$ and the dependent variable derivative(s) $\frac{\partial x}{\partial z}$. Using the mapping methods made available through the PDAESO concept, the projected dependent variables $\underline{x}_P$ can very easily be obtained by copying the values from dependent variables $\underline{x}$ at the fixed independent variable location and for all unspecified independent variables. The way the value for the fixed independent variable location (i.e. the projected domain) is obtained will become apparent in the pseudo code presented at the end of this general overview.

The dependent variable derivative $\frac{\partial x}{\partial z}$ is the partial derivative of a particular dependent variable x with respect to a particular independent variable z and has to be obtained via discretisation. Using the mapping information once more, the position of the corresponding dependent variable x in the inner variable vector $\underline{v}$, e.g. j, can be obtained. Further, out of the independent variables $\underline{z}$, the direction of differentiation, z_{DoD}, can be obtained. This information is sufficient to establish the partial derivative by using for instance central finite differences of spacing Δz_{DoD}:

$$\frac{\partial x}{\partial z}(\underline{z},t) = \frac{IVV(j,z_1,z_2,..,z_{DoD}+\Delta z_{DoD},..,t) - IVV(j,z_1,z_2,..,z_{DoD}-\Delta z_{DoD},..,t)}{2\Delta z_{DoD}}$$

Overall, this gives the vector of inner variables at time step t_n and residual iteration step k, $\underline{v}_n^k$ (internally stored as $IVV(i,z,t)_n^k$ by the solver). Providing the innerESO with $\underline{v}_n^k$ enables us to invoke the *solve* command and returns the values of the inner equation vector $\underline{E}_n^k$, which contains the list of $\underline{f}$, $\underline{g}$ and $\underline{k}$, for all $\underline{z}$ and t.

The dependence of $\underline{E}$ on $\underline{z}$ and t once more needs to be handled appropriately by the PDAE solver, e.g. by introducing a tensor *IEV(i,*$\underline{z}$*,t)*, which stores *E(i)* for each value of $\underline{z}$ and t.

So far, we have dealt with exclusively the inner Solver. However, it is only the outer Solver that provides a direct indication of how well the overall equations $\underline{F}$ are satisfied.

The outer Solver is given by the outerESO and requires the outer variable vector $\underline{w}$ to be set, before the *solve* command can be invoked on the outerESO in order to obtain $\underline{F}$. Each entry in the outer variable vector $\underline{w}$ is either $\underline{w}^f$, $\underline{w}^g$ or $\underline{w}^k$, i.e. the partial derivatives of $\underline{f}$ with respect to t, the partial derivatives of $\underline{g}$ with respect to $\underline{z}$ or plain $\underline{k}$, respectively. Note, that because the outerESO (like the innerESO) can evaluate its equations only for one particular variable vector at a time, there is **one** $\underline{w}$ and **one** $\underline{F}$ for **each** value of independent variables $\underline{z}$ and t. The outer variable vector $\underline{w}$ and outer equation vector $\underline{F}$ is different for each value of independent variables $\underline{z}$ and t. Therefore the tensor *OVV(i,*$\underline{z}$*,t)*, which stores all outer variable vector entries i for each value of independent variables $\underline{z}$ and time t, is introduced. Accordingly, *OEV(i,*$\underline{z}$*,t)* stores all outer equation vector entries i for each value of independent variables $\underline{z}$ and time t.

The outer variable interface provides means to access mapping information required to obtain the position j of the equation under consideration in the inner equation vector $\underline{E}$, and, if required, the direction of discretisation, i.e. either t or independent variable z_{DoD}.

Contrary to the outer variable w^k, which is simply equal to k, the creation of w^f and w^g requires a discretisation step. In the latter case, it is possible to proceed as in the discretisation described above for the inner dependent variable derivative.

If the outer variable is discretised with respect to z_{DoD}, a PDAE solver might want to use forward differentiation on step size Δz_{DoD}, for instance. This requires the value of g at multiple values of independent variable z_{DoD}, which can be accessed readily via tensor *IEV(j,*$\underline{z}$*,t)*.

$$w^g = \frac{\partial g}{\partial z_{DoD}} = \frac{IEV(j, z_1, z_2, .., z_{DoD} + \Delta z_{DoD}, .., t) - IEV(j, z_1, z_2, .., z_{DoD}, .., t)}{\Delta z_{DoD}}$$

If the outer variable is discretised with respect to time *t*, the values of inner equations at previous time steps is required. This brings us back to remember the two loops mentioned initially. It is the time stepping loop that has computed the required values of *IEV(z,t)* at previous time steps t_{n-1}, t_{n-2}, …, and information is readily available. Assume a PDAE solver might want to use backward differentiation. The time step is given by $\Delta t_n = t_n - t_{n-1}$.

$$w^f = \frac{\partial f}{\partial t} = \frac{IEV(j, \underline{z}, t_n) - IEV(j, \underline{z}, t_{n-1})}{\Delta t_n}$$

As already mentioned, the outer variable *k* is equal to the value of the corresponding equation *j* in the inner equation vector.

$$w^k = IEV(j, \underline{z}, t)$$

Overall, the vector $\underline{w}$ of outer variables can be established by proper insertion of $\underline{w}^f$, $\underline{w}^g$ and $\underline{w}^k$. Once $\underline{w}$ is established, it can be set in the outerESO and the *solve* command invoked. This returns the values of outer equations $\underline{F}$.

For reasons mentioned earlier, this whole procedure on the outer ESO needs to be repeated for each value of independent variable $\underline{z}$ and $\underline{t}$, in order to create *OVV* and *OEV,* which exist for all values of $\underline{z}$ and *t*.

Taking a step back, what have we achieved so far? Basically, we are now able to calculate the vector of outer variables for all $\underline{z}$ and *t* given values of dependent variables $\underline{x}$ for all $\underline{z}$ and *t*.

This is essential for the residual iteration. Here, we started off by a given guess for $\underline{x}_n^k$. Following the steps above, we can obtain the values of $\underline{F}_n^k$. If they are not close enough to zero, we have to create better guesses in order to get a better guess than the previous for our dependent variables $\underline{x}$. This new guess we denote by $\underline{x}_n^{k+1}$. Executing the whole procedure described above once more, we get a new set of residuals of $\underline{F}$, i.e. $\underline{F}_n^{k+1}$. If they are still not close enough to zero, we need once more better guesses $\underline{x}_n^{k+2}$ to evaluate new residuals $\underline{F}_n^{k+2}$ and so on, until the residuals are close enough to zero and we call the overall equations to be satisfied at time step t_n.

Now, how do we get better guesses? A PDAE solver might use a simple Newton-like algorithm, for instance:

$$v_n^{k+1} = v_n^k - F_n^k * \left(\frac{\partial \underline{F}}{\partial \underline{v}} \bigg|_n^k \right)^{-1}$$

Even though this gives a new guess for the whole inner variable vector $\underline{v}_n^{k+1}$, only the values of $\underline{v}$ corresponding to dependent variables $\underline{x}$ are of interest and have to be extracted using mapping information provided via the inner variable interface.

Information concerning the partial derivative of $\underline{F}$ with respect to $\underline{v}$ can be obtained using the Jacobian information provided by the innerESO and outerESO, which provide $\frac{\partial \underline{F}}{\partial \underline{w}}$ and $\frac{\partial \underline{E}}{\partial \underline{v}}$, respectively. Obviously, the solver would have to consider the influence of the discretisation step, which was used to obtain $\underline{w}$ from $\underline{E}$, on the attempt to combine both Jacobian matrices in order to calculate $\frac{\partial \underline{F}}{\partial \underline{v}}$.

Once we arrived at satisfying the outer equations $\underline{F}$ at time step t_n, the solver has to come up with an estimate or guess for the next time step t_{n+1}. For instance, one very easy way would be to use the Jacobian information

provided by the innerESO, which provides $\frac{\partial \underline{E}}{\partial \underline{v}}$. Using the mapping information, it is possible to extract information concerning $\frac{\partial \underline{f}}{\partial t}$ out of the Jacobian matrix for the innerESO.

Algebraic manipulations transform this to give: $\frac{\partial \underline{x}}{\partial t} = \frac{\partial \underline{f}}{\partial t}\left(\frac{\partial \underline{f}}{\partial \underline{x}}\right)^{-1}$. Doing a simple Newton step, the desired new guesses for the dependent variable vector $\underline{x}_{n+1}$ can be obtained.

$$\underline{x}_{n+1} = \underline{x}_n + \left.\frac{\partial \underline{x}}{\partial t}\right|_n (t_{n+1} - t_n)$$

At this stage, all the information required to advance the solution in time is present. Note, however, that all kinds of discretisations or exploitation of derivative information presented above have purely demonstrative character and do not claim to be appropriate or accurate for a given problem at all. There exists a wide variety of possibilities how to address the issues of discretisation and derivative processing, but they are not part of the interface specification at all and part of the PDAE solver itself (see also Section 6.2).

In the following, the whole algorithm as described above is presented in pseudo code. Whenever variables are to be discretised, the notation $\delta\alpha/\delta\beta$ is used in order to indicate that this step is done internally by the solver. Further, it is assumed that all required guesses for the time stepping or residual iteration are provided by the solver.

```
// Procedure: OVERALL SOLVER

// construct the inner variable vector at time step tn and residual iteration k

// Procedure: INNER SOLVER

// initially, need to get mapping information
// get reference to innerESO

InnerESO := PDAESO -> GetInnerESO

// get reference to inner variable interfaces

InnerVariableRef := PDAESO -> GetInnerVariables

// InnerVariableRef is a list of innerVariable Interfaces

// receive number of variables in the innerESO

NIV := InnerESO -> GetNumVars

// there is an innerVariable Interface for each i=1..NIV
// each innerVariable v(i) is characterised by its kind
// depending on the kind of v(i), construct the innerVariable vector at tn and k

// Note: there is one innerVariable vector v for one specific value of z and t
// Solver creates and stores the innerVariable vector v for all values of z and // t in
IVV(i,z,t). Thus, the solver needs to loop for all z accordingly.

FOR i:=1 TO NIV DO

InnerVariableKind := InnerVariableRef(i) -> GetKind

IF InnerVariableKind=”IDVARIABLE” THEN
// in case InnerDependentVariable, we get the proper mapping of v(i)
// to its corresponding PDAESOVariable
            PDAESOVariable := InnerVariableRef(i) -> GetDependentVariable
// can now set the guess for x_n^k
      v(i) := x_n^k
```

```
// Further, can access all desired information. For instance:
v(i).LB := PDAESOVariable -> GetLB
// and also the index i_IVDV of the PDAESOVariable in the InnerVariable vector
i_IVDV := PDAESOVariable -> GetIndex
// this information is essential for the following kind of innerVariable

ELSEIF InnerVariableKind=”IDDVariable” THEN
// case InnerDependentDerivativeVariable
PDAESOVariable := InnerVariableRef(i) -> GetDependentVariable
PDAESODomain := InnerVariableRef(i) -> GetIndependentVariable
// the solver has to evaluate the partial differential
// δPDAESOVariable/δPDAESODomain. Thus, it needs to know which i_IVDV
// corresponds to the position of the dependent variable in the innerVariable
// vector v(i).
// Further, the solver has to know in which independent variable to discretise,
// i.e. the position of the corresponding independent variable in the
// innerVariable vector, say i_IVIV.

i_IVDV := PDAESOVariable -> GetIndex
i_IVIV := PDAESODomain -> GetIndex

// This will enable the solver to create the desired partial derivative by
// discretising information inherent in IVV(i_IVDV,z,t)
// specify value:
       v(i) := δIVV(i_IVDV,z,t)/δz(i_IVIV)

ELSEIF InnerVariableKind=”IDPVariable” THEN
// case InnerDependentProjectedVariable

PDAESOProjection:= InnerVariableRef(i) -> GetProjection
// the projected variable is characterised by the corresponding
// dependent variable and the value of the projected domain, i.e.
// the particular value of the independent variable vector z.

PDAESOVariable := PDAESOProjection -> GetDependentVariable
i_IVDV := PDAESOVariable -> GetIndex
       PDAESODomainRef := PDAESOProjection -> GetPDAESOPDDomainList
                     FOR each entry j in list DO
PDAESODomain := PDAESODomainRef(j) -> GetDomain
i_IVIV := PDAESODomain -> GetIndex
       z_P := PDAESODomainRef(j) -> GetValue
// can access desired dependent variable information from IVV(i_IVDV,z(z_P),t)
// and specify value
                           v(i) := IVV(i_IVDV,z(z_P),t)

ELSEIF InnerVariableKind=”IIVariable” THEN
// case InnerIndependentVariable
PDAESODomain := InnerVariableRef(i) -> GetIndependentVariable
// can access all desired information now. For instance:
i_IVIV := PDAESODomain -> GetIndex
// returns the index of the independent variable in innerVariable vector v
// Set value which corresponds to the specified z and t for that innerVariable.
             v(i) := v(i_IVIV)

ELSEIF InnerVariableKind=”ITVariable” THEN
// case InnerTimeVariable
             v(i) := tn

ENDIF

// inner variable vector v established
END

// can now evaluate inner equation for v
// set variable values
InnerESO -> SetVariableValues(v)

// and return residuals, i.e. function values in case of the innerESO
E := InnerESO -> GetResiduals
```

```
// Internally, the solver needs to execute the steps described above for all
// values of z, in order to create vector of all innerVariable vectors
// IVV(i,z,t) as well as the vector of all innerEquation vectors IEV(j,z,t).
// Remember, this is the desired information for time step t=tn and residual
// iteration step k. Now it is possible to prepare information required by the
// outerESO.


// Procedure: OUTER SOLVER

// Values for the outer variable vector w_n^k are constructed using the values
// stored in IVV(i,z,t). Once more w is the actual vector used by the solver in
// in order to compute values of the outer variables F at tn and k.

// initially, need to get mapping information
// get reference to outerESO

OuterESO := PDAESO -> GetOuterESO

// get reference to outer variable interfaces

OuterVariableRef := PDAESO -> GetOuterVariables

// OuterVariableRef is a list of outerVariable Interfaces

// receive number of variables in the outerESO

NOV := outerESO -> GetNumVars

// there is an outerVariable Interface for each i=1..NOV
// each outerVariable w(i) is characterised by its kind
// depending on the kind of w(i), construct the outerVariable vector at tn and k

// Note: there is one outerVariable vector w for one specific value of z and t
// The solver stores the outerVariable vector w for all values of z and t in // //
OVV(i,z,t). Thus, the solver needs to loop for all z accordingly.

// loop for all outer variables
FOR i := 1 TO NOV DO

// obtain index of corresponding equation in innerEquation vector IEV(j,z,t).
EqIndex := OuterVariableRef(i) -> GetInnerEqIndex

// obtain kind of outer variable
OuterVariableKind := OuterVariableRef(i) -> GetKind

// now have to distinguish all possible cases

IF OuterVariableKind=”fFunction” THEN
// case partial derivative wrt time
             w(i)= δIEV(EqIndex,z,t)/δt

ELSEIF OuterVariableKind=”gFunction” THEN
// case partial derivative wrt IndependentVariable z
// need to identify the particular independent variable z
             PDAESODomain := OuterVariableRef(i) -> GetIndependentVariable
// identify index of that domain in InnerVariableVector
             i_IVIV := PDAESODomain -> GetIndex
// can now calculate partial derivative
w(i)= δIEV(EqIndex,z,t)/δz(i_IVIV)

ELSEIF OuterVariableKind=”kFunction” THEN
// case non differentiated terms
       w(i)= IEV(EqIndex,z,t)
ENDIF

// finish outer variable loop, outer variable vector now established.

END

// can now evaluate outer equation for w
```

```
// set variable values
OuterESO -> SetVariableValues(w)

// and return residuals of F
F := OuterESO -> GetResiduals

// Internally, the solver needs to execute the steps described above for all
// values of z, in order to create the vector of all outerVariable vectors
// OVV(i,z,t) as well as the vector of all outerEquation vectors OEV(j,z,t).

// Remember, the vector OEV(j,z,t) is the vector of F for all z and t, and thus,
// the desired information for time step t=tn and residual iteration step k.

// Now it is possible to check whether the residuals of F are small enough,
// i.e. F=0 satisfied, or if another residual iteration k+1 is necessary.

// In the later case, the solver needs to provide new guesses for x_n^(k+1)
// and execute the overall Solver procedure again.

// Otherwise, time step t=t_n is finished and one step forward in time is to be
// taken, i.e. t=t_(n+1). This requires that the solver once more provides guesses
// for the dependent variables, x_(n+1)^(k=1) this time. Now, the overall Solver
// procedure needs to be executed again.
```

3.3 Interface diagrams

This section presents all interface diagrams. Notice that in section 3.5 a possible implementation of the interfaces is given in a class diagram. This contains as well the associations between he different classes, which help to understand the concept. In the interface diagrams the associations were omitted because between interfaces they do not exist.

IN- NUMBER 1 THE PDAESO INTERFACES

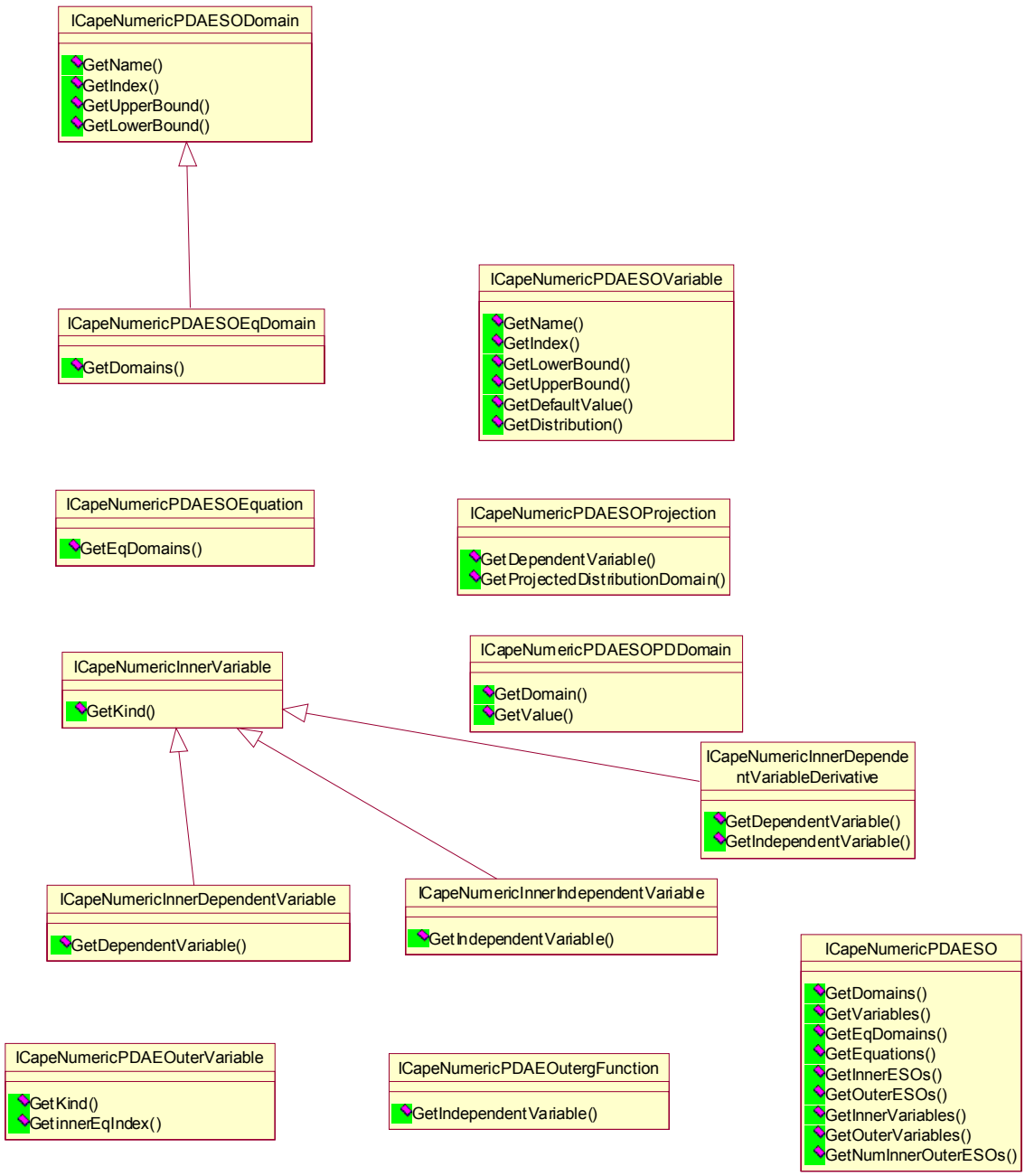


Figure 3 Interface diagram

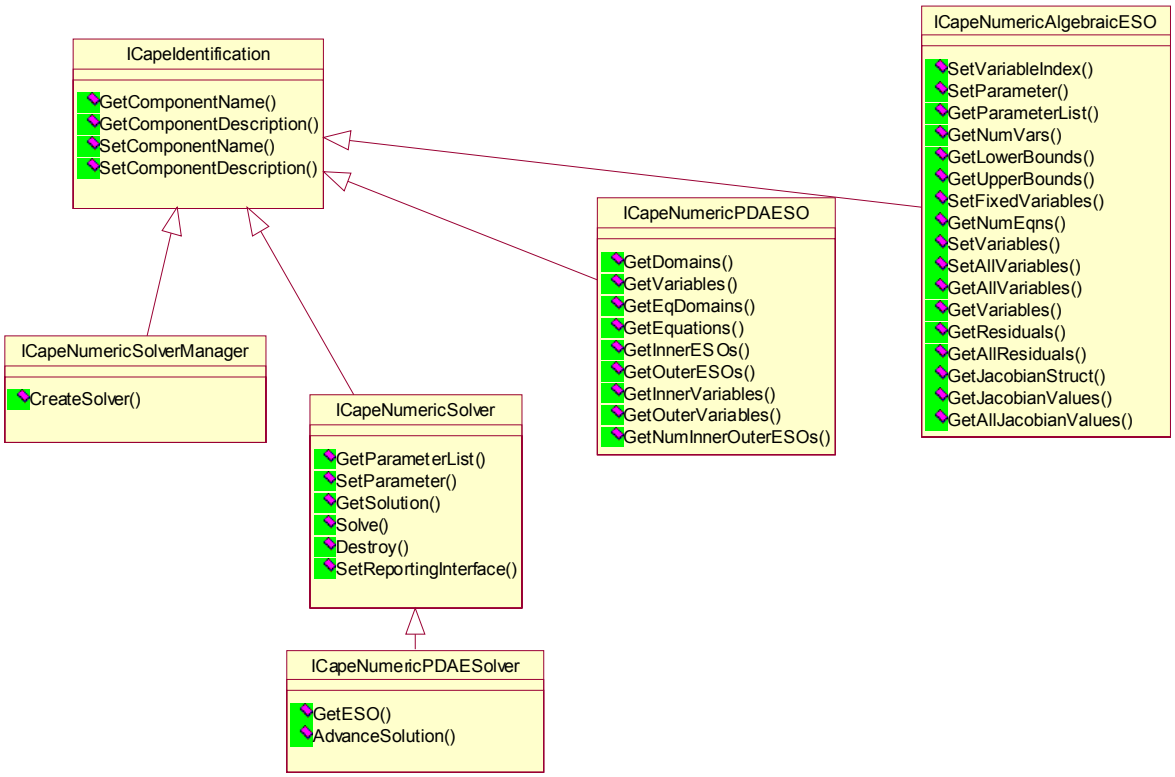


Figure 4 Interface diagram

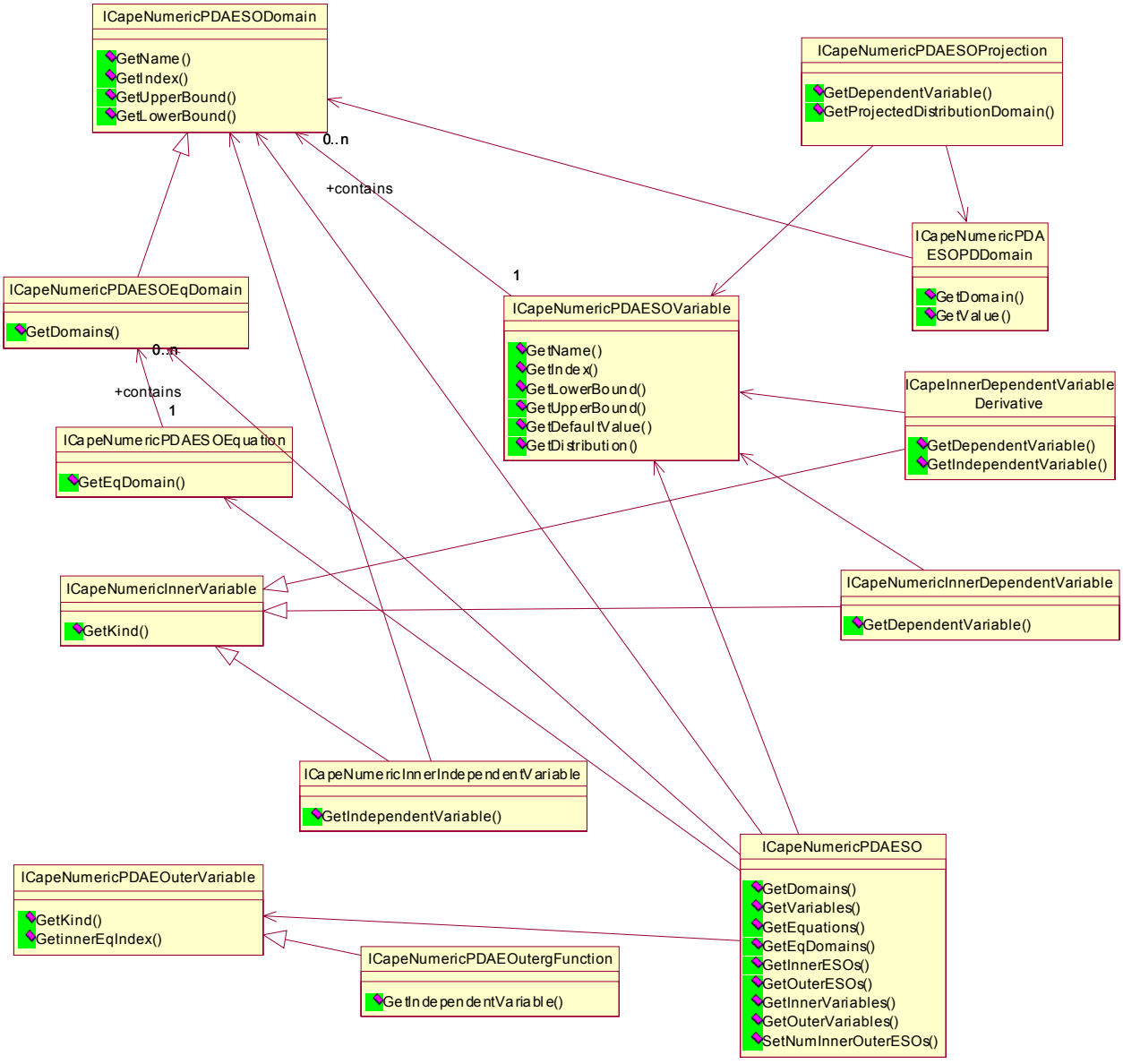
3.4 State diagrams

There are no state diagrams.

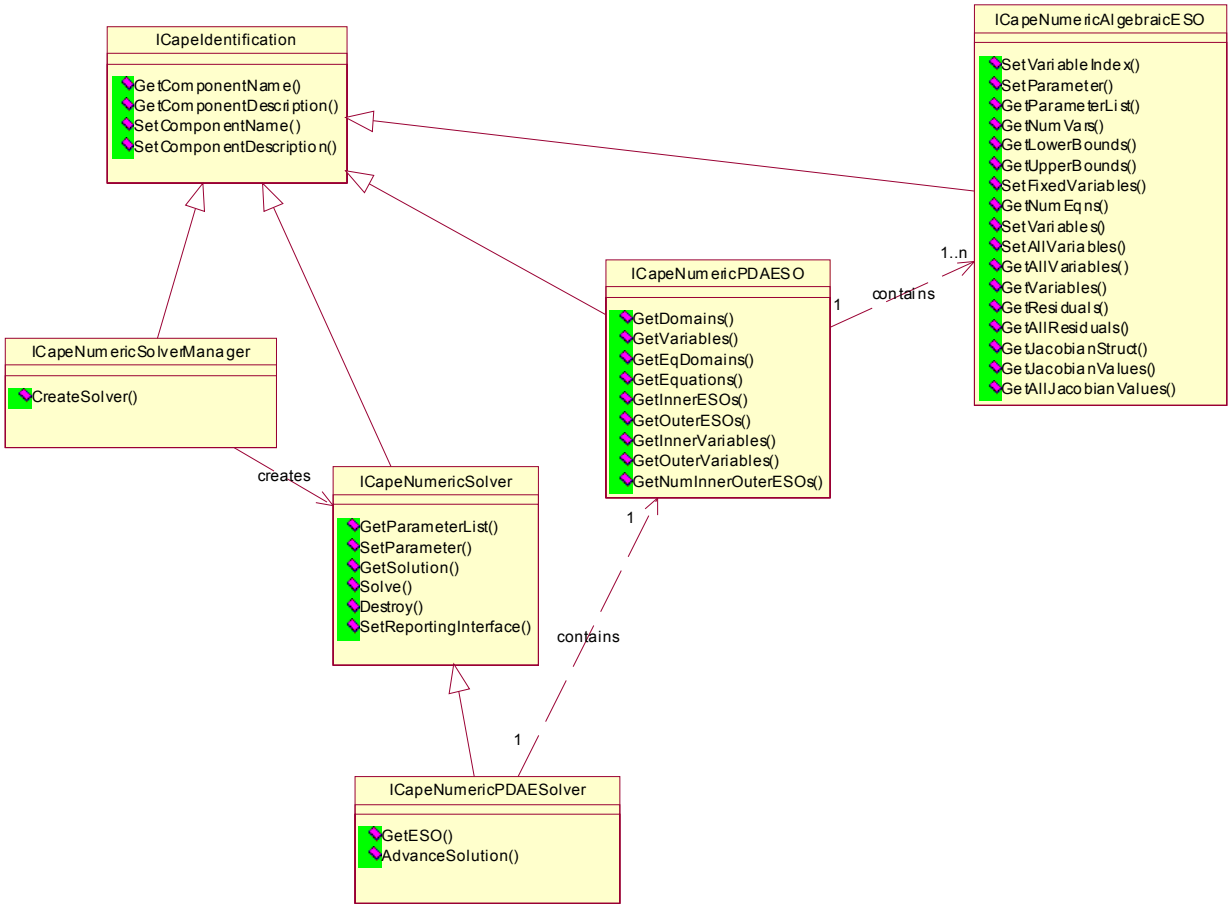
3.5 Other diagrams

If there are other diagrams (collaboration, component, class etc.) they should go here and follow the numbering scheme starting with two letters (CO, CP, CL etc…).

CL- NUMBER 1 POSSIBLE IMPLEMENTATION OF THE PDAESO INTERFACES



CL- NUMBER 2 POSSIBLE IMPLEMENTATION OF THE PDAESOLVER INTERFACES



3.6 Interface descriptions

This section details the specification of the methods appearing in the interface diagram. It should be noted that:

- Inherited methods are documented only under the parent interface, which defines them.
- To conform to the Guidelines document [3], all methods should return a CapeError value. One rôle of this value is to report a successful execution: the error conditions applicable to each method will have to be defined as part of the refinement of this interface definition.

The PDAESO Interfaces

This section describes the collection of interfaces used for the partial and differential algebraic equation set object.

In the following, interface specifications for domains are presented. A domain is used to identify on which list of independent variables a dependent variable is distributed.

3.6.1 ICapeNumericPDAESODomain

This is the interface PDAESODomain, which defines domains by their name and index, i.e. the position of the corresponding independent variable z in the inner variable vector $\underline{v}$ for the inner ESO.

Interface Name	ICapeNumericPDAESODomain
Method Name	GetName
Returns	CapeString

Description

Gets the name of the domain.

Arguments

None

Errors

ECapeUnknown

Interface Name	ICapeNumericPDAESODomain
Method Name	GetIndex
Returns	CapeLong

Description

Gets the index of the corresponding independent variable within the variable vector $\underline{v}$ of the inner ESO (cf. Section 3.1.2..5).

Arguments

None

Errors

ECapeUnknown

Interface Name	ICapeNumericPDAESODomain
Method Name	GetLowerBound
Returns	CapeDouble

Description

Gets the lower bound of the equation domain as well as the kind of the corresponding boundary, i.e. open or closed.

CapeBoundKind: CAPE_OPEN, CAPE_CLOSED

Arguments

Name	Type	Description
[out] bound_kind	CapeBoundKind	OPEN or CLOSED

Errors

ECapeUnknown

Interface Name	ICapeNumericPDAESODomain
Method Name	GetUpperBound
Returns	CapeDouble

Description

Gets the upper bound of the equation domain as well as the kind of the corresponding boundary, i.e. open or closed.

CapeBoundKind: CAPE_OPEN, CAPE_CLOSED

Arguments

Name	Type	Description
[out] bound_kind	CapeBoundKind	OPEN or CLOSED

Errors

ECapeUnknown

3.6.2 ICapeNumericPDAESOVariable

This interface is used to characterise the dependent variables occurring in the PDAESO. They are identified by a name, a lower bound and an upper bound for their value, a default value as well as the index for the corresponding entry in the inner variable vector $\underline{v}$. Additionally, information needs to be provided concerning the domains (independent variables) over which the variable is distributed.

Interface Name	ICapeNumericPDAESOVariable
Method Name	GetName
Returns	CapeString

Description

Gets the dependent variable name.

Arguments

None

Errors

ECapeUnknown

Interface Name	ICapeNumericPDAESOVariable
Method Name	GetIndex
Returns	CapeLong

Description

Gets the index of the dependent variable within the variable vector v of the inner ESO (cf. Section 3.1.2..5).

Arguments

None

Errors

ECapeUnknown

Interface Name	ICapeNumericPDAESOVariable
Method Name	GetLowerBound
Returns	CapeDouble

Description

Gets the lower bound of the dependent variable.

Arguments

None

Errors

ECapeUnknown

Interface Name	ICapeNumericPDAESOVariable
Method Name	GetUpperBound
Returns	CapeDouble

Description

Gets the upper bound of the dependent variable.

Arguments

None

Errors

ECapeUnknown

Interface Name	ICapeNumericPDAESOVariable
Method Name	GetDefaultValue
Returns	CapeDouble

Description

Gets the default value of the dependent variable.

Arguments

None

Errors

ECapeUnknown

Interface Name	ICapeNumericPDAESOVariable
Method Name	GetDistribution
Returns	CapeInterface (CapeArrayNumericPDAESODomain)

Description

Gets the list of distribution domains over which the dependent variable is distributed.

Arguments

None

Errors

ECapeUnknown

3.6.3 ICapeNumericPDAESOEqDomain

Inherits from: ICapeNumericPDAESODomain

Just like the dependent variables that are distributed on a particular set of PDAESO-domains, are also distributed on domains, so called equation domains or PDAESO-EqDomain. They differ from variable domains by the fact that both lower and upper bound are not only characterised by a bounding value, but also a kind which indicated whether the boundary is an open or a closed boundary. Note further that

PDAESOEqDomains are subsets of PDAESODomains, i.e. the bounds of the equation domains have to remain within the bounds of the variables.

Interface Name	ICapeNumericPDAESOEqDomain
Method Name	GetDomain
Returns	CapeInterface (ICapeNumericPDAESODomain)

Description

Gets the independent variable domain, of which the equation validity domain is a subset. This can be used to check whether the bounds of the PDAESOEqDomain lie within the bounds of the domains of the PDAESOVariables.

Arguments

None

Errors

ECapeUnknown

Interface Name	ICapeNumericPDAESOEqDomain
Method Name	GetUBKind
Returns	CapeBoundKind

Description

None.

Arguments

None

Errors

ECapeUnknown

Interface Name	ICapeNumericPDAESOEqDomain
Method Name	GetLBKind
Returns	CapeBoundKind

Description

None.

Arguments

None

Errors

ECapeUnknown

3.6.4 ICapeNumericPDAESOEquation

This interface is used to characterise an equation by the list of equation validity domains over which it is distributed.

Interface Name	ICapeNumericPDAESOEquation
Method Name	GetEqDomains
Returns	CapeArrayNumericPDAESOEqDomain

Description

Gets the list of equation validity domains over which an equation is distributed.

Arguments

None

Errors

ECapeUnknown

3.6.5 ICapeNumericPDAESOProjection

This interface is used to characterise dependent variables that are present in the equations on a restricted set of distribution domains, i.e. one or more distributions have a fixed value.

Interface Name	ICapeNumericPDAESOProjection
Method Name	GetDependentVariable
Returns	ICapeNumericPDAESOVariable

Description

Gets the dependent variable which is used on a restricted distribution by fixing one or more domains.

Arguments

None

Errors

ECapeUnknown

Interface Name	ICapeNumericPDAESOProjection
Method Name	GetProjectedDistributionDomain
Returns	ICapeNumericPDAESOPDDomain

Description

Gets the PDAESODomain, which is restricted by fixing the value of the corresponding independent variable.

Arguments

None

Errors

ECapeUnknown

3.6.6 ICapeNumericPDAESOPDDomain

This interface is used to characterise domains, which obtain a fixed value and thus, restrict the distribution of the dependent variable under consideration.

Interface Name	ICapeNumericPDAESOPDDomain
Method Name	GetDomain
Returns	ICapeNumericPDAESODomain

Description

Gets the domain which is given a fixed value.

Arguments

None

Errors

ECapeUnknown

Interface Name	ICapeNumericPDAESOPDDomain
Method Name	GetValue
Returns	CapeDouble

Description

Gets the value of the independent variable corresponding to the domain which is fixed.

Arguments

None

Errors

ECapeUnknown

3.6.7 ICapeNumericInnerVariable

This interface is used to characterise the variables occurring in the inner ESO, the so-called ``inner variables''. It provides the mapping between the variable array of the inner ESO and the dependent and independent variables of the mathematical problem formulation.

Interface Name	ICapeNumericInnerVariable
Method Name	GetKind
Returns	CapeInnerVariableKind

Description

Gets the kind of inner variable. This can be:

CapeInnerVariableKind: CAPE_ID, CAPE_IDP, CAPE_IDD, CAPE_II, CAPE_IT

- IDVariable: Inner Dependent Variable
- IDPVariable: Inner Dependent Projected Variable
- IDDVariable: First-order partial derivative of Inner Dependent Variable with respect to a (non-temporal) distribution domain.
- IIVariable: Inner Independent Variable
- ITVariable: Inner Time Variable

Arguments

None

Errors

ECapeUnknown

3.6.8 ICapeNumericInnerDependentVariable

Inherits from: ICapeNumericInnerVariable

This interface is used to characterise information about the dependent variables occurring in the inner ESO.

Interface Name	ICapeNumericInnerDependentVariable
Method Name	GetDependentVariable
Returns	CapeInterface (ICapeNumericPDAESOVariable)

Description

Provides the reference to the PDAESOVariable which represents this inner dependent variable.

Arguments

None

Errors

ECapeUnknown

3.6.9 ICapeNumericInnerDependentVariableDerivative

Inherits from: ICapeNumericInnerVariable

This interface is used to characterise information about the partial derivatives of dependent variables occurring in the inner ESO.

Interface Name	ICapeNumericInnerDependentVariableDerivative
Method Name	GetDependentVariable
Returns	ICapeNumericPDAESOVariable

Description

Gets the inner dependent variable which is differentiated.

Arguments

None

Errors

ECapeUnknown

Interface Name	ICapeNumericInnerDependentVariableDerivative
Method Name	GetIndependentVariable
Returns	ICapeNumericPDAESODomain

Description

Gets the independent variable with respect to which the inner dependent variable is differentiated.

Arguments

None

Errors

ECapeUnknown

3.6.10 ICapeNumericInnerIndependentVariable

Inherits from: ICapeNumericInnerVariable

This interface is used to characterise information about independent variables occurring in the inner ESO.

Interface Name	ICapeNumericInnerIndependentVariable
Method Name	GetIndependentVariable
Returns	ICapeNumericPDAESODomain

Description

Gets the distribution domain corresponding to this inner independent variable.

Arguments

None

Errors

ECapeUnknown

3.6.11 ICapeNumericOuterVariable

This interface is used to characterise information about variables occurring in the outer ESO, so called outer variables.

Interface Name	ICapeNumericOuterVariable
Method Name	GetKind
Returns	CapeOuterVariableKind

Description

Gets the kind of outer variable. This can be:

CapeOuterVariableKind: CAPE_FFUNCTION, CAPE_GFUNCTION, CAPE_HFUNCTION, CAPE_KFUNCTION

- fFunction: Variable wt which represents the time derivative term
- gFunction: Variable wg which represents the term for differentiation with respect to independent variables z.
- kFunction: Variable wk which represents the (non-)linear terms.

Arguments

None

Errors

ECapeUnknown

Interface Name	ICapeNumericOuterVariable
Method Name	GetInnerEqIndex
Returns	CapeLong

Description

Gets the index of the expression (``equation'') within the inner ESO that corresponds to this outer variable.

Arguments

None

Errors

ECapeUnknown

3.6.12 ICapeNumericOutergFunction

Inherits from: ICapeNumericOuterVariable

This interface is used to characterise information about the independent variables occurring in the gFunction in the outer ESO, i.e. the domain z with respect to which g is differentiated.

Interface Name	ICapeNumericOutergFunction
Method Name	GetIndependentVariable
Returns	ICapeNumericPDAESODomain

Description

Gets the domain of distribution for the inner function g.

Arguments

None

Errors

ECapeUnknown, ECapeNoImpl

3.6.13 ICapeNumericPDAESO

Due to the separation of the PDAESO in an inner and outer ESO, main issue of this interface is to identify all issues related to assembling necessary information for the iESO and oESO. This includes domains on which dependent (inner) variables are defined, the (inner) variables themselves as well as the domains on which the equations are defined and the equations themselves.

This interface is further used to identify the proper addressing of inner and outer variables as well as inner and outer ESOs.

Interface Name	ICapeNumericPDAESO
Method Name	GetDomains
Returns	CapeArrayNumericPDAESODomain

Description

Gets the domains on which the (inner) variables are defined.

Arguments

None

Errors

ECapeUnknown

Interface Name	ICapeNumericPDAESO
Method Name	GetVariables
Returns	CapeArrayNumericPDAESOVariable

Description

Gets the (inner) variables.

Arguments

None

Errors

ECapeUnknown

Interface Name	ICapeNumericPDAESO
Method Name	GetEquations
Returns	CapeArrayNumericPDAESOEquation

Description

Gets the equations.

Arguments

None

Errors

ECapeUnknown

Interface Name	ICapeNumericPDAESO
Method Name	GetInnerESOs
Returns	CapeArrayNumericESO

Description

Gets the inner ESOs.

Arguments

None

Errors

ECapeUnknown

Interface Name	ICapeNumericPDAESO
Method Name	GetOuterESOs
Returns	CapeArrayNumericESO

Description

Gets the outer ESOs.

Arguments

None

Errors

ECapeUnknown

Interface Name	ICapeNumericPDAESO
Method Name	GetInnerVariables
Returns	CapeArrayNumericInnerVariable

Description

Gets the inner Variables.

Arguments

None

Errors

ECapeUnknown

Interface Name	ICapeNumericPDAESO
Method Name	GetOuterVariables
Returns	CapeArrayNumericOuterVariable

Description

Gets the outer Variables.

Arguments

None

Errors

ECapeUnknown

Interface Name	ICapeNumericPDAESO
Method Name	GetNumInnerOuterESOs
Returns	CapeLong

Description

Gets the number of inner and outer ESOs contained in the PDAESO for each domain.

Arguments

None

Errors

ECapeUnknown

Interface Name	ICapeNumericPDAESO
Method Name	SetNumInnerOuterESOs
Returns	--

Description

Set the number of Inner and Outer ESOs for each domain.

Arguments

Name	Type	Description
[in] num	CapeLong	Set the number of Inner and Outer ESOs for each domain

Errors

ECapeUnknown

PDAE Solution Interfaces

These interfaces define methods for the identification and setting of parameters that will occur in *all* CAPE-OPEN compliant partial-differential-algebraic components. So far, it has not been possible to identify any number of such generic parameters, because each solver relies on a very specific set. Therefore the interface specification is preliminary.

3.6.14 ICapeIdentification

See specification of Common CO Interfaces.

3.6.15 ICapeNumericSolver

See CO Solvers Specification.

3.6.16 ICapeNumericSolverManager

See CO Solvers Specification.

3.6.17 ICapeNumericPDAESolver

Inherits from: ICapeNumericSolver

This is the interface of the Partial-Differential-Algebraic Equation Solver, which solves systems of equations of the form:

$$\underline{F}\left[\frac{\partial}{\partial t}\underline{f}(\underline{x},\underline{x}^{P},\underline{z},t),\frac{\partial}{\partial \underline{z}}\underline{g}\left(\frac{\partial \underline{x}}{\partial \underline{z}},\underline{x},\underline{x}^{P},\underline{z},t\right),\underline{k}(\underline{x},\underline{x}^{P},\underline{z},t)\right]=\underline{0}$$

over some range of values of the independent variables $\underline{z},t$.

Interface Name	ICapeNumericPDAESolver
Method Name	GetESO
Returns	ICapeNumericPDAESO

Description

Gets the ESO with which an PDAESolver was constructed.

Arguments

None

Errors

ECapeUnknown

Interface Name	ICapeNumericPDAESolver
Method Name	AdvanceSolution
Returns	--

Description

Advances the solution of the PDAESO with respect to its independent variables.

Arguments

None

Errors

ECapeUnkown, ECapeInvalidArgument, ECapeBadArgument, ECapeOutOfBounds, ECapeTimeOut, ECapeOutOfResources, ECapeNoMemory

3.7 Scenarios

3.7.1 Example of PDAESO

This example is meant to describe how the mathematical model of a physical problem is placed into a form that is accessible by all the interfaces defined in chapter 0. Provided an appropriate PDAE solver and placing it into the framework of Solver and Solver factories, the problem could then be solved like outlined in chapter 3.2.1.

Problem formulation

The physical problem under consideration is as follows. Consider the flow of a binary, non-reacting mixture through a tube of lengthand radius. The component balance for this system reads:

$$\frac{\partial \rho_i}{\partial t} + (\nabla \cdot \rho_i \underline{u}) = (\nabla \cdot D_i \nabla \rho_i) \qquad i = 1,..,2$$

Consider the flow to be adequately described by plug flow, i.e. no variation of entities with respect to radius *r* and azimutal direction Thus, in total, the problem extends over 2 domains, described by the independent variables time *t* and space *z*.

Let us choose the components Water (=998 kg/m^3) and Ethanol (=789 kg/m^3) for our example. The densities are quite similar and thus, the assumption of constant overall bulk density $\overline{\rho}$ can be justified. The overall density is denoted by, and thus,.

Thus, the system of partial differential and algebraic equations is as follows.

Model equations

$$\bar{\rho}\frac{\partial y_1}{\partial t}+\bar{\rho}\frac{\partial}{\partial z}(y_1 u_z)=\bar{\rho}\frac{\partial}{\partial z}\left(D_{12}\frac{\partial y_1}{\partial z}\right) \qquad \forall z\in(0,L_z) \qquad (7.1)$$

$$y_1+y_2=1 \qquad \forall z\in[0,L_z] \qquad (7.2)$$

$$D_{12}=a_1 {y_1}^4+a_2 {y_1}^3+a_3 {y_1}^2+a_4 y_1+a_5 \qquad \forall z\in[0,L_z] \qquad (7.3)$$

$$u_z(z)=10^{-1}\frac{m}{s} \qquad \forall z\in[0,L_z] \qquad (7.4)$$

$$\bar{\rho}=893{,}5\frac{kg}{m^3} \qquad (7.5)$$

At 25°C, the parameters are:

$$a_1=204{,}45\frac{m^2}{s} \quad a_2=-483{,}98\frac{m^2}{s} \quad a_3=415{,}67\frac{m^2}{s} \quad a_4=-127{,}82\frac{m^2}{s} \quad a_5=16{,}711\frac{m^2}{s}$$

So far, no information has been provided as to which initial and boundary conditions are specified.

Initial conditions

Since the above problem is posed as a time evolution problem, we need one initial condition. This is the following:

$$t=0: \quad y_1=0.5 \qquad \forall z\in(0,L_z)$$

Boundary conditions

Equation (7.1) requires four boundary conditions, one for each domain boundary.

At the inlet, assume a given mass fraction for y_1. At the outlet, Neumann boundary conditions are applied. Symmetry conditions hold for the tube axis, and at the tube wall, it is assumed that the mass fraction cannot change because there is neither convection nor diffusion in radial direction.

$$z=0: \qquad y_1=y_1^0 \qquad (7.6)$$

$$z=L_z: \qquad \frac{\partial y_1}{\partial z}=0 \qquad (7.7)$$

Now, before the PDAESO can be constructed, this mathematical model needs to be placed into a form that can be accessed via the PDAESODomain, PDAESOVariable, PDAESOEqDomain and PDAESOEquation interfaces.

Because the independent variables are positioned after the dependent variables in the inner variable vector, the index of the corresponding domains has to be left blank until all dependent variables and their derivatives have been placed into the inner variable vector.

Domains

The problem is posed on the spatial domain *z,* denoted by "AXIAL" in the PDAESODomain context and defined as follows. The domain time dealt with separately and the discrete entity mass fraction is considered simply by introducing as many variables as there are discrete entries.

The domains are summarised in a sequence of type CapeArrayPDAESODomain with the name PDAESODomain.

- PDAESODomain(1).Name="AXIAL"
- PDAESODomain(1).Index=7
- PDAESODomain(1).LB=0
- PDAESODomain(1).UB=Lz

Variables

Dependent variables in use are $y_1, y_2, u_z, D_{12}, \overline{\rho}$, which corresponds to mass fraction, velocity, diffusion coefficient and overall bulk density, respectively.

$\frac{\partial y_1}{\partial z}$ is the only dependent partial derivative in use.

$\underline{y}, u_r, u_z, D_{12}$ and $\frac{\partial y_1}{\partial z}$ are distributed on the spatial domain AXIAL and time. Providing information accessible via the PDAESOVariable interface is as follows.

- PDAESOVariable(1).Name="*y1*"
- PDAESOVariable(1).Index=1
- PDAESOVariable(1).LB=0
- PDAESOVariable(1).UB=1
- PDAESOVariable(1).DefaultValue=0.5
- PDAESOVariable(1).Distribution=PDAESODomain(1)

- PDAESOVariable(2).Name="*y2*"
- PDAESOVariable(2).Index=2
- PDAESOVariable(2).LB=0
- PDAESOVariable(2).UB=1
- PDAESOVariable(2).DefaultValue=0.5
- PDAESOVariable(2).Distribution=PDAESODomain(1)

- PDAESOVariable(3).Name="uz"
- PDAESOVariable(3).Index=3
- PDAESOVariable(3).LB=-1E+10

- PDAESOVariable(3).UB=1E+10
- PDAESOVariable(3).DefaultValue=1E0
- PDAESOVariable(3).Distribution=PDAESODomain(1)

- PDAESOVariable(4).Name="D12"
- PDAESOVariable(4).Index=4
- PDAESOVariable(4).LB=0
- PDAESOVariable(4).UB=100
- PDAESOVariable(4).DefaultValue=0.9
- PDAESOVariable(4).Distribution=PDAESODomain(1)

- PDAESOVariable(5).Name="$\overline{\rho}$"
- PDAESOVariable(5).Index=5
- PDAESOVariable(5).LB=0
- PDAESOVariable(5).UB=1E+4
- PDAESOVariable(5).DefaultValue=893,5
- PDAESOVariable(5).Distribution= <empty>

Equation Validity Domains

Before specifying the equations, all possible domains on which they are valid need to be declared. The equation validity domains are subsets of the variable domains, i.e. they can be excluded on the boundary. Note that boundary conditions are treated as ordinary equations on a specific domain, i.e. PDAEs as boundary conditions are perfectly permitted. Each case requires its own interface. With respect to the example, this corresponds to two different domains. We use a sequence of type CapeArrayPDAESOEqDomain with the name PDAESOEqDomain.

$z \in (0, L_z)$ is described by:

- PDAESOEqDomain(1).Domain=PDAESODomain(1)
- PDAESOEqDomain(1).LB=PDAESODomain(1).LB
- PDAESOEqDomain(1).UB=PDAESODomain(1).UB
- PDAESOEqDomain(1).LBKind=OPEN
- PDAESOEqDomain(1).UBKind=OPEN

$z \in [0, L_z]$ is described by:

- PDAESOEqDomain(2).Domain=PDAESODomain(1)
- PDAESOEqDomain(2).LB=PDAESODomain(1).LB
- PDAESOEqDomain(2).UB= PDAESODomain(1).UB
- PDAESOEqDomain(2).LBKind=CLOSED
- PDAESOEqDomain(2).UBKind=CLOSED

Equations

Now, the information accessed via the PDAESOEquation interface needs to be established. In fact, these interfaces provide references to the corresponding PDAESOEqDomain interfaces. The index in the PDAESEOEquation list corresponds to the actual equation number in this chapter, i.e. information about equation (7.1) will be accessible via PDAESOEquation(1).

- PDAESOEquation(1).EqDomain=PDAESOEqDomain(1)
- PDAESOEquation(2).EqDomain=PDAESOEqDomain(2)
- PDAESOEquation(3).EqDomain=PDAESOEqDomain(2)
- PDAESOEquation(4).EqDomain=PDAESOEqDomain(2)
- PDAESOEquation(5).EqDomain={ }
- PDAESOEquation(6).EqDomain={ }
- PDAESOEquation(7).EqDomain={ }

PDAESO

Now, it is possible to construct the information inherent to the PDAESO, i.e. the innerESO, outerESO, InnerVariables and OuterVariables.

3.7.1..1 InnerESO

In order to assemble the innerESO, it is necessary to extract the information about the inner equations out of the system of overall model equations. Thus, initially the functions $\underline{f}, \underline{g}, \underline{k}$ are extracted and then placed into the vector $\underline{E}$ of inner equations. In the same time, this provides information as to how to embed mapping information which is to be accessed using the InnerVariable interface.

Equation (7.1):

$f_1 = \overline{\rho} y_1$ differentiated with respect to t

$g_1 = \overline{\rho} u_z y_1$ differentiated with respect to z

$g_2 = -\overline{\rho} D_{12} \frac{\partial y_1}{\partial z}$ differentiated with respect to z

Equation (7.2):

$$k_1 = y_1 + y_2 - 1$$

Equation (7.3):

$$k_2 = D_{12} - \left(a_1 y_1^4 + a_2 y_1^3 + a_3 y_1^2 + a_4 y_1 + a_5\right)$$

Equation (7.4):

$$k_3 = u_z - 10^{-1}$$

Equation (7.5):

$$k_4 = \overline{\rho} - 893{,}5$$

Equation (7.6):

$$k_5 = y_1 - y_1^0$$

Equation (7.7):

$g_3 = y_1$ differentiated with respect z

Thus, the vector of inner equations can now be assembled:

$$\underline{E}(\underline{v}) = \begin{pmatrix} f_1 \\ g_1 \\ \vdots \\ g_3 \\ k_1 \\ \vdots \\ k_5 \end{pmatrix}, \text{ where } \underline{v} = \begin{pmatrix} y_1 \\ y_2 \\ u_z \\ D_{12} \\ \overline{\rho} \\ \frac{\partial y_1}{\partial z} \\ z \\ t \end{pmatrix}.$$

This reduces the innerESO to be an equation set object on $\underline{E}(\underline{v}) = 0$

3.7.1..2 Inner Variables

It is now possible to devise the information accessible via the InnerVariable Interface.

It follows from *dim(v)=8*, that there are 8 inner variables and thus, 8 corresponding InnerVariable interfaces.

Information regarding y_1:

- InnerVariable(1).Kind=IDVARIABLE
- InnerVariable(1).DependentVariable=PDAESOVariable(1)

Information regarding y_2:

- InnerVariable(2).Kind=IDVARIABLE
- InnerVariable(2).DependentVariable=PDAESOVariable(2)

Information regarding u_z:

- InnerVariable(3).Kind=IDVariable
- InnerVariable(3).DependentVariable=PDAESOVariable(3)

Information regarding D_{12}:

- InnerVariable(4).Kind=IDVARIABLE
- InnerVariable(4).DependentVariable=PDAESOVariable(4)

Information regarding $\overline{\rho}$:

- InnerVariable(5).Kind=IDVARIABLE
- InnerVariable(5).DependentVariable=PDAESOVariable(5)

Information regarding $\frac{\partial y_1}{\partial z}$:

- InnerVariable(6).Kind=IDDVARIABLE
- InnerVariable(6).DependentVariable=PDAESOVariable(1)
- InnerVariable(6).IndependentVariable=PDAESODomain(1)

Information regarding *z*:

- InnerVariable(7).Kind=IIVARIABLE
- InnerVariable(7).IndependentVariable=PDAESODomain(1)

Information regarding *t*:

- InnerVariable(8).Kind=ITVariable

Provided this information, the InnerESO can be evaluated by the solver and used to produce the vector of outer variables $\underline{w}$. These are required for the outerESO.

3.7.1..3 Outer ESO

Using the functions constituting the inner ESO, the outer variables can be assembled.

$$w_1 = \frac{\partial f_1}{\partial t}$$

$$w_2 = \frac{\partial g_1}{\partial z}$$

$$w_3 = \frac{\partial g_2}{\partial z}$$

$$w_4 = \frac{\partial g_3}{\partial z}$$

$$w_{i=5,..,9} = k_{i-4}$$

This in turn allows constructing the outer equations $\underline{F}(\underline{w}) = 0$.

Equation (7.1):

$$F_1 = w_1 + w_2 + w_3 = 0$$

Equation (7.2):

$$F_2 = w_5 = 0$$

Equation (7.3):

$$F_3 = w_6 = 0$$

Equation (7.4):

$$F_4 = w_7 = 0$$

Equation (7.5):

$$F_5 = w_8 = 0$$

Equation (7.6):

$$F_6 = w_9 = 0$$

Equation (7.7):

$$F_7 = w_4 = 0$$

This is the vector of outer equations and together with the outer variables, the outer ESO can be stated as the following problem:

$$\underline{F}(\underline{w}) = 0$$

3.7.1..4 Outer Variables

For each of the outer variables presented above, specific information has to be made accessible via the OuterVariable interface. Thus, there are NOV=9 interfaces.

Outer variable $w_1 = \frac{\partial f_1}{\partial t}$:

- OuterVariable(1).Kind=FFUNCTION
- OuterVariable(1).InnerEqIndex=1

Outer variable $w_2 = \frac{\partial g_1}{\partial z}$:

- OuterVariable(2).Kind=GFUNCTION
- OuterVariable(2).InnerEqIndex=2
- OuterVariable(2).IndependentVariable=PDAESODomain(1)

Outer variable $w_3 = \frac{\partial g_2}{\partial z}$:

- OuterVariable(3).Kind=GFUNCTION
- OuterVariable(3).InnerEqIndex=3
- OuterVariable(3).IndependentVariable=PDAESODomain(1)

Outer variable $w_4 = \frac{\partial g_3}{\partial z}$:

- OuterVariable(4).Kind=GFUNCTION
- OuterVariable(4).InnerEqIndex=4
- OuterVariable(4).IndependentVariable=PDAESODomain(1)

Outer variables $w_{i=5,..,9} = k_{i-4}$ for i=5,..,9:

- OuterVariable(i).Kind=KFUNCTION
- OuterVariable(i).InnerEqIndex=i

Provided this information, a proper mapping of the outer variables is possible.

Concluding, all the information required for a solution of the model equations (7.1)-(7.7) following the solution approach described in section 3.2.1 is given. This includes the information accessible using the following interfaces:

- PDAESODomain
- PDAESOVariable
- PDAESOEqDomain
- PDAESOEquation
- InnerVariable
- OuterVariable

Further, the innerESO and outerESO as part of the PDAESO have been derived from the model equations.

4. Interface Specifications

This section contains the CORBA IDL instructions; they are compilable files that you can directly use for producing CAPE-OPEN compliant components.

4.1 COM IDL

```
// This specification is not currently available for COM platform.
```

4.2 CORBA IDL

```
// You can get these intructions in CAPE-OPENv1-0-0.idl within the
CAPEOPEN100::Business::Numeric::PdaEso module and CAPEOPEN100::Business::Numeric::Solver
module
```

5. Notes on the interface specifications

6. Prototypes implementation

See prototypes document.

7. Specific Glossary Terms

Introduce glossary definitions as needed, in addition to the existing glossary material.

8. Bibliography

The bibliography is organised in three sections as shown below.

8.1 Process simulation references

8.2 Computing references

8.3 General references

(i) Keeping, B., Pantelides, C.C.: *WP4.1 Numerical Solvers Open Interface Specification Draft*. CAPE-OPEN project document, 1999.

(ii) J. Hackenberg, C. Krobb, W. Marquardt: An Object-Oriented Data Model to Capture Lumped and Distributed Parameter Models of Physical Solvers. In: I. Troch, F. Breitenecker (Eds.): Proceedings of IMACS Symposium on Mathematical Modelling, Vienna, Austria, 2-4.2.2000, ARGESIM Report No. 15, 339-342, 2000.

(iii) Cape Open Solvers Specification 1999

(iv) CAPE-OPEN Interfaces Guidelines and Review: CO-MGT-QS-20 (Version 1, November 1997).

9. Appendices