

CAPE-OPEN

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

Open Interface Specification: Parameter Common Interface



www.colan.org

ARCHIVAL INFORMATION

Filename	Parameter Common Interface.doc
Authors	CO-LaN consortium
Status	Public
Date	August 2003
Version	version 5
Number of pages	50
Versioning	version 5, reviewed by Jean-Pierre Belaud, August 2003
	version 4, Methods & Tools group, March 2001
Additional material	
Web location	www.colan.org
Implementation specifications version	CAPE-OPENv1-0-0.idl (CORBA) CAPE-OPENv1-0-0.zip and CAPE-OPENv1-0-0.tlb (COM)
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.

Copyright © 2003 CO-LaN and/or suppliers. All rights are reserved unless specifically stated otherwise.

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 describes a Common Interface proposed by the Methods & Tools group: the **Parameter Common Interface**. The Common Interfaces are interfaces and implementation models for handling concepts that may be required by any CO interface specification.

The various interfaces representing model parameters were already used by the UNIT specification (CO-CUNIT-1 Version 2.0), where this was included as Appendix 2 to the specification, and the Numerical Solvers Interface Specification (CO-NUMR-EL-03 Version 1.08). Therefore this document consolidates these two approaches into a single set of interfaces that can be re-used by these two packages and any other CO packages requiring similar services.

In addition, the document corrects some of the deficiencies encountered in the original Interface Proposal for parameter. This concerns e.g. handling of vector/matrices variables as a single Parameter object.

ACKNOWLEDGEMENTS

CONTENTS

1. INTRODUCTION	8
2. REQUIREMENTS	9
2.1 TEXTUAL REQUIREMENTS.....	9
2.2 USE CASES.....	10
2.2.1 <i>Actors</i>	11
2.2.2 <i>List of Use Cases</i>	11
2.2.3 <i>Use Cases Map</i>	11
2.2.4 <i>Use Cases</i>	12
3. ANALYSIS AND DESIGN	17
3.1 OVERVIEW.....	17
3.2 SEQUENCE DIAGRAMS.....	19
3.3 INTERFACE DIAGRAMS.....	21
3.4 STATE DIAGRAMS.....	23
3.5 OTHER DIAGRAMS.....	23
3.6 INTERFACES DESCRIPTIONS.....	24
3.6.1 <i>ICapeParameter</i>	24
3.6.2 <i>ICapeParameterSpec</i>	29
3.6.3 <i>ICapeRealParameterSpec</i>	32
3.6.4 <i>ICapeArrayParameterSpec</i>	34
3.6.5 <i>ICapeIntegerParameterSpec</i>	37
3.6.6 <i>ICapeOptionParameterSpec</i>	39
3.6.7 <i>ICapeBooleanParameterSpec</i>	41
3.7 SCENARIOS.....	42
4. INTERFACE SPECIFICATIONS	43
4.1 COM IDL.....	43
4.2 CORBA IDL.....	43
5. NOTES ON THE INTERFACE SPECIFICATIONS	44
5.1 CHANGES WITH RESPECT TO APPENDIX 2 OF THE UNIT SPECIFICATION.....	44
6. PROTOTYPES IMPLEMENTATION	45
6.1 CREATING AND CONFIGURING THE LIST OF PARAMETERS (ACTOR: PARAMETERS OWNER).....	45
6.2 VALIDATING PARAMETERS (ACTOR: PARAMETERS CLIENT).....	46
7. SPECIFIC GLOSSARY TERMS	48
8. BIBLIOGRAPHY	49
9. APPENDICES	50

LIST OF FIGURES

FIGURE 1 PARAMETER USE CASES MAP12

FIGURE 2 CONCEPTUAL OVERVIEW OF PARAMETER (A)17

FIGURE 3 CONCEPTUAL OVERVIEW OF PARAMETER (B).....18

FIGURE 4 CREATING/CONFIGURING LIST OF PARAMETERS.....19

FIGURE 5 GETTING LIST OF PARAMETERS20

FIGURE 6 DISPLAYING LIST OF PARAMETERS20

FIGURE 7 PARAMETERS CLIENT CHANGES PARAMETER AND VALIDATES IT21

FIGURE 8 INTERFACE DIAGRAM22

FIGURE 9 PARAMETER STATE DIAGRAM23

1. Introduction

This document uses Appendix 2 of the UNIT specification, where a series of interfaces to express the needs of Parameters exposed by a Unit Operation module are documented, and extends it to propose interfaces that can be used by other CAPE-OPEN packages (i.e. Numerical Solvers).

The document includes Use Cases, Use Cases Maps, Sequence and State Diagrams and IDL definitions for those interfaces.

2. Requirements

2.1 Textual requirements

Several Models will need to make use of a common definition for here is referred as “Public Parameters”. These are those variables exposed by the model in order that other human or software clients can get and set their values as well as specify their bounds, default values, etc...

This general requirement is a clear reflection of the already existing CAPE-OPEN Use Cases. Thus, in the Solvers Specification, the following Use Case is included:

Configure Numerical Code

Actors: <Solver Manager>

Priority:

Classification: <Solvers Use Cases>, <Simulation Context Use Cases>

Context:

Pre-conditions: <A solver has been selected>

Flow of events:

Basic Path

The Solver Manager asks the Solver for a list of its parameters: each of which will have a name, a type, a default value and a valid range (for real values).

It may then provide this list to the user to give him/her the opportunity to override the default values.

Post-conditions:

<Parameter list obtained>

<...>

Errors:

<Required parameter missing>

Uses:

Extends:

While, in Unit Operations the following requirement was included:

Set Unit Specific Data

Actors: <Flowsheet User>

Priority: <High >

Classification: <Unit Use Case >, <Specific Unit Use Case >, <Mixer/Splitter Use Case >.

Context: This Use Case is fulfilled by using the following methods: Edit, GetParameters, Count, Item, Mode, Value, OriginalSource

Pre-conditions:

<[Add Unit To Flowsheet] has been used and successfully passed>

<Unit has specific data whose values need to be specified>

Flow of events:

Basic Path:

The Flowsheet User asks the Simulator Executive to start a Flowsheet Unit's user interface. The Simulator Executive then asks the Flowsheet Unit to start its user interface. If the Flowsheet Unit does not have a user interface, the Simulator Executive either generates its own user interface, or tells the Flowsheet Builder that no user interface is available. If the Flowsheet Unit has a user interface, the Flowsheet Unit starts it. The Flowsheet Builder then supplies some or all the Flowsheet Unit's specific data through the user interface. When the Flowsheet Builder completes the data input, he terminates the user interface and the Flowsheet Unit's specific data are passed to the Flowsheet Unit. Note that the Simulator Executive is able to use [Get Unit Specific Data Names] to query the Flowsheet Unit for the list of parameters, so that it can construct a user interface for the Flowsheet Unit.

Post-conditions:

<Unit specific data have been specified>

Errors:

<Unit does not have a user interface and Simulator Executive can not create a user interface>

<Data values cannot be specified>

Uses: [Get Unit Specific Data Names]

Extends:

2.2 Use Cases

This section contains the set of General Purpose Use Cases (i.e. those that strictly belong to the Parameter Package), and Specific Purpose Use Cases (i.e. particular usage of the Parameter package that several client types, such as UNIT and Numerical Solvers can make).

Use Cases Categories

- ❑ General Purpose Use Cases.
- ❑ Specific Use Cases.

Use Cases Priorities

- ❑ **High.** Essential functionality. Functionality without which usability or performance might be seriously compromised
- ❑ **Low.** Desirable functionality that will improve performance. If this Use Case is not met, usability or acceptance can decrease.

2.2.1 Actors

- **Parameters Client.** Any human user of piece of software that will require accessing the functionalities provided by the Parameter Package. These can be:
 - **Solver Manager** A subsystem that handles the selection and configuration of solver “factory” components.
 - **Flowsheet Builder.** The person who sets up the flowsheet, the structure of the flowsheet, chooses thermo models and the unit operation models that are in the flowsheet. This person hands over a working flowsheet to the Flowsheet User. The Flowsheet Builder can act as a Flowsheet User.
 - **Flowsheet User.** The person who uses an existing flowsheet. This person will put new data into the flowsheet, rather than change the structure of the flowsheet.
- **Parameters Owner.** Any CAPE-OPEN component that needs to expose any of its internal variables as a Public Parameter (i.e. here referred simply as Parameter).

2.2.2 List of Use Cases

- UC-001 : Create/Configure List of Parameters
- UC-002 : Get List of Parameters
- UC-003 : Display List of Parameters
- UC-004 : Change Parameter
- UC-005 : Validate Parameter
- UC-006 : Validate Parameters Owner List of Parameters

2.2.3 Use Cases Map

Use Cases Map as well as individual descriptions of each Use Case have been produced by assigning high level actors (i.e. Parameters Owner and Parameters Client). This will facilitate re-using requirements and interface specifications across multiple CAPE-OPEN Packages.

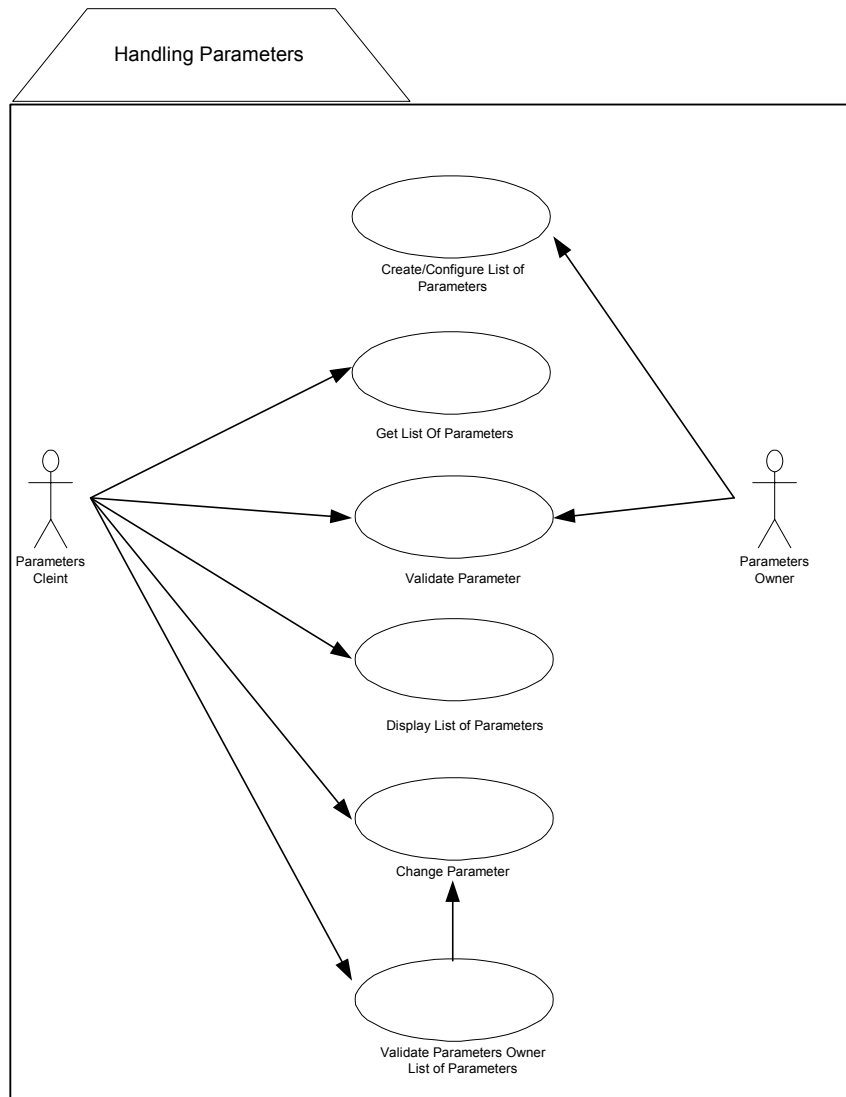


Figure 1 Parameter Use Cases Map

2.2.4 Use Cases

UC-001: CREATE/CONFIGURE LIST OF PARAMETERS

Actors: Parameters Owner

Priority: <High>

Classification: <General Purpose Use Cases >

Context:

Pre-conditions:

The Parameters Owner needs at least a Parameter to be exposed to external clients.

Flow of events:

The Parameters Owner creates a Parameter.

The Parameters Owner specifies the characteristics of the Parameter. This will imply specifying the type of Parameter (i.e. Real, Vector of Reals, Integer, etc), its default value, its dimensionality (e.g. in a reactor UNIT exposing length of the tube as a parameter this would be (1, 0, 0, 0, 0, 0) and its upper and lower bounds.

The dimensionality of the parameter for which this is the specification. The dimensionality represents the physical dimensional axes of this parameter. The dimensionality covers the 6 fundamental axes (length, mass, time, angle, temperature and charge).

The Parameters Owner repeats the above actions till the entire list of Parameters to expose has been created. Then The Parameters Owner create a list to store all the Parameters, so that it can be delivered afterwards when a Parameters Client request them.

Post-conditions:

Creation and Configuration of all Parameter succeeds.

Creation of the list of Parameters succeeds

Errors:

Parameter can not be created

Error while configuring any of the Parameter characteristics

Error while creating the list and/or storing the Parameters in the list

Uses:

Extends:

UC-002: GET LIST OF PARAMETERS

Actors: Parameters Client

Priority: <High>

Classification: <General Purpose Use Cases >

Context:

Pre-conditions:

The Parameters client has access to a Parameters Owner package (e.g. a UNIT)

Flow of events:

The Parameters Client asks the Parameters Owner to supply its list of Parameters, and this action is fulfilled by the Parameters.

The Parameters Client then can browse on the list of Parameters to obtain the desired Parameter for e.g. displaying, changing and/or validating its configuration (see [Change Parameter], [Display List of Parameters] and [Validate Parameter] Use Cases).

Post-conditions:

The Parameters Owner has delivered the list of Parameters successfully.

The Parameters Clients succeeds in getting the desired Parameter

Errors:

The Parameters Owner fails while delivering the Parameters list

The Parameters Client can not obtain the desired Parameter.

Uses:

Extends:

UC-003: DISPLAY LIST OF PARAMETERS

Actors: Parameters Client

Priority: <High>

Classification: <General Purpose Use Cases >

Context:

Pre-conditions:

The Parameters client has access to a list of Parameters from a Parameters Owner package (e.g. a UNIT), and therefore the [Get List of Parameters] Use Case has been completed successfully.

Flow of events:

The Parameters client obtains the Parameters information of its interest (e.g. value, upper and lower bounds, dimensionality etc...)

The Parameters client displays this information using Client private means (e.g. for a reporting sub-system this may imply generating a graphical user interface)

Post-conditions:

Information contained in each Parameter has been accessed correctly and it is displayed as required.

Errors:

Fails while accessing Parameters information.

Uses:

[Get List of Parameters]

Extends:

UC-004: CHANGE PARAMETER

Actors: Parameters Client

Priority: <High>

Classification: <General Purpose Use Cases >

Context:

Pre-conditions:

The Parameters client has access to the Parameter, and therefore the [Get List of Parameters] Use Case has been completed successfully.

Flow of events:

The Parameters Client modifies the configuration of the Parameter. This will typically imply changing the value, but also in some cases the original source (i.e. if a Parameter that was initially specified by the user is now calculated by any other external client, e.g. an optimiser).

After completion of these tasks the Parameters Client may invoke Validate Parameter to insure the performed changes are acceptable.

Post-conditions:

The Parameter has been changed successfully.

Errors:

Error while changing any of the configuration details of the Parameter

Uses:

Extends:

UC-005: VALIDATE PARAMETER

Actors: Parameters Client and Parameters Owner

Priority: <High>

Classification: <General Purpose Use Cases >

Context:

Pre-conditions:

The Parameters client has access to the Parameters, and therefore the [Get List of Parameters] Use Case has been completed successfully.

Flow of events:

Typically this Use Case will be executed after [Change Parameter] in order to insure that the changes performed in the Parameter are allowed.

Validation of a Parameter can be performed as a User input level (e.g. checking that a given value is within its allowed bounds, etc), or it may imply a more sophisticated validation using particular criteria of each Parameter (e.g. the Parameter may consult its owner whether the current value is permitted at that particular moment)

Post-conditions:

Validation succeeds

Errors:

Validation fails

Uses:

Extends:

UC-006: VALIDATE PARAMETERS OWNER LIST OF PARAMETERS

Actors: Parameters Client

Priority: <High>

Classification: <General Purpose Use Cases >

Context:

Pre-conditions:

The Parameters client has access to the Parameters Owner.

Flow of events:

This Use Case represents a generalisation of the UNIT Use Case [Check Unit Specific Data], and it is applicable to any Parameters Owner.

The Parameters Client request from a Parameters Owner to validate its Parameters. The Parameters Owner will make use of the Use Case [Validate Parameter] to check that each one of its Parameters is Valid.

Post-conditions:

Validation succeeds

Errors:

Validation fails

Uses:

Extends:

3. Analysis and Design

3.1 Overview

There are really two different parts below the concept of Parameter, each corresponding to a different client need. The first part is a fixed, static aspect that describes the Parameter, such as a type, name, description, dimensionality and so on. This is proposed to be used (explicitly stated in the UNIT case) to assist the User of the simulation in deciding what value to give the Parameter. This could be for the client to dynamically build a GUI widget (or set of widgets) to present to the User, or, using some other idiom, guide the User in selecting the Parameter value. It is therefore quite distinct from the actual value of the Parameter itself. This is something that is expected to change quite frequently both within and outside of the Component that needs it (for example by a controller, optimiser or other unit). Therefore there is a behavioural distinction which should be reflected in the interfaces for these two parts.

From an object modelling view we would in UML state that a Parameter *has* a Specification and represent this as below.

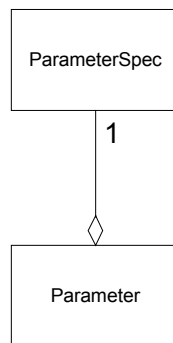


Figure 2 Conceptual Overview of Parameter (A)

Clearly there are different *types* of Parameter such as doubles and longs, in addition to non-numerical types such as Boolean and lists of string options. Therefore some further refinement of the object model can be made so as to reflect this as shown below.

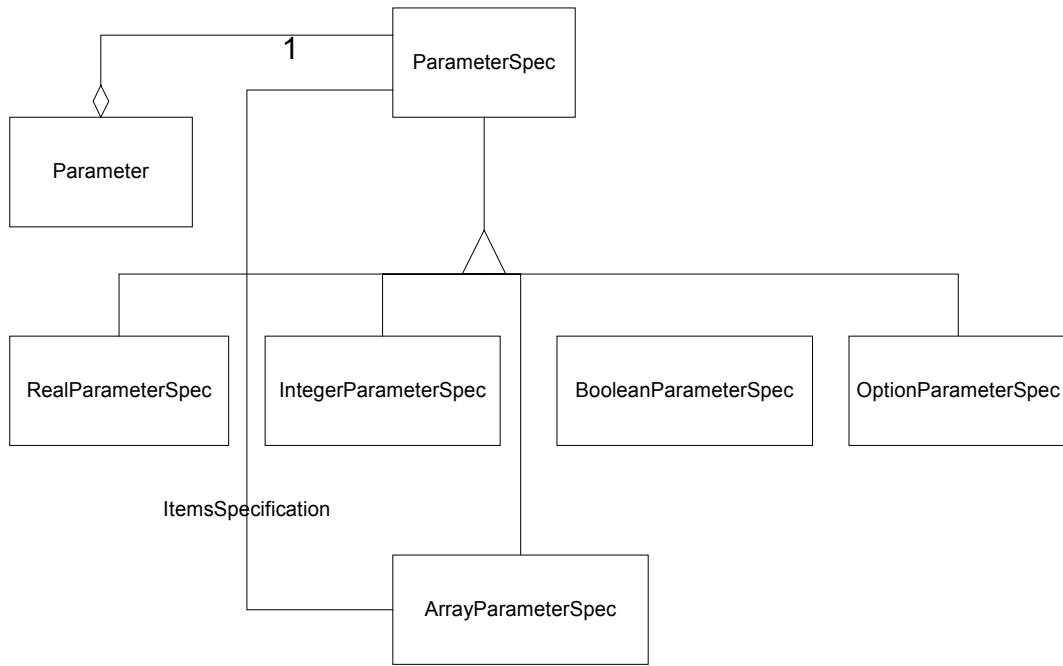


Figure 3 Conceptual Overview of Parameter (B)

3.2 Sequence diagrams

SQ-001: CREATING/CONFIGURING LIST OF PARAMETERS

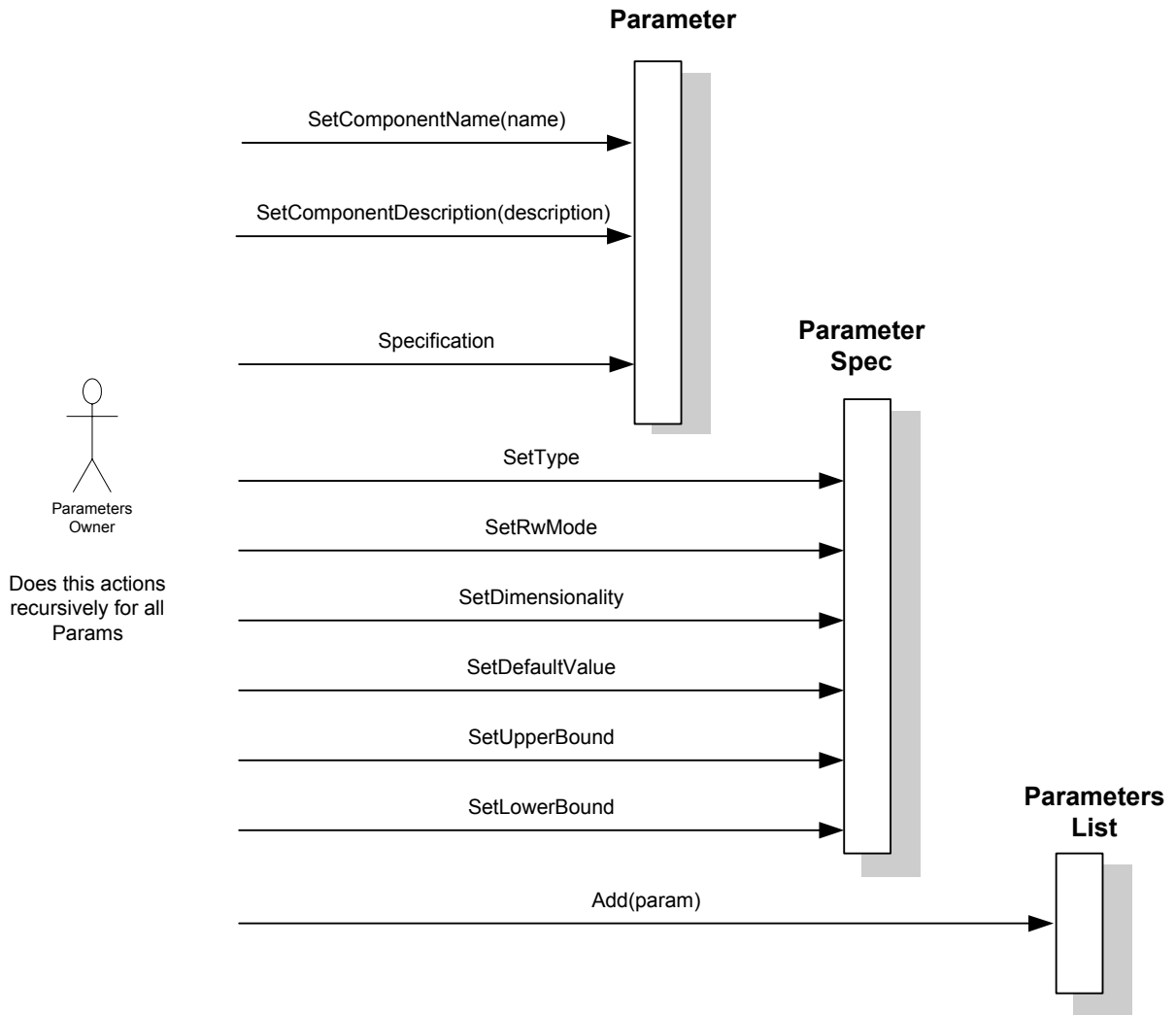


Figure 4 Creating/Configuring List of Parameters

SQ-002: GETTING LIST OF PARAMETERS

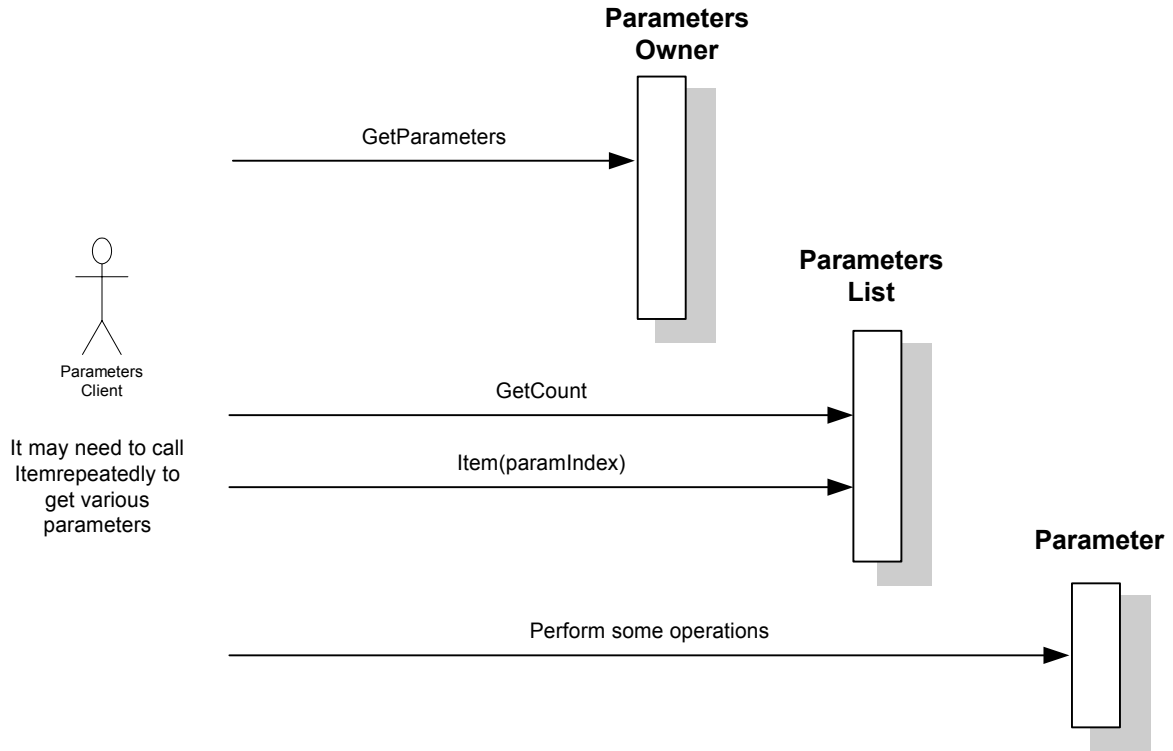


Figure 5 Getting List of Parameters

SQ-003: DISPLAYING LIST OF PARAMETERS

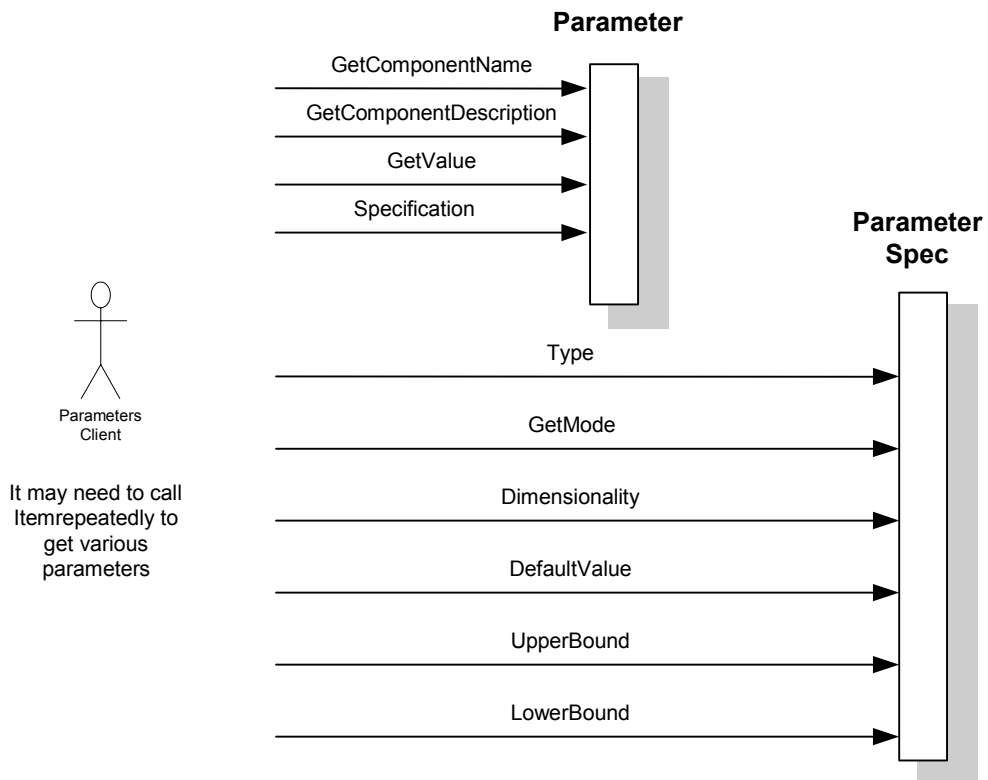


Figure 6 Displaying List of Parameters

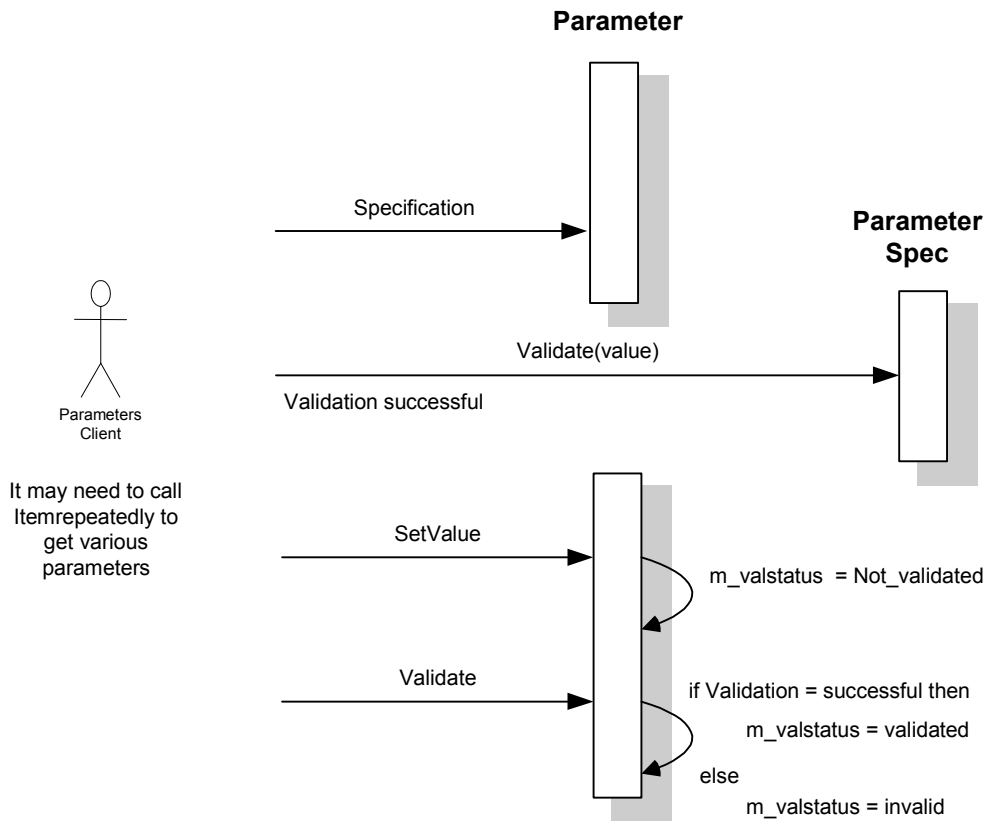


Figure 7 Parameters Client changes parameter and validates it

3.3 Interface diagrams

From an interface perspective, the separation of behaviours can be accommodated by the introduction of a basic interface for a Parameter Specification, ICapeParameterSpec, together with specialised interfaces for the various types, such as ICapeRealParameterSpec. Then for the value aspect of the Parameter, a single ICapeParameter interface can be utilised which treats the value as a CapeVariant data member. Thus the interface diagram for the interfaces is shown below.

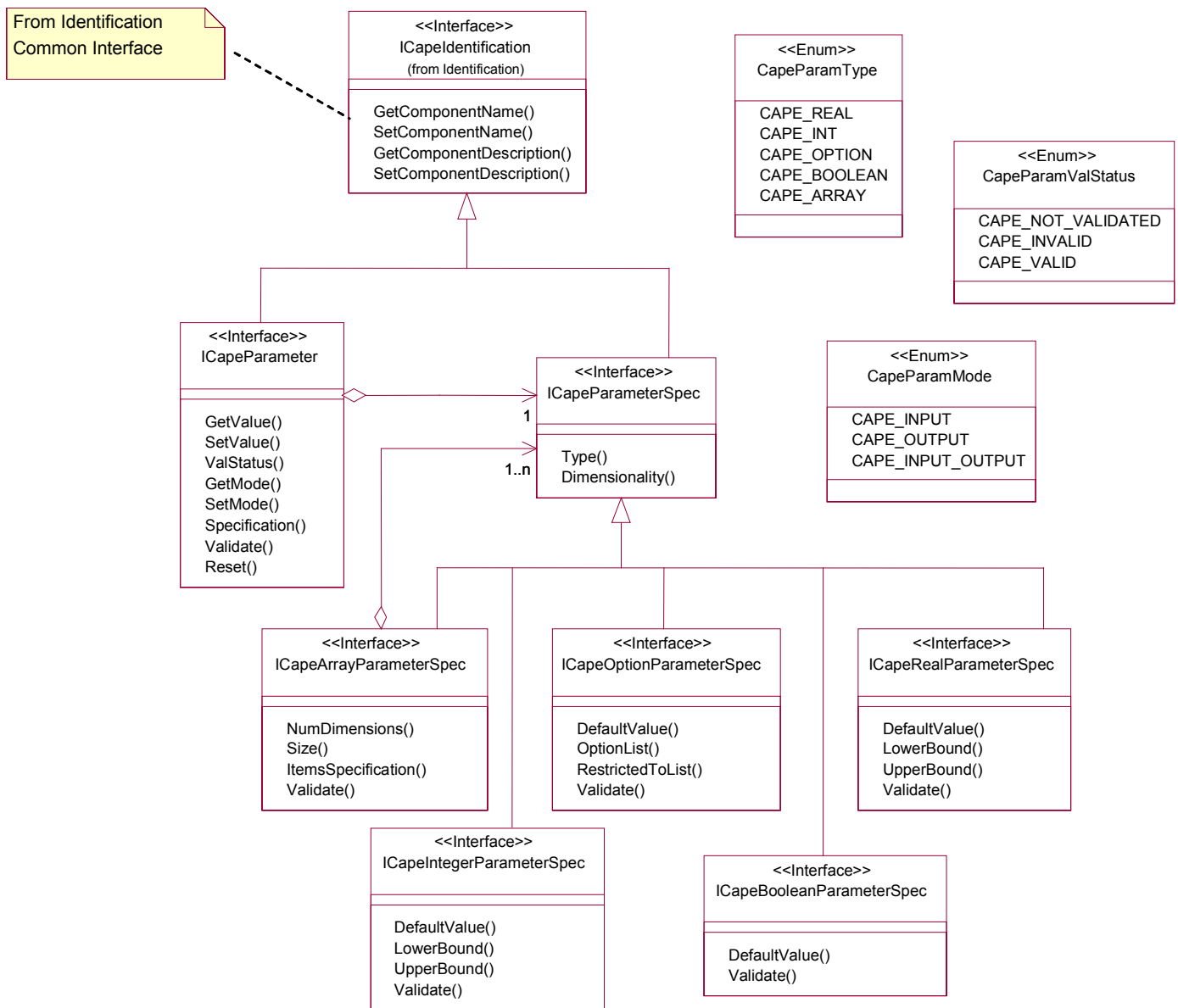


Figure 8 Interface diagram

There is no proposed way of for a common parameter storage mechanism. This is left to the decision of each CAPE-OPEN package. Thus, for instance in UNIT parameters are grouped in a specific entity named “Parameters Collection” that exposes an agreed CAPE-OPEN interfaces (ICapeUnitCollection). Other packages may decide grouping parameters in arrays. For the former, a general ICapeCollection interface could be used (has been proposed to M&T as a new Common Interface). For the latter, a sequence of CapeInterface’s should be used.

3.4 State diagrams

ST-001: PARAMETER STATE DIAGRAM

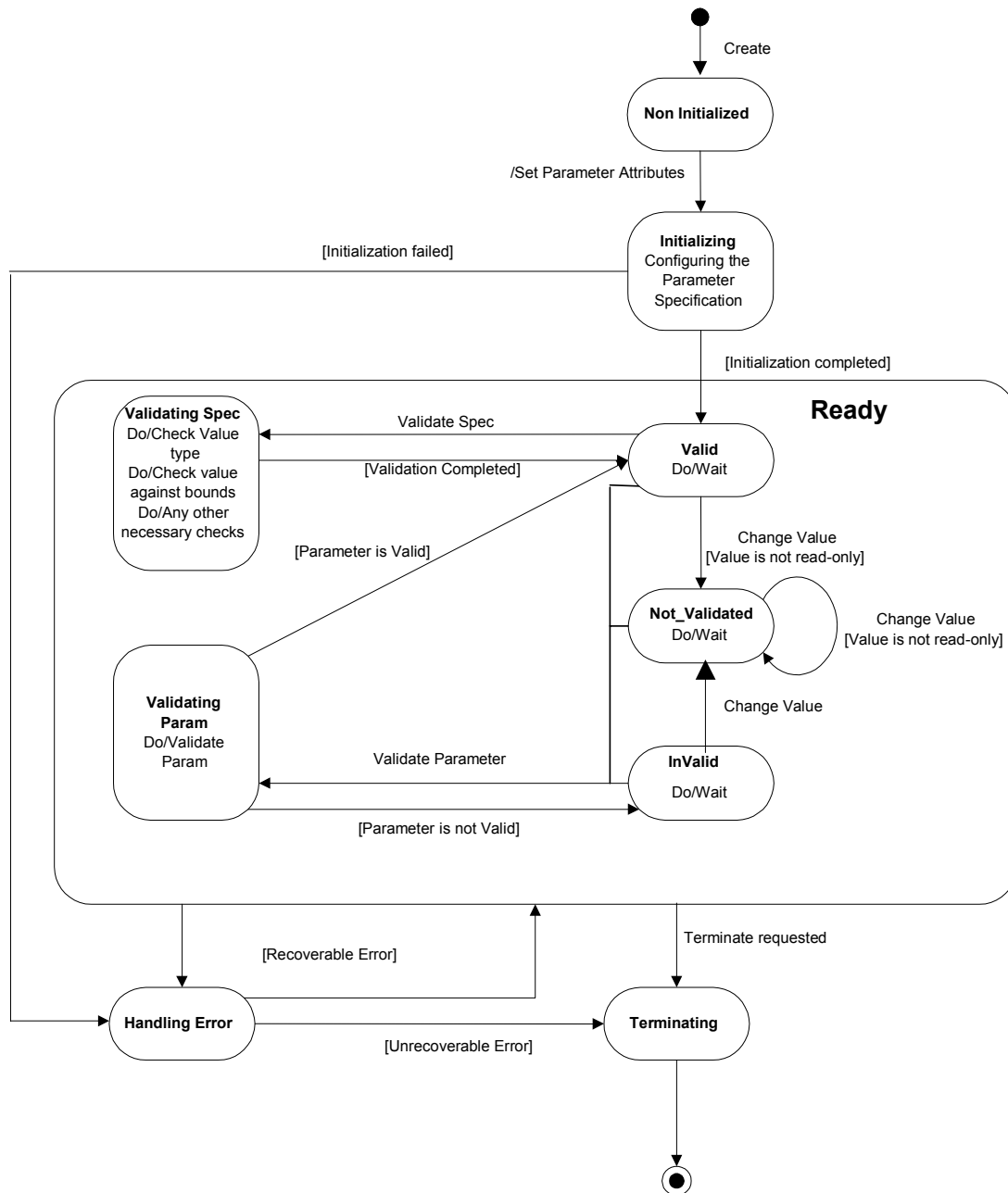


Figure 9 Parameter State Diagram

Note that the states outside the Ready super-state may not be visible states for a Parameters Client. Thus, creation and initialisation of the parameter will only be a visible state for the Parameters Owner, similarly to the Terminating State.

3.5 Other diagrams

None.

3.6 Interfaces descriptions

3.6.1 ICapeParameter

Interface Name	ICapeParameter
Method Name	Specification
Returns	CapeInterface

Description

Gets the specification of the parameter. The Get method returns the specification as an interface to the correct specification type.

Arguments

None

Errors:

ECapeUnknown

Interface Name	ICapeParameter
Method Name	ValStatus
Returns	CapeParamValStatus

Description

Gets the flag to indicate parameter validation status. It has three possible values:

- (i) notValidated(CAPE_NOT_VALIDATED): its validate() method has not been called after the last time that its value had been changed.
- (ii) invalid(CAPE_INVALID): the last time that its validate() method was called it returned false.
- (iii) valid(CAPE_VALID): the last time that its validate() method was called it returned true.

Arguments

None

Errors

ECapeUnknown

Interface Name	ICapeParameter
Method Name	GetMode
Returns	CapeParamMode

Description

Gets the mode in which the unit will interpret the parameter. It allows the following values:

- (i) Input (CAPE_INPUT): the Unit(or whichever owner component) will use its value to calculate.
- (ii) Output (CAPE_OUTPUT): the Unit will place in the parameter a result of its calculations.
- (iii) Input-Output (CAPE_INPUT_OUTPUT): the user inputs an initial estimation value and the user outputs a calculated value.

The tricky thing is that the Unit could allow the client to change the value of this parameter mode. Imagine that a UO has 5 parameters, but is able to calculate having the value of only 2 of them. The user will set the mode of the parameter considered as independent variables to in, and out for the other ones. Most parameters won't allow this parameter to be changed, because their parameter will only be input or output

Arguments

None

Errors:

ECapeUnknown

Interface Name	ICapeParameter
Method Name	SetMode
Returns	--

Description

Sets the mode in which the unit will interpret the parameter. It allows the following values:

- (i) Input (CAPE_INPUT): the Unit(or whichever owner component) will use its value to calculate.
- (ii) Output (CAPE_OUTPUT): the Unit will place in the parameter a result of its calculations.
- (iii) Input-Output (CAPE_INPUT_OUTPUT): the user inputs an initial estimation value and the user outputs a calculated value.

The tricky thing is that the Unit could allow the client to change the value of this parameter mode. Imagine that a UO has 5 parameters, but is able to calculate having the value of only 2 of them. The user will set the mode of the parameter considered as independent variables to in, and out for the other ones. Most parameters won't allow this parameter to be changed, because their parameter will only be input or output

Arguments

Name	Type	Description
[in] mode	CapeLong	The flag that indicates whether the Unit should take the parameter as input or as output

Errors:

ECapeUnknown, ECapeInvalidArgument

Interface Name	ICapeParameter
Method Name	Validate
Returns	CapeBoolean

Description

Checks if the value of the parameter is valid in specification terms and in any other terms that are specialised.

Arguments

Name	Type	Description
[out] Message	CapeString	Message conveying information regarding the validation

Errors:

ECapeUnknown

Interface Name	ICapeParameter
Method Name	GetValue
Returns	CapeVariant

Description

Gets the value of this parameter. Passed as a CapeVariant that should be the same type as the Parameter type.

Arguments

None

Errors

ECapeUnknown

Interface Name	ICapeParameter
Method Name	SetValue
Returns	--

Description

Sets the value of this parameter. Passed as a CapeVariant that should be the same type as the Parameter type, if not this is an error (invalid argument).

Arguments

Name	Type	Description
[in] value	CapeVariant	The value of this parameter.

Errors:

ECapeUnknown, ECapeInvalidArgument

Interface Name	ICapeParameter
Method Name	Reset
Returns	--

Description

Sets the value of the parameter to its default value.

Arguments

None

Errors

ECapeUnknown

3.6.2 ICapeParameterSpec

Interface Name	ICapeParameterSpec
Method Name	Type
Returns	CapeParamType

Description

Gets the type of the parameter for which this is a specification: real (CAPE_REAL), integer(CAPE_INT), option(CAPE_OPTION), boolean(CAPE_BOOLEAN) or array(CAPE_ARRAY)

Arguments

None

Errors

ECapeUnknown

Interface Name	ICapeParameterSpec
Method Name	Dimensionality
Returns	CapeVariant

Description

Gets the dimensionality of the parameter for which this is the specification. The dimensionality represents the physical dimensional axes of this parameter. It is expected that the dimensionality must cover at least 6 fundamental axes (length, mass, time, angle, temperature and charge).

A possible implementation could consist in being a constant length array vector that contains the exponents of each basic SI unit, following directives of SI-brochure (from <http://www.bipm.fr/>) So if we agree on order <m kg s A K,>... velocity would be <1,0,-1,0,0,0>: that is $m^1 * s^{-1} = m/s$

We have suggested to the CO Scientific Committee to use the SI base units plus the SI derived units with special symbols (for a better usability and for allowing the definition of angles).

Table 1. SI base units

Base quantity	SI base unit	
	Name	Symbol
length	metre	m
mass	kilogram	kg
time	second	s
electric current	ampere	A
thermodynamic temperature	kelvin	K
amount of substance	mole	mol
luminous intensity	candela	cd

Table 3 SI derived units with special names and symbols

Derived quantity	Name	Symbol	Expressed in terms of other SI units	
plane angle	radian ^(a)	rad		
solid angle	steradian ^(a)	sr ^(c)		
frequency	hertz	Hz		
force	newton	N		
pressure, stress	pascal	Pa	N/m ²	
energy, work, quantity of heat	joule	J	N · m	
power, radiant flux	watt	W	J/s	
electric charge, quantity of electricity	coulomb	C		
electric potential difference, electromotive force	volt	V	W/A	
capacitance	farad	F	C/V	
electric resistance	ohm	Ω	V/A	
electric conductance		siemens	S	A/V
magnetic flux		weber	Wb	V · s
magnetic flux density		tesla	T	Wb/m ²
inductance		henry	H	Wb/A
Celsius temperature		degree Celsius ^(d)	°C	
luminous flux		lumen	lm	cd · sr ^(c)
illuminance		lux	lx	lm/m ²
activity (referred to a radionuclide)		becquerel	Bq	
absorbed dose, specific energy (imparted), kerma		gray	Gy	J/kg
dose equivalent, ambient dose equivalent, directional dose equivalent, personal dose equivalent, organ equivalent dose		sievert	Sv	J/kg

Arguments

None

Errors

ECapeUnknown

3.6.3 ICapeRealParameterSpec

Interface Name	ICapeRealParameterSpec
Method Name	DefaultValue
Returns	CapeDouble

Description

Gets the default value of the real valued parameter for which this is the specification.

Arguments

None

Errors

ECapeUnknown

Interface Name	ICapeRealParameterSpec
Method Name	LowerBound
Returns	CapeDouble

Description

Gets the lower bound of the real valued parameter for which this is the specification.

Arguments

None

Errors

ECapeUnknown

Interface Name	ICapeRealParameterSpec
Method Name	UpperBound
Returns	CapeDouble

Description

Gets/Sets the upper bound of the real valued parameter for which this is the specification.

Arguments

None

Errors

ECapeUnknown

Interface Name	ICapeRealParameterSpec
Method Name	Validate
Returns	CapeBoolean

Description

Validates a real value against its specification. It returns a flag to indicate the success or failure of the validation together with a text message which can be used to convey the reasoning to the client/user.

The flag is True for success and False for failure.

Arguments

Name	Type	Description
[in] value	CapeDouble	The value of the parameter to validate
[out] message	CapeString	Message conveying information regarding the validation

Errors

ECapeUnknown, ECapeInvalidArgument

3.6.4 ICapeArrayParameterSpec

Interface Name	ICapeArrayParameterSpec
Method Name	NumDimensions
Returns	CapeLong

Description

Gets the number of dimensions of the array contained in this Parameter.

Arguments

None

Errors

ECapeUnknown

Interface Name	ICapeArrayParameterSpec
Method Name	Size
Returns	CapeArrayLong

Description

Gets the size of each one of the dimensions of the array.

Arguments

None

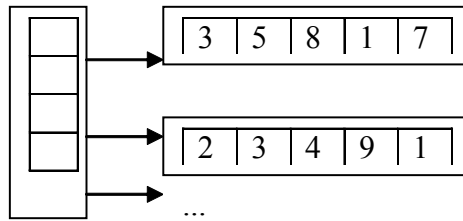
Errors

ECapeUnknown

Interface Name	ICapeArrayParameterSpec
Method Name	ItemsSpecification
Returns	CapeArrayInterface

Description

Gets an array of the specifications of each of the items in the value of a parameter. The Get method returns an array of interfaces to the correct specification type (ICapeRealParameterSpec, ICapeOptionParameterSpec, ...) Note that it is also possible, for example, to configure an array of arrays of integers (see diagram bellow), which would a similar but not identical concept to a two-dimensional matrix of integers



Arguments

None

Errors

ECapeUnknown

Interface Name	ICapeArrayParameterSpec
Method Name	Validate
Returns	CapeArrayBoolean

Description

Validates an array against its specification. It returns a flag to indicate the success or failure of the validation together with a text message which can be used to convey the reasoning to the client/user.

Arguments

Name	Type	Description
[in] value	CapeVariant	The value of the parameter to validate
[out] message	CapeArrayString	Message conveying information regarding the validation

Errors

ECapeUnknown, ECapeInvalidArgument

3.6.5 ICapeIntegerParameterSpec

Interface Name	ICapeIntegerParameterSpec
Method Name	DefaultValue
Returns	CapeLong

Description

Gets/Sets the default value of the integer valued parameter for which this is the specification.

Arguments

None

Errors

ECapeUnknown

Interface Name	ICapeIntegerParameterSpec
Method Name	LowerBound
Returns	CapeLong

Description

Gets the lower bound of the integer valued parameter for which this is the specification.

Arguments

None

Errors

ECapeUnknown

Interface Name	ICapeIntegerParameterSpec
Method Name	UpperBound
Returns	CapeLong

Description

Gets the upper bound of the integer valued parameter for which this is the specification.

Arguments

None

Errors

ECapeUnknown

Interface Name	ICapeIntegerParameterSpec
Method Name	Validate
Returns	CapeBoolean

Description

Validates an integer value against its specification. It returns a flag to indicate the success or failure of the validation together with a text message which can be used to convey the reasoning to the client/user.

Arguments

Name	Type	Description
[in] value	CapeLong	The value of the parameter to validate
[out] message	CapeString	Message conveying information regarding the validation

Errors

ECapeUnknown, ECapeInvalidArgument

3.6.6 ICapeOptionParameterSpec

Interface Name	ICapeOptionParameterSpec
Method Name	DefaultValue
Returns	CapeString

Description

Gets the default value that the parameter will take. Since it does not indicate an index of the option list, it is allowed to set any default value when the optionList is empty.

Arguments

None

Errors

ECapeUnknown

Interface Name	ICapeOptionParameterSpec
Method Name	OptionList
Returns	CapeArrayString

Description

Gets the list of possible values that the parameter may take. If attribute restrictedToList is false, then the parameter may also accept other values.

Arguments

None

Errors

ECapeUnknown

Interface Name	ICapeOptionParameterSpec
Method Name	RestrictedToList
Returns	CapeBoolean

Description

States whether the parameter allows to set values different to the ones contained in the option list.

Arguments

None

Errors

ECapeUnknown

Interface Name	ICapeOptionParameterSpec
Method Name	Validate
Returns	CapeBoolean

Description

Validates whether the string argument is accepted by the parameter as a valid value. It returns a flag to indicate the success or failure of the validation together with a text message which can be used to convey the reasoning to the client/user.

Arguments

Name	Type	Description
[in] value	CapeString	The value of the parameter to validate
[out] message	CapeString	Message conveying information regarding the validation

Errors

ECapeUnknown, ECapeInvalidArgument

3.6.7 ICapeBooleanParameterSpec

Interface Name	ICapeBooleanParameterSpec
Method Name	DefaultValue
Returns	CapeBoolean

Description

Gets the default value that the parameter will take.

Arguments

None

Errors

ECapeUnknown

Interface Name	ICapeBooleanParameterSpec
Method Name	Validate
Returns	CapeBoolean

Description

Validates whether the argument is accepted by the parameter as a valid value. It returns a flag to indicate the success or failure of the validation together with a text message which can be used to convey the reasoning to the client/user.

Arguments

Name	Type	Description
[in] value	CapeBoolean	The value of the parameter to validate
[out] message	CapeString	Message conveying information regarding the validation

Errors

ECapeUnknown, ECapeInvalidArgument

3.7 Scenarios

4. Interface Specifications

4.1 COM IDL

// You can get these instructions in Parameter.idl file from CAPE-OPENv1-0-0.zip

4.2 CORBA IDL

// You can get these instructions in CAPE-OPENv1-0-0.idl within the
CAPEOPEN100::Common::Parameter module

5. Notes on the interface specifications

5.1 Changes with respect to Appendix 2 of the UNIT specification

All attributes from all the ICapeParameterSpec (including realSpec, integerSpec,...) are now read-only (ie: their property sets have been removed). It didn't make any sense that a client could modify the specification of a unit parameter, since probably the Unit would not be prepared to interpret the new values if its specification had been changed.

A new interface has been added: ICapeArrayParameterSpec. This interface is intended to improve and facilitate the usability of the parameter package when the parameter value is of an array of real or integer values.

The previous interfaces ICapeRealParameterSpec and ICapeIntegerParameterSpec, can be used to represent multidimensional variables. Nevertheless, it was impossible to specify different bounds or default values for each one of the items in the array. Furthermore, the Validate method was not designed to operate with arrays, but with single real or integer values

ICapeArrayParameterSpec includes the functionality of ICapeRealParameterSpec or ICapeIntegerParameterSpec (i.e. upperBound, lowerBound and defaultValue.) with a reference to an array of specification interfaces, one for each element of the array (although a single component could be used as specification to all the elements). ICapeArrayParameterSpec adds the attributes NumDimensions and Size. These two additional methods are to facilitate clients knowing the characteristics of the array stored within the Parameter, for e.g. displaying correctly the Parameter in a GUI facility

Attributes ICapeParameterSpec.mode and ICapeParameter.OriginalSource have been merged into a new ICapeParameter.mode, since it was not clear the difference between this two attributes. Mode has only two possible values:

- (i) Input: the Unit(or whichever owner component) will use its value to calculate.
- (ii) Output: the Unit will place in the parameter a result of its calculations.

The new thing is that the Unit could allow the client to change the value of the parameter mode. Imagine that a UO has 5 parameters, but is able to calculate having the value of only 2 of them. The user will set the mode of the parameters considered as independent variables to in, and out for the other ones. Most parameters won't allow this parameter to be changed

6. Prototypes implementation

6.1 Creating and Configuring the list of Parameters (Actor: Parameters Owner)

This code excerpt shows how a UNIT Component creates a collection of parameters and initialises their specifications.

```
Dim myVar As CAPEOPEN.ICapeParameter
Dim myVarSpec As CAPEOPEN.ICapeParameterSpec
Dim myRealVarSpec As CAPEOPEN.ICapeRealParameterSpec
Dim myArrayRealVarSpec As CAPEOPEN.ICapeArrayParameterSpec

' Accessing the private interface of the collection of public variables
' and configuring it...

' Creating variables...(m_publicVariables is the collection of variables of this Unit '
Operation. Note in this case the parameters owner is using a private interface of
' the collection of parameters to create each one of its individual parameters)

Set m_heatInput = m_publicVariables.Add("Heat Input(kJ/h)")
Set m_pressureDrop = m_publicVariables.Add("Pressure Drop (kPa)")
Set m_splitFactors = m_publicVariables.Add("Split Factors")

' This section configures each one of the parameters, by accessing the appropriate
' interface, and specifying the required values (i.e. ValueSource, Dimensionality,
' Mode, Type, Default Value and Bounds)

c Configuring information regarding variable m_heatInput...

Set myVar = m_heatInput ' Accessing the ICapeParameter interface
Set myVarSpec = myVar.Specification ' Accessing the ICapeParameterSpec interface
Set myRealVarSpec = myVarSpec ' Accessing the ICapeRealParameterSpec interface

myVarSpec.Dimensionality = "" ' Not specified
myVarSpec.mode = CAPEOPEN.CAPE_INPUT ' Read/Write
myVarSpec.Type = CAPEOPEN.CAPE_REAL ' Double

myRealVarSpec.DefaultValue = 0#
myRealVarSpec.LowerBound = -100000#
myRealVarSpec.UpperBound = 100000#

'Configuring information regarding variable m_pressureDrop...

Set myVar = m_pressureDrop
Set myVarSpec = myVar.Specification
Set myRealVarSpec = myVarSpec

myVarSpec.Dimensionality = ""
myVarSpec.mode = CAPEOPEN.CAPE_INPUT
myVarSpec.Type = CAPEOPEN.CAPE_REAL

myRealVarSpec.DefaultValue = 0#
myRealVarSpec.LowerBound = 0#
myRealVarSpec.UpperBound = 10000#

' Configuring information regarding variable m_splitFactors...

Set myVar = m_splitFactors
Set myVarSpec = myVar.Specification
Set myArrayRealVarSpec = myVarSpec

myVarSpec.Dimensionality = ""
myVarSpec.mode = CAPEOPEN.CAPE_INPUT
myVarSpec.Type = CAPEOPEN.CAPE_REAL
```

```

Dim defaultsArray(0 To 0) As Double
Dim itemsArray(0 To 0) As Double
Dim itemsVnt as Variant
Dim defaultsVnt as Variant

itemsArray(0) = 3&
itemsVnt = itemsArray

defaultsArray(0) = 0.333
defaultsArray(1) = 0.333
defaultsArray(2) = 0.333

defaultsVnt = defaultsArray

myArrayRealVarSpec.NumDimensions = 1           ' it is a vector
myArrayRealVarSpec.Size = itemsArray         ' with 3 elements
myArrayRealVarSpec.DefaultValues = defaultsVnt
myArrayRealVarSpec.LowerBounds = 0#
myArrayRealVarSpec.UpperBounds = 10000#

' More code goes here...

```

6.2 Validating Parameters (Actor: Parameters Client)

This code excerpt shows how a client of a Parameters Owner gets access to the list of parameters (in this example stored as a collection) and Validates each one of the individual parameters.

```

Dim myParams As CAPEOPEN.ICapeUnitCollection
Dim myParam As CAPEOPEN.ICapeParameter
Dim myParamIdent As CAPEOPEN.ICapeIdentification
Dim myParamSpec As CAPEOPEN.ICapeParameterSpec
Dim iParam As Integer
Dim numParams As Long
Dim ValueToPut As Variant
Dim msg As String

' The Parameters Owner...
' ValueToPut is the variable that represent the value to supply to the Parameter

' The Parameters Owner...
Set myUnit = capeUnit

' Gets the parameters collection from the Parameters Owner...
Set myParams = myUnit.Parameters

' Number of Parameters in the collection
numParams = myParams.Count

For iParam = 1 To numParams

    ' Gets each Parameter, and connects to different interfaces...
    Set myParam = myParams.Item(iParam)

' Accessing the ICapeIdentification interface to find out the parameter name
    Set myParamIdent = myParam

    ' Accessing the ICapeParameterSpec interface...
    Set myParamSpec = myParam.Specification

' If the Parameter is ReadWrite, the value can be changed
    If myParamSpec.mode = CAPEOPEN.CAPE_INPUT_OUTPUT Then

        ' Puts the value
        myParam.value = ValueToPut

    ' Validates the Param

```

```
        if myParam.Validate(msg) = False then
MsgBox "Warning: Parameter " & iParam & " is not valid (" &msg & ")"
        End If

        End If

Next iParam
```

7. Specific Glossary Terms

Parameters Owner. Any CAPE-OPEN Component that exposes one or more CAPE-OPEN Parameters. These may be e.g. CAPE-OPEN Unit Operation Components

Parameters Client. Any external actor that request Parameters owned by a CAPE-OPEN Parameters Owner. These may be e.g. CAPE-OPEN Unit Operation Components

8. Bibliography

9. Appendices