

# CAPE-OPEN

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

---

## Unit Operations

---



[www.colan.org](http://www.colan.org)

---

## ARCHIVAL INFORMATION

---

Filename	CO Unit Operations v6.25.doc
Authors	CO-LaN consortium
Status	Released for RFC November 2011
Date	November 2011
Version	version 1.0
Number of pages	103
Versioning	version 1.0.6.25
Additional material	
Web location	
Implementation specifications version	
Comments	Contributors to this document have been Peter Banks (BP), Peter Edwards (DuPont), Juan Carlos Rodriguez (DuPont), Richard Martin (QuantiSci), Mike Williams (QuantiSci), Sergio Cebollero (Hyprotech), Michael Halloran (Aspentech), Sergi Sama (Hyprotech), Christophe Poulain (Aspentech), Jörg Köller (RWTH-I5), Bill Johns (QuantiSci), Bertrand Braunschweig (IFP), Daniel Pinol (Hyprotech), Costas Pantelides (Imperial College), Ben Keeping (Imperial College), Lars von Wedel (RWTH), Michael White (Aspentech), Nii Asante (QuantiSci), Christian Kulhmann (QuantiSci), David Smith (DuPont), Malcolm Woodman (BP), Ron Chartres (BP), Richard Baur (Shell), Jasper van Baten (AmsterCHEM), Michel Pons (CO-LaN), Didier Paen (RSI), Tom Williams (Process Systems Enterprise).

---

## 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 © 2011 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 sellers 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.

Adobe Acrobat is a registered trademark of Adobe Corporation.

---

## SUMMARY

---

This document, which was produced originally by the Unit Operations Work Package (UNIT) of the CAPE-OPEN project, describes version 1.0 of the Interface Specifications for the Unit Operation component of the CAPE-OPEN standards. This corresponds to the GRP1 sub-task described in the CAPE-OPEN project work plan<sup>1</sup>. Treatment of sub-task GRP2, which deals with steady-state and dynamic equation-oriented simulation, is also included, although this has not been prototyped.

The document starts with a text description of the requirements identified for an open unit operation software component. This is then expressed in Unified Modelling Language and developed into a specification of the interfaces necessary for a CAPE-OPEN unit operation software component to plug into a compliant flowsheet simulator. On the CO-LaN website, the implementation specifications are available as COM IDL, COM type library and CORBA IDL. The final part of the document deals with the extension of these interfaces to handle open steady-state and dynamic equation-oriented simulation.

---

## **ACKNOWLEDGEMENTS**

---

The authors of this document wish to acknowledge the particular contribution of the various people whose names appear in the archival section, as well as the more general contribution and encouragement from many other colleagues. Without their help, thoughts and technical expertise, we would not have been able to complete this project. We would also like to thank the companies and organisations involved for supporting the project and providing the significant resources needed to carry out the work.

---

# CONTENTS

---

<b>CAPE-OPEN DOCUMENT ROADMAP.....</b>	<b>10</b>
<b>1. INTRODUCTION.....</b>	<b>11</b>
<b>2. REQUIREMENTS.....</b>	<b>12</b>
2.1 USER REQUIREMENTS FOR AN OPEN FLOWSHEET UNIT COMPONENT .....	12
2.1.1 <i>Setting the scene</i> .....	12
2.1.2 <i>Architecture</i> .....	13
2.1.3 <i>CO Objects Present in a Compliant Simulator</i> .....	14
2.1.4 <i>Communications between a Unit and a Simulator Executive</i> .....	15
2.1.5 <i>Notes on Unit persistence and configuration</i> .....	16
2.1.6 <i>Conceptual Examples</i> .....	17
2.1.7 <i>Notes on Conceptual Operation</i> .....	21
2.1.8 <i>Desirable Characteristics for the CO Interface</i> .....	23
2.1.9 <i>Thermodynamic Properties</i> .....	23
2.1.10 <i>Simultaneous Modular/Optimization</i> .....	24
2.1.11 <i>Granularity</i> .....	25
2.2 USE CASES.....	25
2.2.1 <i>Use Case Categories</i> .....	26
2.2.2 <i>Use Case Priorities</i> .....	26
2.2.3 <i>Actors</i> .....	26
2.2.4 <i>List of Use Cases</i> .....	27
2.2.5 <i>Use Cases maps</i> .....	28
2.2.6 <i>Use Cases</i> .....	34
<b>3. ANALYSIS AND DESIGN .....</b>	<b>65</b>
3.1 OVERVIEW .....	65
3.2 SEQUENCE DIAGRAMS.....	65
3.2.1 <i>Retrieve Flowsheet</i> .....	65
3.2.2 <i>Add Unit to Flowsheet</i> .....	66
3.2.3 <i>Evaluate Unit</i> .....	67
3.2.4 <i>Reporting</i> .....	68
3.3 INTERFACE DIAGRAMS .....	68
3.4 STATE DIAGRAMS.....	69
3.5 OTHER DIAGRAMS.....	72
3.6 INTERFACES DESCRIPTIONS .....	72
3.6.1 <i>ICapeUnit</i> .....	73
3.6.2 <i>ICapeUnitPort</i> .....	77
3.6.3 <i>ICapeUnitReport</i> .....	82
3.6.4 <i>Interface ICapeUnitPortVariables</i> .....	86
3.7 SCENARIO .....	87
<b>4. INTERFACE SPECIFICATIONS .....</b>	<b>90</b>
<b>5. NOTES ON THE INTERFACE SPECIFICATIONS .....</b>	<b>91</b>
<b>6. PROTOTYPE IMPLEMENTATION.....</b>	<b>92</b>
<b>7. GUIDELINES ON VERSIONING.....</b>	<b>92</b>
<b>8. SPECIFIC GLOSSARY TERMS.....</b>	<b>94</b>
<b>9. BIBLIOGRAPHY .....</b>	<b>95</b>

9.1	PROCESS SIMULATION REFERENCES .....	95
9.2	COMPUTING REFERENCES .....	96
9.3	GENERAL REFERENCES .....	96
<b>10.</b>	<b>APPENDIX: EQUATION-ORIENTED SIMULATION .....</b>	<b>97</b>
10.1	INTRODUCTION .....	97
10.2	AN EQUATION-ORIENTED UNIT OPERATION .....	97
10.2.1	<i>Equations for a typical EO Unit Operation.....</i>	<i>97</i>
10.2.2	<i>The Equation Set Object (ESO).....</i>	<i>98</i>
10.2.3	<i>Operation of an EO simulator.....</i>	<i>98</i>
10.2.4	<i>Proposal .....</i>	<i>100</i>
10.2.5	<i>Interoperability of SM and EO Unit Operations .....</i>	<i>100</i>
10.2.6	<i>Other issues .....</i>	<i>101</i>
10.2.7	<i>Extra methods required for UNIT and NUMR .....</i>	<i>101</i>
10.2.8	<i>Mixer-splitter design scenario for steady-state EO Unit and Simulator .....</i>	<i>101</i>

---

## LIST OF FIGURES

---

FIGURE 2-1 SINGLE STAGE FROM AN OIL AND GAS SEPARATION SYSTEM.....	12
FIGURE 2-2 FLOWSHEET SIMULATOR SCHEMA.....	13
FIGURE 2-3 AREAS OF RESPONSIBILITY OF THE DIFFERENT WORK PACKAGES (APPROACH 1) .....	14
FIGURE 2-4 AREAS OF RESPONSIBILITY OF THE DIFFERENT WORK PACKAGES (APPROACH 2) .....	14
FIGURE 2-5 EXAMPLE OF INTERFACE OPERATION .....	17
FIGURE 2-6 RESETTING PUBLIC UNIT PARAMETERS.....	19
FIGURE 2-7 USE CASES ON SIMULATOR EXECUTIVE SYSTEM WITH "HUMAN" ACTORS INVOLVED .....	29
FIGURE 2-8 USE CASES ON FLOWSHEET UNIT SYSTEM BY "NON-HUMAN" ACTORS .....	30
FIGURE 2-9 USE CASES ON FLOWSHEET UNIT SYSTEM BY SIMULATOR EXECUTIVE AND ITS SUB-SYSTEMS .....	31
FIGURE 2-10 USE CASES ON FLOWSHEET UNIT SYSTEM BY FLOWSHEET UNIT.....	32
FIGURE 2-11 USE CASES ON THERMO SYSTEM BY FLOWSHEET UNIT .....	33
FIGURE 3-1 SQ-001 RETRIEVE FLOWSHEET .....	65
FIGURE 3-2 SQ-002 ADD UNIT TO FLOWSHEET.....	66
FIGURE 3-3 SQ-003 EVALUATE UNIT.....	67
FIGURE 3-4 SQ-004 SELECTING AND PRODUCING A REPORT .....	68
FIGURE 3-5 INTERFACE DIAGRAM FOR UNIT .....	68
FIGURE 3-6 PORT STATE DIAGRAM.....	69
FIGURE 3-7 INITIALIZATION STATE DIAGRAM.....	70
FIGURE 3-8 UNIT VALIDATION STATE DIAGRAM .....	71
FIGURE 3-9 COLLABORATION DIAGRAM.....	72
FIGURE 7-1 STATE CHART SHOWING THERMODYNAMIC SPECIFICATION USAGE FOR DIFFERENT IMPLEMENTED CATEGORY COMBINATIONS.....	93

---

## LIST OF TABLES

---

TABLE 2-1 FLOWSHEET CREATION .....	15
TABLE 2-2 FLOWSHEET AND UNIT CONFIGURATION.....	16
TABLE 2-3 FLOWSHEET SOLUTION AND UNIT CALCULATION.....	16
TABLE 2-4 FLOWSHEET AND UNIT REPORTING .....	16
TABLE 2-5 ENERGY CONSUMPTION AND PRODUCTION BY A UNIT .....	22
TABLE 2-6 PARAMETERS IN ENERGY OBJECTS .....	23

## CAPE-OPEN Document Roadmap

This document belongs to the documentation set of CAPE-OPEN interface specifications in its version 1.0. This imposes that for Microsoft COM Unit Operation components implementing the interfaces defined in this document, the CapeVersion registry key must be set at the value “1.0”.

This document is intended primarily for software engineers, who are interested in producing CAPE-OPEN compliant Unit Operation components.

All other readers need not go beyond **Section 2 Requirements**.

# 1. Introduction

This document describes the Interface Specifications for the Unit Operation component of the CAPE-OPEN standards.

This document was originally produced by the Unit Operations Work Package (Work Package 3) of the CAPE-OPEN project and the document develops the concepts introduced in the CAPE-OPEN Concepts Document<sup>2</sup>.

The Unit Operations (UNIT) Work Package work programme<sup>1</sup> was organised in two parts:

- ❑ **GRP1**, which deals with straightforward unit operations using regular fluids in a sequential modular, steady-state simulator. It makes use of the Thermodynamics and Physical Properties interfaces originally developed by Work Package 2 (THRM) of the CAPE-OPEN project.
- ❑ **GRP2**, which extends this scenario to deal with equation-oriented simulators and dynamic simulation. This makes use of the interfaces developed originally by both the THRM and the Numerical (NUMR) Work Packages.

The UNIT Work Package itself was organised around a Focus Group and a Review Team. The Focus Group generated proposals, which were then reviewed by the Review Team and iterated until a satisfactory solution was obtained. This process first produced two internal UNIT documents: a Conceptual Requirements document and a UML Description document. The latter was an expression of the Conceptual Requirements in Unified Modelling Language, which is the standard applied within the CAPE-OPEN project for describing and analysing interface specifications. These documents were used to derive the “Unit Operations: Interim Interface Specification” document, which was the first UNIT deliverable and the basis for the UNIT demonstration prototype.

The UNIT Special Interest Group of the CAPE-OPEN Laboratories Network (CO-LaN) has taken up the task of improving and revising the document released by the CAPE-OPEN project. The current document is a synthesis of all of the above cited documents and of the learning obtained to date from the many implementations made of these interfaces.

The main body of this document describes the interfaces originally grouped as GRP1, with the extension to GRP2 contained in an appendix, since this scenario has not been prototyped.

The document starts with a text description of the requirements identified for an open unit operation component. This is then expressed in UML and developed into a specification of the interfaces necessary for a CAPE-OPEN unit operation component to plug into a compliant flowsheet simulator. The implementation specifications are provided in both COM and CORBA IDL in type libraries available separately from CO-LaN website. The final part of the document deals with the extension of these interfaces to handle steady-state and dynamic equation-oriented simulation.

## 2. Requirements

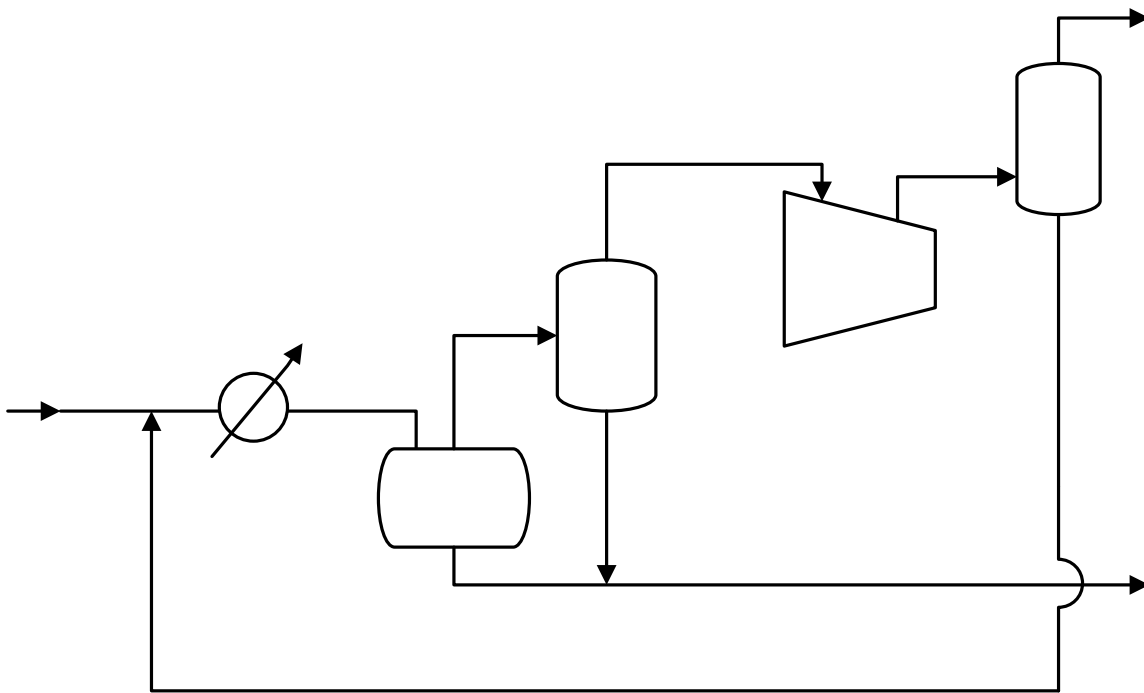
This chapter introduces the requirements. It contains a textual description followed by use cases, various diagrams and scenarios.

### 2.1 User requirements for an Open Flowsheet Unit Component

This section describes the user requirements in plain English with figures.

#### 2.1.1 Setting the scene

Flowsheet simulators are designed to calculate the behaviour of processes. They take a description of the flowsheet topology and process requirements and assemble a model of the flowsheet from a library of unit operations contained in the simulator. For example, the flowsheet in figure 2-1 represents a single stage from an oil and gas separation system:

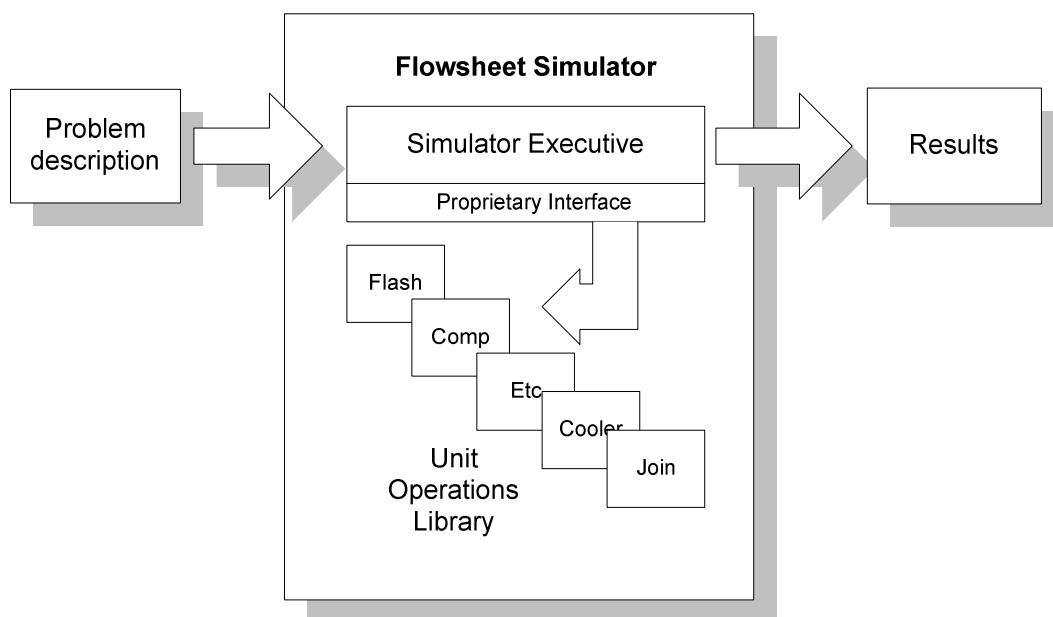


**Figure 2-1 Single stage from an oil and gas separation system**

This would be simulated in a typical commercial simulator from a description of the topology, probably entered through a Graphical User Interface (GUI) and looking much like the above picture, plus a description of the process requirements. It is a simple flowsheet that would use the flash, compressor, cooler and junction unit operations, as shown in figure 2-2.

Although the simulator mimics the plant's behaviour, it is organised differently. For example, in the plant the unit operations are connected together directly, e.g. the cooler connects directly to the separator. In the simulator, the unit operations are not connected to each other, but only to the executive. Also, in this plant there are three distinct flash separators, whereas in the simulator there is only one flash algorithm, which is reused by the simulator executive as required by the topology.

This is a straightforward flowsheet that could be adequately simulated by most simulators. However, if, for instance, the separations were to be done with a membrane unit, it might be necessary to use an external representation of the membrane unit to capture the specific performance of a proprietary membrane. Most simulators allow external unit operations to be added but, because of the proprietary nature of the interface between the unit operations library and the simulator executive and the monolithic structure of the simulator, this is a bespoke activity for each simulator. The result is a non-standard version of the simulator, which can be difficult and expensive to maintain. The CAPE-OPEN (CO) project envisages a new situation in which a unit operation (and other simulator software sub-systems, such as thermodynamic or numerical packages) can be bought off-the-shelf and plugged directly into any compliant simulator without modification,



**Figure 2-2 Flowsheet simulator schema**

compiling or linking. It will also continue to work without modification with subsequent versions of the simulator. All that is required is that the unit operation and the simulators conform to the CAPE-OPEN standards. This standards has been defined by the CO project, which was organised into Work Packages. Three of these Work Packages deal with the main sub-systems of a simulator that are likely to be exchanged: unit operations (UNIT), thermodynamics (THRM) and numerical solvers (NUMR). This document was originally produced by the UNIT Work Package and describes the requirements of the part of the CAPE-OPEN standards dealing with unit operations.

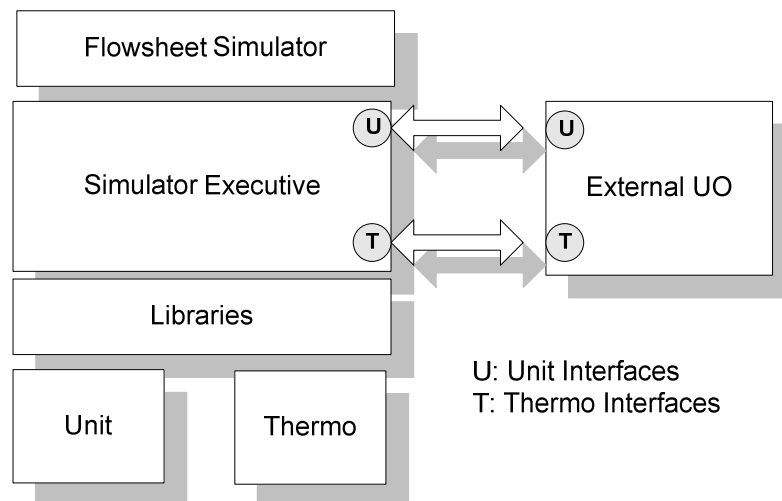
### 2.1.2 Architecture

The architecture of the CAPE-OPEN standards is based on an object-orientated technology, which allows software systems to be constructed from binary software components. These components are able to talk to each other via defined interfaces. The software components can come from different vendors and may reside on the same machine or be on different machines across a network.

The CAPE-OPEN standards are a means of connecting an external software component, which models, for example, a unit operation (UO), to any compliant simulator. The interface can be thought of as a “socket” and “plug”, which exchanges information between the two parts. The simulator and the UO do not have to know anything about the internal coding and standards used by the other. The job of the interface is to translate requests for information or action, by either party, into something the other understands.

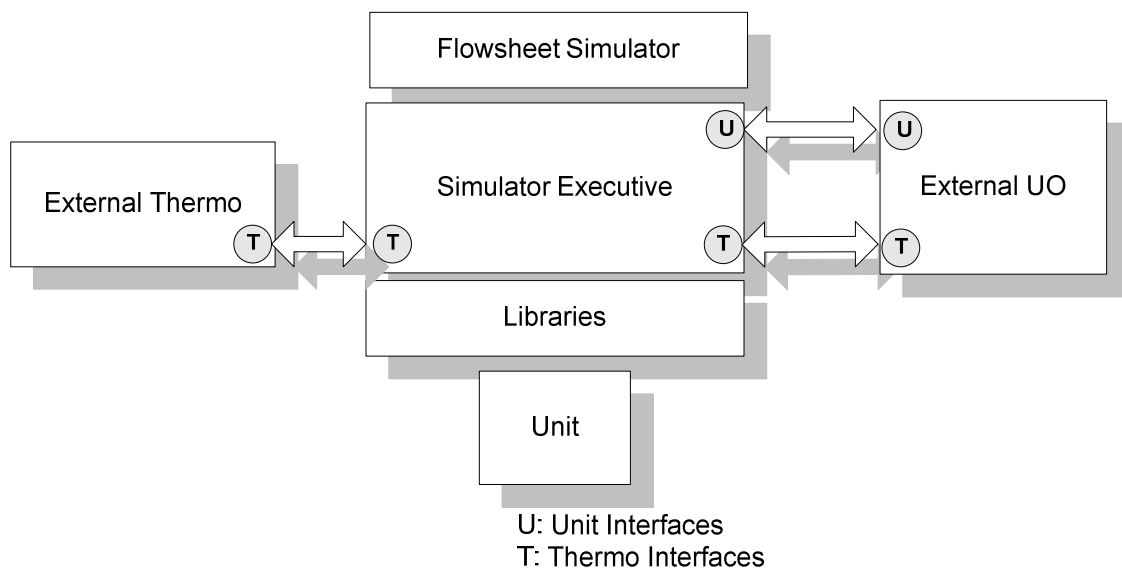
For example, figures 2-3 and 2-4 show some of the ways in which external facilities can communicate with a host simulator through the interfaces created by the CAPE-OPEN Work Packages. Please note that these are

purely conceptual representations. The separate connections shown represent the areas of responsibility of the different CO Work Packages, rather than any physical segregation in the final interface.



**Figure 2-3 Areas of responsibility of the different Work Packages (approach 1)**

Figure 2-3 shows an external UO using the flowsheet simulator’s thermodynamic facilities. The UO is solving its own equations and so is not shown sending numerical information to the simulator. This is likely to be the default method of operation in sequential modular simulators. An alternative scenario is as follows:



**Figure 2-4 Areas of responsibility of the different work packages (approach 2)**

In Figure 2-3, the thermodynamic calculations are provided by the simulator. The external UO uses the thermodynamic calculation services as provided by the simulator. In Figure 2-4, the thermodynamic calculation services are provided by an external library (e.g. a CAPE-OPEN Physical Property Package). The simulator is responsible for communication with the external thermodynamic library. The external UO still uses the thermodynamic services that are provided by the simulator, as in Figure 2-3. The external UO is not aware that the simulator uses an external thermodynamic library.

### 2.1.3 CO Objects Present in a Compliant Simulator

In software terms, this architecture requires the following types of objects in a CO system:

- ❑ **Unit:** this represents the CO unit operation and provides methods for initialisation, calculation and reporting. A CO compliant simulator uses these methods to operate the plug-in unit operation.
- ❑ **Simulator Executive:** this provides services that a unit operation is likely to need from the simulator, such as stream information and the Simulation Context.
- ❑ **Thermo:** this provides physical property calculation services and defines the Material Objects connected to the Unit's material ports
- ❑ **Numerics:** this provides numerical services for Equation Oriented units (see Chapter 10)

The actual location of thermodynamic services may be in a separate plug-in, or the thermodynamic services may be provided by the Simulator Executive itself. To the Unit this is of no consequence, as it accesses the thermodynamic services via the interfaces made available by the Simulator Executive. These are a) the Material Objects connected to the material ports of the Unit and b) the ICapeMaterialTemplateSystem that may be exposed by the Simulation Context object.

A Unit does not, of course, have to use the thermodynamic services of the Simulator Executive, if it has these services built-in already, with the exception of obtaining material feed conditions from the Material Objects connected to the material inlet ports, and specifying product conditions through the Material Objects connected to the material outlet ports.

### 2.1.4 Communications between a Unit and a Simulator Executive

The communications between the Unit and the Simulator Executive take place during four conceptual phases of a simulation run, using a number of two-way conversations of the following type (Implementation may differ from simulator to simulator):

#### PHASE 1: FLOWSHEET CREATION

In this phase, the user adds a Unit to the flowsheet. This causes the Simulator Executive to initialize the Unit, and to ask the Unit how many input and output ports it can handle, to check that the usage is correct.

**Table 2-1 Flowsheet creation**

<b>Simulator Executive</b>	<b>External Unit</b>
Asks Unit how many inlet & outlet ports it has	Responds with number of inlet & outlet ports

Editing the Unit may cause this information to change. Creating the Unit may involve loading a previously stored Unit using persistence interfaces.

#### PHASE 2: FLOWSHEET AND UNIT CONFIGURATION

The flowsheet has been created and the configuration of the Unit can be specified. The specification of the Unit may involve invoking its Graphical User Interface (GUI), and/or setting its public input parameters. Some Units may depend on invocation of their Graphical User Interface to be able to function. Such Units will not function in scenarios where configuration is done only via Public Unit Parameters (PUPs). Configuration via PUPs only is a scenario that may occur in situations where the Flowsheet Builder is not an actual human being, for example in automated testing environments.

In case the Graphical User Interface is invoked, the Simulation Environment should expose thermodynamic functionality (as far as the Property System is configured at the point of editing the unit) via Material Objects

connected to the Unit’s materials Ports (if any). Some Units may depend on this functionality to function properly.

**Table 2-2 Flowsheet and Unit configuration**

<b>Simulator Executive</b>	<b>External Unit</b>
Asks Unit to display GUI	Displays GUI in which the user can enter configuration details
Modifies values of input parameters	Receives new input parameter data and adjusts configuration accordingly

Once configuration has ended, the Simulation Executive should re-obtain the available ports and parameters from the Unit, as they may have changed during Unit configuration.

PHASE 3: FLOWSHEET SOLUTION AND UNIT CALCULATION

In this phase, the Simulator Executive evaluates each Unit in turn. In the case of CAPE-OPEN Units, it invokes the “Calculate” method of the Unit. The Unit in turn invokes methods of the Simulator Executive, Thermo objects, etc... as required, during the course of its calculation.

**Table 2-3 Flowsheet Solution and Unit Calculation**

<b>Simulator Executive</b>	<b>External Unit</b>
Invokes calculation	Requests Unit feed information from inlet ports
Sends Unit feed information	Begins calculation
Waits for completion, sending data and providing services as requested	Requests data and services as required during the course of the calculation.
	On completion, sets outlet port values

PHASE 4: FLOWSHEET AND UNIT REPORTING

In this phase, the flowsheet has converged. The Simulator Executive asks the Unit to provide its results, either as a specific report, or to pass them through the interface for inclusion in a general simulator report. Units may provide their own ASCII text file report which can be output to the user or included in an overall flowsheet report. A Simulator Executive may choose to ignore this facility and instead may fetch results from the Unit and format them itself.

**Table 2-4 Flowsheet and Unit Reporting**

<b>Simulator Executive</b>	<b>External Unit</b>
Asks Unit for values of output parameters	Returns values of output parameters
Asks Unit to output a selected report	Outputs report

**2.1.5 Notes on Unit persistence and configuration**

It is highly recommended for Units to implement persistence according to the CAPE-OPEN Persistence Common Interface specification. This allows the Simulator Executive to store and reload Units in a binary format that is specified by the Unit, while keeping the data files or storage with the flowsheet application data.

This method of storage is less suitable for text based storage, as the Persistence Common Interface specification describes binary file storage.

A less effective way of storage could involve storing all values of the input Public Unit Parameters, and restoring these values when re-creating the Unit. Although this method of storage is more suitable to text-based storage, as it is easier to convert parameter values to human readable text, it is not guaranteed that all data, required by the Unit to operate properly, is captured by just the Public Unit Parameter values.

Finally, some Units may not implement Public Unit Parameters or persistence; these Units do not require data storage.

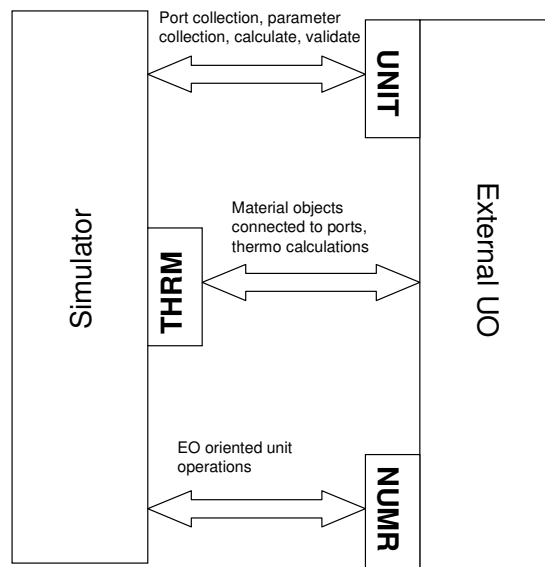
Two mechanisms are available for Unit configuration:

- The Simulator Executive can modify values of the input Public Unit Parameters; this method allows for seamless integration of the user interface with the Simulator Executive’s native user interface;
- The Simulator Executive can invoke ICapeUtilities::Edit on the Unit; this will allow for the Unit to show its own user interface. Although this method may invoke a Unit based GUI that may not fit in with the Simulator Executive’s native GUI in terms of style, design and look-and-feel, it allows a more flexible configuration from a Unit point of view; a Unit based GUI may allow access to the Unit’s configuration details that are not necessarily represented by parameters alone. The Simulator Executive should ideally support both methods of Unit configuration: via the Public Unit Parameters as well as via the private GUI of the Unit.

### 2.1.6 Conceptual Examples

These examples show the types of information passing through the interfaces, the way Public Unit Parameters are set and the transfer of stream information between a Simulator Executive and an external Unit.

#### INFORMATION TRAFFIC THROUGH THE INTERFACES



**Figure 2-5 Example of Interface Operation**

Notes on Figure 2-5, which shows the types of information passing through the interfaces:

- **Material Variables** - These are essential for the calculation of heat and mass balances, e.g. stream temperatures, etc. They have mandatory names across all CO interfaces and are

included in the CO standards. These variables are exchanged via Material Objects connected to material ports.

- ❑ **Energy Variables** - These are essential for the calculation of heat balances, e.g. work, etc. They have mandatory names across all CO interfaces and are included in the CO standards. These variables are exchanged via Energy Objects that are connected to energy Ports.
- ❑ **Information Variables** - These represent data exchanged between two Units or a Unit and the Simulator Executive. The names of these variables are not defined by the CO standards. These variables are exchanged via Information Objects that are connected to information Ports.
- ❑ **Public Unit Parameters (PUPs)** - These are used for control/optimization (SM simulators), equation solving (EO simulators) and custom reporting. They are internal variables of an external Unit and are named and made accessible to CO interfaces by the Unit provider. Their names are not part of the CO standards. PUPs are identified as input if they can be modified by the Simulator Executive or user, or as output if they are calculated by the Unit and therefore should not be changed by an optimizer for instance. The Unit should raise an error condition, if the Simulator Executive attempts to change an output Public Unit Parameter.
- ❑ **Specific Unit data** - These will be handled through Input/Output routines supplied with the Unit, making use of the Persistence Common Interfaces. Global data, such as feed streams, will be handled through the Input/Output routines of the Simulator Executive.
- ❑ **Unit reports** - Textual Unit Reports can be exchanged with the Simulator Executive through the Unit's reporting interface. Custom reports created by the Simulator Executive can include information from the external Unit via the PUPs defined by the Unit provider.

The above can be summarised as follows:

#### ❑ **In a Sequential Modular Simulator (SM)**

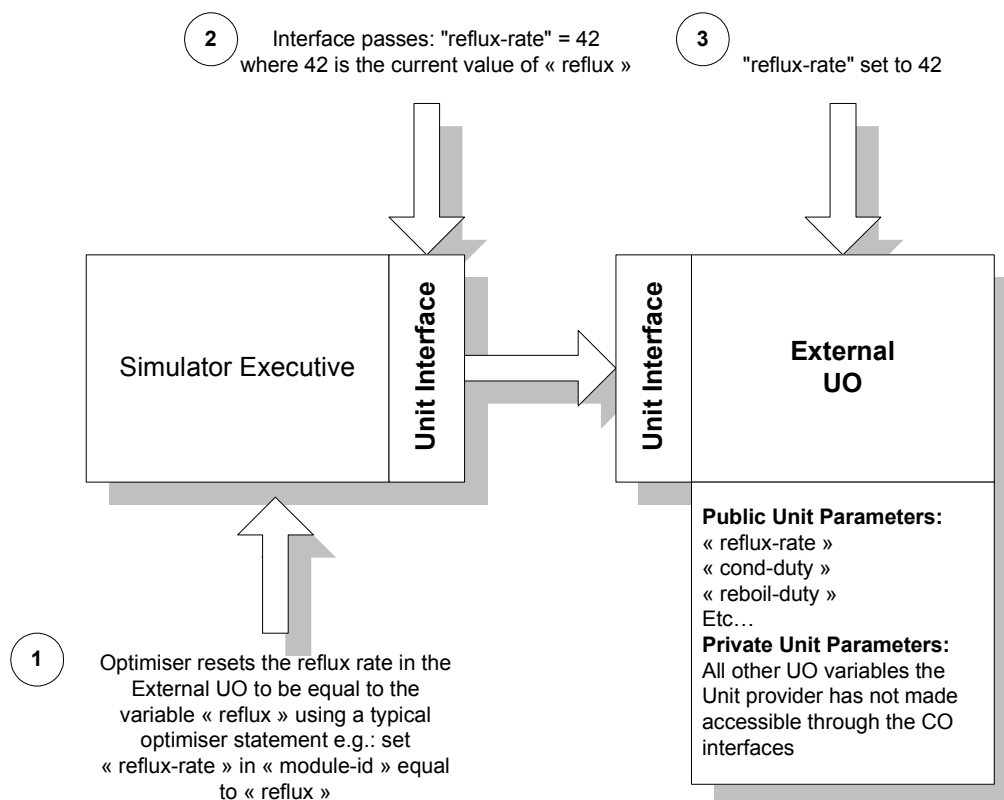
- The input Public Unit Parameter values are specified by the user and supplied to the Unit via Utilities, Collection and Parameter interfaces.
- The latest values of input Material variables are requested by and supplied to the Unit via Material Object interfaces.
- The latest value of input Energy variables are requested by and supplied to the Unit via Energy Objects; e.g. Collection and Parameter interfaces.
- The latest value of input Information variables are requested by and supplied to the Unit via Information Objects; e.g. Collection and Parameter interfaces.
- The Unit calculates new values of the output Material-, Information- and Energy variables, and passes them to the Simulator Executive via the Material Object, Collection and Parameters interfaces.
- The Unit calculates new values of the output PUPs; the Simulator Executive can request those values via the Utilities, Collection and Parameter interfaces.
- During the solution of the Unit, repeated thermophysical property requests and responses will occur via Thermodynamics and Physical Properties Interfaces

□ **In an Equation Oriented Simulator (EO)**

See chapter 9 for more information about EO units and current status.

- The input Public Unit Parameter values are specified by the user and supplied to the Unit via Utilities, Collection and Parameter interfaces.
- The latest values of input Material variables are requested by and supplied to the Unit via Material Object interfaces.
- The latest value of input Energy variables are requested by and supplied to the Unit via Energy Objects; e.g. Collection and Parameter interfaces.
- The latest value of input Information variables are requested by and supplied to the Unit via Information Objects; e.g. Collection and Parameter interfaces.
- Calculated values of the Unit equation information, e.g. derivatives and residuals, are passed to the Simulator Executive via the Equation Set Object interfaces.

Consider an external Unit, which we have assumed to be a distillation column, being manipulated by an optimizer in the Simulator Executive. The optimizer has calculated a new value of 42 for the reflux rate of the column, which it can change, as the Unit provider has included a Public Unit Parameter called “reflux\_rate” in the list of Public Unit Parameters for this particular Unit.



**Figure 2-6 Resetting Public Unit Parameters**

Note that:

- The optimizer implementation will differ from simulator to simulator, but the PUP name “**reflux-rate**” will be the same on all occasions for this particular Unit. It is defined by the

Unit provider (not by CAPE-OPEN standards). The variables “module-id” and “reflux” represent typical simulator-specific ways of identifying a module and storing a value. They will not be recognised by the CO standards.

- ❑ The form in which the Interface passes the information “reflux-rate” = 42 is the only part of this transaction that is defined by CAPE-OPEN. The external Unit will not recognize the simulator variable “reflux”. It will only be passed its current value.
- ❑ The list of PUPs will be named by the Unit provider, who will choose how many of the Unit parameters to expose through the CO interfaces and whether they will be read only.

#### DATA TRANSFER BETWEEN AN SM SIMULATOR AND AN EXTERNAL UNIT

A sequential modular simulator will exchange data with a Unit via:

- **Material Objects connected to material inlet ports:** the Unit will obtain feed conditions, such as pressure, temperature, flow, composition, phase fractions and phase composition, from Material Objects connected to its inlet ports which the Unit can assume to be in thermodynamic equilibrium at the time of calculation.
- **Collections of Parameter(s) connected to energy or information inlet ports:** the Unit can obtain values from objects connected to inlet energy or information ports.
- **Input Public Unit Parameters:** during configuration of the Unit, the Simulator Executive can change values of input Public Unit Parameters, thereby passing configuration data to the Unit.
- **Intermediate Material Objects:** a Unit can duplicate a Material Object connected to one of its ports for intermediate thermodynamic property or equilibrium calculations. The Unit can subsequently set conditions on the duplicated Material Object (such as pressure, temperature, composition), perform calculations on the Material Object and retrieve calculated values.
- **Material Objects connected to outlet ports:** the Unit must have specified the conditions of each of the material outlet ports at the end of a successful calculation. It must specify overall composition and total flow (or component flows) and instruct the thermodynamic services to calculate the thermodynamic phase equilibrium on the outlets by specification of two additional variables (e.g. temperature and pressure, or pressure and enthalpy) and call of CalcEquilibrium method on the Material Object attached to each outlet stream.
- **Collections of Parameter(s) connected to energy or information outlet ports:** at calculation time, the Unit must specify all relevant data on energy and information outlet ports.
- **Output Public Unit Parameters:** the Unit is responsible for providing output Public Unit Parameters with proper values.
- **Textual Reports:** at any time the Simulator Executive can ask the Unit for a list of textual reports, and the content of any of the reports in this list.

It is important to emphasize that Units are not allowed to set or calculate values on Material Objects connected to inlet ports. However, Units can use the Duplicate method of a Material Object connected to an inlet port to create a duplicate Material Object, and perform any physical and thermodynamic property calculations on this duplicated Material Object.

“Pressure”, “temperature”, “fraction”, “phaseFraction”, “flow” and “totalFlow” are available on inlet materials without calculations. Any other properties should be calculated as outlined above.

## 2.1.7 Notes on Conceptual Operation

- ❑ Streams are not part of the Unit Operations interface specification, but are part of the overall executive of a simulator. There does not need to be a standard definition of streams in CO, so that each simulation package can retain its own stream definitions and still be compliant. The representation of a ‘material stream’ as seen by a Unit however is standardized; this is a CAPE-OPEN Material Object as defined by the Thermodynamics and Physical Properties Interface Specification. For ‘regular’ stream types, “pressure”, “temperature”, “fraction”, “phaseFraction”, “flow” and “totalFlow” are available and other CAPE-OPEN defined thermodynamic and physical properties can be calculated using CAPE-OPEN Thermodynamic and Physical Properties interfaces. More specific stream definitions, such as streams describing petroleum properties, can be defined by more specific CAPE-OPEN interfaces.
- ❑ Note that the CAPE-OPEN standards do not refer to streams as such, because streams are uniquely handled in each simulator. Instead, CAPE-OPEN refers to Ports and Material Objects, which provide a CAPE-OPEN view of a stream. A Port allows a Unit to be connected to Material Objects. Material Objects pass information in a standard format, performing any necessary conversions between the formats on either side of them (i.e. Unit and Simulator Executive).
- ❑ Material Objects that connect to a material Port are expected to implement an ICapeThermoMaterialObject interface (when using version 1.0 of the Thermodynamic and Physical Properties interface specification) or an ICapeThermoMaterial interface (when using version 1.1 of the Thermodynamic and Physical Properties interface specification). A Unit can determine whether support for its thermodynamic requirements is present by querying for either of these interfaces.
- ❑ A Simulator Executive may support only a particular version of the Thermodynamic and Physical Properties interface specification. This implies that only unit operations that also support this version of the Thermodynamic and Physical Properties specification can be used. In order to determine whether a Unit supports the required version of the Thermodynamic and Physical Properties specification, the Unit should expose supported versions of the Thermodynamic and Physical Properties standard via its registration mechanism. See Chapter 7 for details.
- ❑ If Units support usage of both of these interfaces (either one or the other, but not at the same time), and the stream exposes both of the interfaces, a mechanism is desired by which the Unit can determine the most suitable interface to use. Since this can depend on the thermodynamic objects used by the Simulator Executive, the Simulator Executive is allowed to provide advice for this. If the named value (see Simulation Context COSE Interface, ICapeCOSEUtilities, NamedValueList and NamedValue) with name “DefaultThermoVersion” is exported by the Simulation Context, a value of “1.0” indicates that ICapeThermoMaterialObject is the preferred interface, whereas a value of “1.1” indicates that ICapeThermoMaterial is the preferred interface.
- ❑ Information Objects that connect to Information Ports are expected to expose an ICapeCollection interface. This ICapeCollection must expose one or more Parameters. The number and type of parameters in such an Information Object depends on the source Port and / or target Port the Information Object is connected to. In order for the Simulator Executive to determine how to arrange the content of an Information Object, an Information Port is assumed to expose an ICapeCollection interface. This ICapeCollection exposes ICapeParameterSpec interfaces that describe the type of information that is transmitted or received by the Information Port.
- ❑ It is recommended that an Information Port imports / exports exactly one piece of information. Furthermore, in the most common case it will be expected that this information

is of data type `CapeDouble` (Parameter of type `CAPE_REAL`). Adhering to these recommendations will facilitate agreeing on a data format of an Information Object between different flowsheet components (unit operations, optimizer blocks, etc.). This is merely a recommendation: applications that do not adhere to this recommendation may or may not work, depending on the environment that they are used in.

- ❑ Energy Objects that connect to Energy Ports are expected to expose an `ICapeCollection` of parameters, analogous to the Information Objects connected to an Information Ports described above. The parameter “work”, of which the value is of type `Real`, and the dimensionality corresponding to Watt (i.e. [m = 2, kg = 1, S = -3, A = 0, K = 0, mole = 0, ...] representing  $\text{kg m}^2 \text{s}^{-3}$ , see “Open Interface Specification: Parameter Common Interface”) is always part of an Energy Object. In addition to that additional parameters can be present depending on the type of energy that is being transported, as listed in table 2-5.
- ❑ Analogous to Information Ports, in order for the Simulator Executive to determine how to arrange the content of an Energy Object, an Energy Port is assumed to expose an `ICapeCollection` interface. This `ICapeCollection` exposes `ICapeParameterSpec` interfaces that describe the type of information that is transmitted or received by the Energy Port.
- ❑ If the sign of the work parameter is positive, energy is consumed by the Unit if the Energy Port is an inlet Port, and produced if the Energy Port is an outlet Port. The opposite applies if the sign of the work parameter is negative.

**Table 2-5 Energy consumption and production by a Unit**

Energy Port Direction	Work parameter	Energy is...	Unit sets Work value
CAPE_INLET	Positive	consumed by Unit	No
CAPE_INLET	Negative	produced by Unit	No
CAPE_OUTLET	Positive	produced by Unit	Yes
CAPE_OUTLET	Negative	consumed by Unit	Yes

- ❑ Units are independent of the flowsheet topology. The only flowsheet elements a unit operation interacts with are the objects connected to its ports. The flowsheet topology is handled by the Simulator Executive.
- ❑ The Unit is allowed to produce reports in its own internal format. The Unit may expose a list of available reports (a list which may be empty). The host simulator can select a report from this list and ask the Unit to produce the report. If no report is selected before the Unit is asked to produce the report, the Unit may produce its default report.

The host simulator is allowed to create its own report by using information obtained from the Unit. In addition to inlet and outlet streams, input and calculated (output) parameters associated with the Unit are also available to the host simulator, at the discretion of the Unit’s provider. These would be addressed using a list of variable names invented by the Unit provider. The method of accessing these variables would be a function of the host simulator’s scripting language and so would normally differ from simulator to simulator, but the variable names used would be the same in each case. The parameter names are strings that map onto the actual internal variables used by the Unit code.

- ❑ It is not the responsibility of the CAPE-OPEN standards to ensure overall flowsheet consistency, when an external Unit is plugged in. For example, if the Unit works internally

with a different set of thermodynamic models from the rest of the flowsheet, then the result will not necessarily give a heat balance at the flowsheet level. Obviously, this allows the Unit providers to differentiate themselves by providing Units which ensure consistency, and for the simulator vendors to differentiate themselves by ensuring that results with an external Unit are consistent. **In all cases, it is the simulator user's responsibility to ensure that the final system is fit for purpose.** It is recommended therefore that the Unit uses thermodynamic calculations supplied by the Simulator Executive where possible.

- A Unit graphical representation in the flowsheet is not part of the CO standards. The representation of a Unit in the flowsheet is the responsibility of the simulation software vendor. If the simulator's graphical representation of a Unit includes connection points for feed and product ports, the simulation vendor must ensure that sufficient connection points are available to connect the feeds and products that are exposed by the Unit at run-time. The Unit vendor may supply icons for its Unit for each individual simulator that supports such icons, in the format required by the simulator.

**Table 2-6 Parameters in Energy Objects**

Type of energy being transported	Additional parameter(s)	Unit of measure for additional parameter(s)
General work, e.g. electrical power or unspecified	<none>	
Heat duty in a temperature range, with $T_{low} \leq T_{high}$ . For isothermal heat duty, use $T_{low} = T_{high}$	“temperatureLow”, “temperatureHigh”	K
Work performed by a rotating axis	“axisFrequency”	s <sup>-1</sup>

### 2.1.8 Desirable Characteristics for the CO Interface

In the designs that follow, we have tried to meet the following criteria:

- Minimal performance degradation compared to native simulator facilities.
- Minimal impact on the rest of the simulator: other native facilities do not need any change
- Extendable without reworking existing facilities.
- No limitations on the data that can be transferred.
- Consistent design for all the interfaces generated across the project.
- Consistent approach to units of measure conversion across all work packages and with the host simulator.

### 2.1.9 Thermodynamic Properties

Thermodynamic properties are provided, as far as a Unit is concerned, by calling methods of a Thermo object. These methods provide for setting up a composition and state and calculating various physical properties and derivatives. A Unit does not, of course, have to use the Thermo object supplied through the Simulator Executive if it has its own thermodynamics system built-in (except for obtaining material inlet

conditions and specifying material outlet conditions). The services that the Thermo object provides may actually come from routines in the simulator itself, or from a separate plug-in software component. The Unit does not need to know where they come from, just that the Thermo object can provide them.

There are a number of ways in which a Unit can obtain a Thermo object:

- ❑ The Unit uses (a duplicate of) the Material Object connected to one of its ports. In doing so, it obtains a Material Object that the Simulator Executive or flowsheet user found most suitable for the Unit. Often all Ports are connected to Material Objects of the same type, but this is not necessarily the case and it is the responsibility of the Unit to check consistency (e.g. whether the compounds defined for inlet Material Objects match those that are defined for outlet Material Objects). It is conceivable that a Unit can deal with multiple Material Object types (e.g. hot and cold stream of a heat exchanger).

The Unit should not change the content of any Material Objects connected to inlet Ports, nor should it have any side effects on Material Objects connected to the outlet ports (other than actual unit operation outputs). Therefore, a Unit can obtain a duplicate of such a Material Object before using it to perform physical and thermodynamic property calculations or equilibrium calculations. More-over, because Material Objects may change configuration between different parts of the life-cycle of a Unit (i.e. between initialization and calculation, or between one calculation and the next), it is advised not to cache copies of Material Objects between function calls. For example, the user may have added a compound to the simulation, which is present at the inlet of the next calculation, but would not be present in a Material Object that was copied before this change.

- ❑ The user chooses from a list of Material Object types already available and known by the Simulator Executive. The Unit can use the ICapeMaterialTemplateSystem interface if exposed by the Simulation Context to create Material Objects.
- ❑ The user chooses from a selection of independent CO compliant Property Package objects, if these are available in the operating system registry. For the Unit to use an external Property Package, it has to implement its own Material Object.
- ❑ The Unit ignores all Thermo objects and performs its own built-in calculations.

The method of access of Thermo objects and the description of properties in the CO interface have been described in the Thermodynamic and Physical Properties Interface Specification. Any derivatives associated with the properties are also available from the “Thermodynamic and Physical Properties” interfaces.

Material Objects that connect to a material Port are expected to implement an ICapeThermoMaterialObject interface (when using version 1.0 of the Thermodynamic and Physical Properties Interface Specification) or an ICapeThermoMaterial interface (when using version 1.1 of the Thermodynamic and Physical Properties Interface Specification). A Unit can determine whether support for its thermodynamic requirements is present by querying for either of these interfaces.

### **2.1.10 Simultaneous Modular/Optimization**

In case of optimisation, or in case a sequential unit operation is used in the context of an equation oriented simulator, we require the provision of derivatives of outputs with respect inputs. There are basically two ways to do this: chain rule or flowsheet perturbation.

- ❑ Chain Rule

In this approach, derivatives are calculated across a flowsheet loop by calculating the derivatives of all outputs with respect to each input for each module in the loop. These are then chained together to obtain the overall derivatives required. Currently, sequential modular Units do not supply a derivative through CAPE-

OPEN interfaces. The derivative for a Unit can therefore only be obtained by perturbation with respect to each of the Unit's input variables.

- Flowsheet Perturbation

In this approach, numerical differentiation is used to calculate the overall derivatives for a flowsheet loop by perturbing the input variables individually and evaluating the flowsheet at each perturbation to obtain a numerical approximation of the derivatives required. This is done by the Simulator Executive or the optimisation system. It imposes no special requirements on the Unit and, hence, has no special implications for the CO Interface.

### 2.1.11 Granularity

In this interface design, we have restricted granularity to the unit operation level. This means we have considered plugging a complete Unit into a simulator, but not component parts of the Unit. For example, we would expect to be able to use the current CO standards to plug a complete distillation column Unit into a compliant simulator, but not necessarily to plug a new tray hydraulics method into an existing Unit. This may or may not be considered in subsequent phases of the CO standards design. In principle, a tray of a distillation column can be described as a Unit.

## 2.2 Use Cases

The next sections formalise the description of the user requirements for unit operations interfacing in the CAPE-OPEN standards, described in the previous section. They provide a Unified Modelling Language (UML)<sup>3</sup> description of the interfaces, which is the basis for the software design described in later sections.

The first step in the UML process is to express the user requirements in the form of a Use Case Model, which is described in this section. It identifies the “users” of the system, called Actors, and describes, in the form of Use Cases, what they wish the system to do. It also identifies the boundaries of the system, which is a Unit Operation Model of a steady-state, sequential modular process simulator. Other types of simulator units, such as dynamic or equation oriented, were deferred.

The process of identifying and describing the Actors and creating the Use Cases required several iterations to ensure that the user requirements were fully and consistently represented. It also identified the need for consistent names and definitions of the entities involved in flowsheet simulation. From this, a CAPE-OPEN Glossary took shape and was spun off as a separate and extremely useful exercise.

After this, the analysis phase moved on to consider the dynamic nature of the Use Cases with Sequence Diagrams and the static relationships between objects that could be identified from the Use Cases.

The Sequence Diagram is a graphical representation of the Use Case description, showing how the sequence of actions, messages and responses occur over time. For simple Use Cases, a Sequence Diagram is not necessary, but if a Use Case is very complex, with many branches and conditions, or loops, then more than one diagram may be needed. By taking this graphical view of the dynamics of the Use Case several things may be identified, such as over-complexity of the Use Case, over-complex responsibilities of the objects identified in the diagram, or even simply not enough detail in the Use Case.

The first step in identifying the static relationships in the Use Cases was to list all the objects (nouns) contained in the Use Case. Then a screening was applied to remove similar words and replace them by the agreed term from the Glossary. Then a further screening was undertaken whereby those objects identified which were outside our system boundary were ignored. Then the remaining list was examined by playing back the Use Cases to identify the actions (verbs). Only those behaviours identified as needed by the external clients of the system became part of the final interface.

The rest of this section lists and describes the Actors and Use Cases and shows their relationships in Use Case Maps. Sequence Diagrams are also shown.

### 2.2.1 Use Case Categories

- ❑ **Unit Use Case.** Applies to all the Use Cases listed in this document. They are applicable to steady-state Flowsheet Units suitable to be exchanged among sequential-modular and non-sequential-modular simulators of processes.
- ❑ **General Purpose Use Case.** Use Case that expresses a software requirement to handle CAPE-OPEN Flowsheet Units. This Use Case does not have a direct impact on the CAPE-OPEN interfaces, and therefore the requirement does not need to be met by the CAPE-OPEN interfaces.
- ❑ **Simulation Context Use Case.** These are Use Cases that list a sequence of actions, expressed as requirements, so that a CAPE-OPEN Flowsheet Unit is guaranteed to be correctly handled in the different Simulator Executives. Many times these Use Cases do not have a direct impact on the CAPE-OPEN interfaces, but they represent behavioural requirements on the simulator side. Many times they also use or extend other more specific Use Cases that do have a direct impact on one or more UNIT Interfaces.
- ❑ **Specific Unit Operation Use Case.** These are Use Cases that represent behaviours of CAPE-OPEN Flowsheet Units that do have a direct impact on one or several interfaces.
- ❑ **Boundary Use Case.** These are Use Cases in which a Flowsheet Unit is an actor in other CAPE-OPEN Use Cases different from those corresponding to Flowsheet Units (e.g. THRM Use Cases)

### 2.2.2 Use Case Priorities

- ❑ **High.** Essential functionality for a Flowsheet Unit. Functionality without which the operation usability or performance of a Flowsheet Unit might be seriously compromised
- ❑ **Medium.** Very desirable functionality that will make Flowsheet Units more usable, transportable or versatile. The essence of a Flowsheet Unit is not compromised by this Use Case, although the usability and acceptance of the component can be.
- ❑ **Low.** Desirable functionality that will improve the performance of Flowsheet Units. If this Use Case is not met, usability or acceptance can decrease.

### 2.2.3 Actors

- ❑ **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.
- ❑ **Flowsheet Solver.** A sub system responsible for converging the flowsheet by changing the adjustable variables to meet specified convergence criteria. In the modular case, this will be done by iterating the adjustable variables. In the equation-oriented case, this will be done by performing Newton iterations on a sparse set of non-linear equations. The function of setting sequencing, nesting of solving sequences and relative convergence limits will be covered in

the Use Cases of the Numerical Interface Specifications. The Flowsheet Solver would at some point make use of the sequence of computation of the Flowsheet Units. For an equation oriented simulator operating on a modular Unit, perturbation of the Unit may be required to obtain its derivatives.

- ❑ **Flowsheet Unit.** A software representation of a physical unit operation, or a non-physical unit such as a controller or optimiser.
- ❑ **Unit Manager.** The part of a simulator that provides a list of available Flowsheet Units, allows the instantiation of a Flowsheet Unit and enables it to be placed in the flowsheet.
- ❑ **Simulator Executive.** The part of a simulator whose job it is to create, or load, a previously stored flowsheet, solve it and display the results.
- ❑ **Reporting sub system.** The part of the executive that reports on the outcome of the calculation of the flowsheet. It reports on the state of the streams and Flowsheet Units involved in the flowsheet. Note: Reporting can be done in different ways. Specific reporting can be done directly by the Flowsheet Unit on a request from the executive. Overall heat and mass balance reporting is done by passing values from each Flowsheet Unit to the Reporting sub system, which has a generalised report generation capability.

#### 2.2.4 List of Use Cases

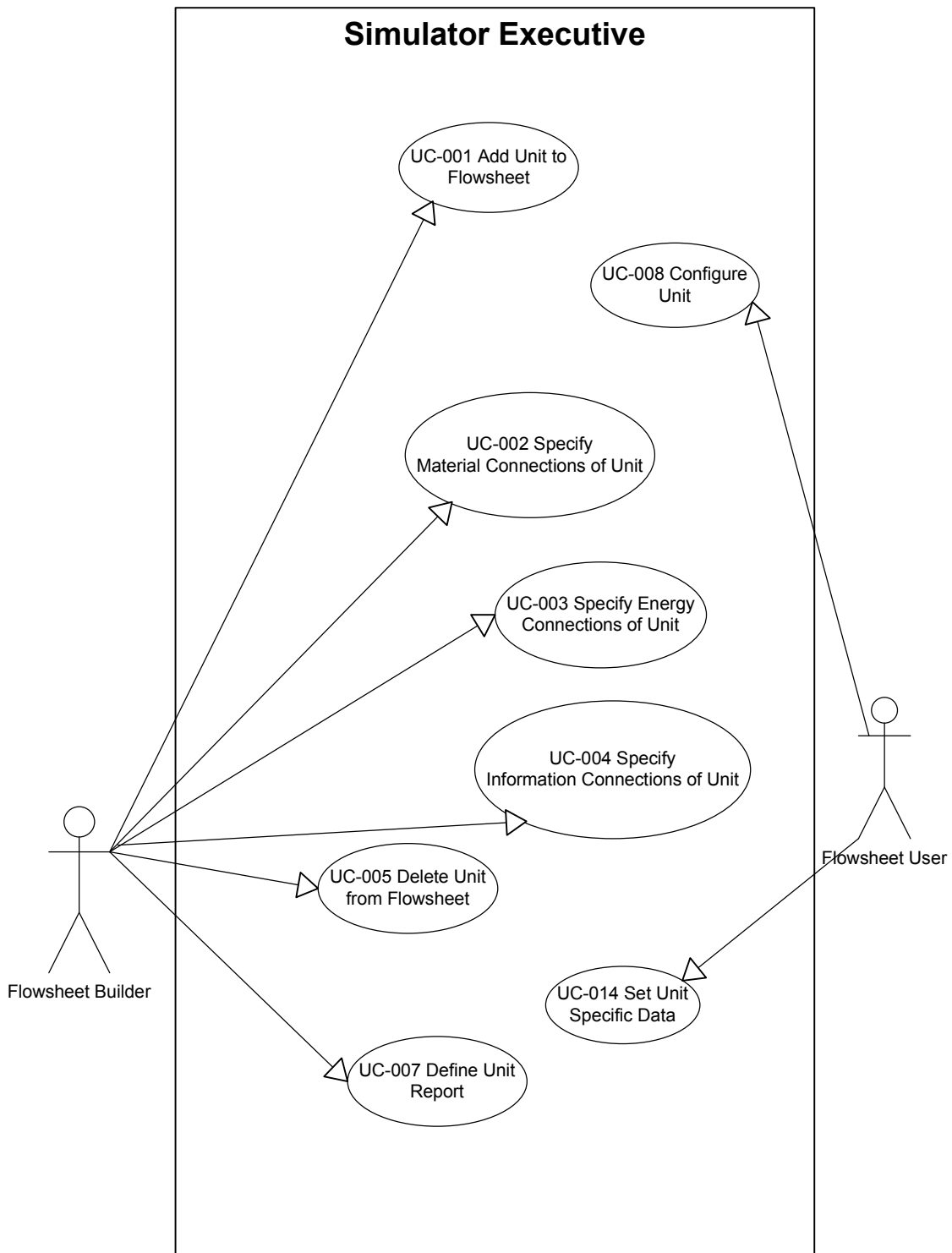
This subsection lists all the Use Cases that are relevant for the Unit Operations Interface Specification.

- ❑ UC-001: Add Unit to Flowsheet
- ❑ UC-002: Specify Material Connection of Unit
- ❑ UC-003: Specify Energy Connection of Unit
- ❑ UC-004: Specify Information Connection of Unit
- ❑ UC-005: Delete Unit from Flowsheet
- ❑ UC-006: Delete Unit
- ❑ UC-007: Define Unit Report
- ❑ UC-008: Configure Unit
- ❑ UC-009: Save Unit
- ❑ UC-010: Check Unit Specific Data
- ❑ UC-011: Check Public Unit Parameter
- ❑ UC-012: Check Physical Property Methods
- ❑ UC-013: Check Material, Energy and Information Ports
- ❑ UC-014: Set Unit Specific Data
- ❑ UC-015: Check Unit
- ❑ UC-016: Get Custom Materials

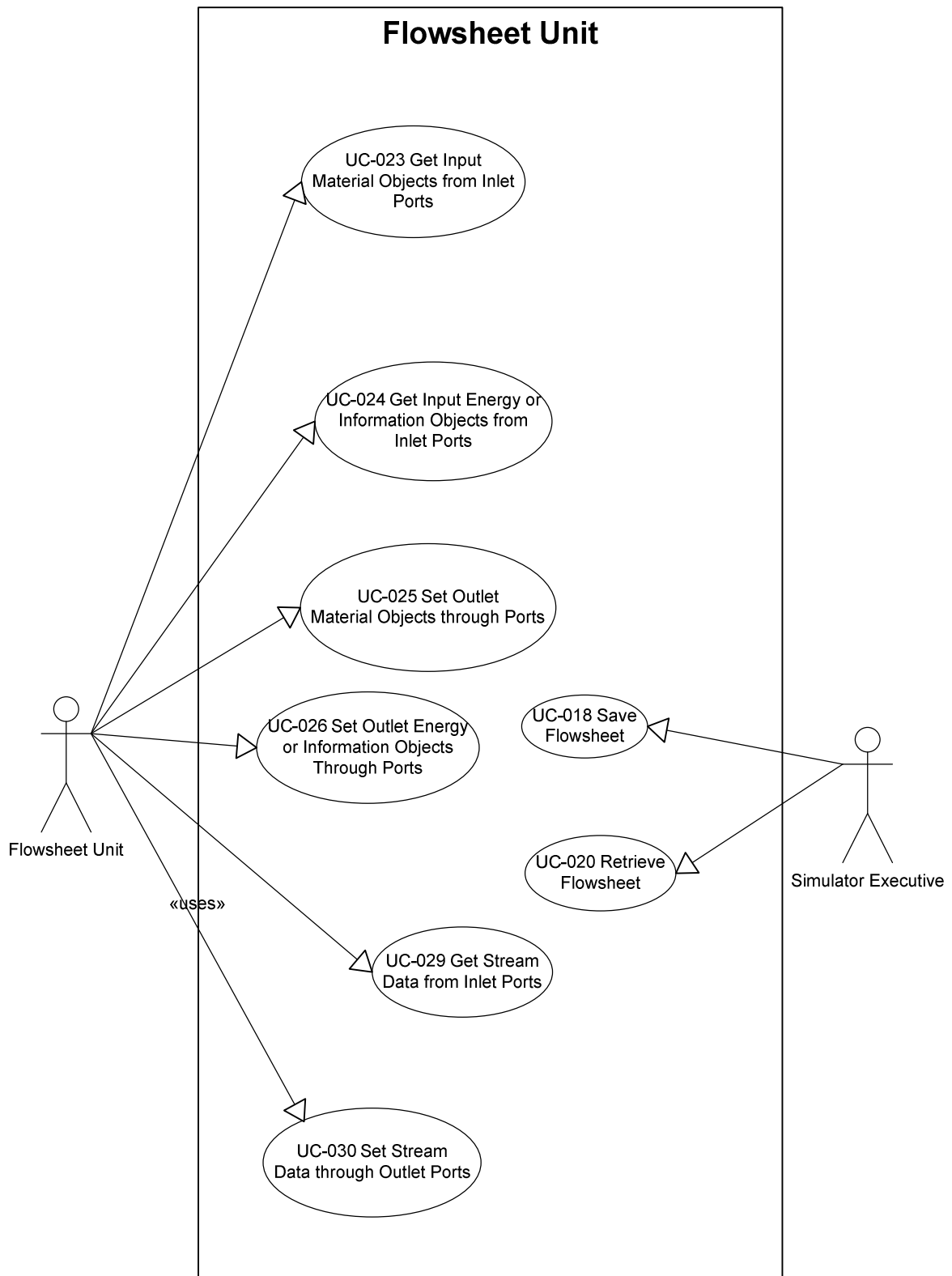
- ❑ UC-017: Get Public Unit Parameter names
- ❑ UC-018: Save Flowsheet
- ❑ UC-019: Restore Unit
- ❑ UC-020: Retrieve Flowsheet
- ❑ UC-021: Evaluate Unit
- ❑ UC-022: Perturb Unit
- ❑ UC-023: Get Input Material Objects from Inlet Ports
- ❑ UC-024: Get Input Energy or Information Objects from Inlet Ports
- ❑ UC-025: Set Outlet Material Objects through Ports
- ❑ UC-026 Set Outlet Energy or Information Objects through Ports
- ❑ UC-027 Get Value of Public Unit Parameter
- ❑ UC-028 Set Value of Public Unit Parameter
- ❑ UC-029 Get Stream Data from Inlet Ports
- ❑ UC-030 Set Stream Data through Outlet Ports
- ❑ UC-031 Produce Unit Report

### **2.2.5 Use Cases maps**

Use Cases maps have been drawn to assemble Use Cases pertaining to actions on the same systems. The systems considered are Simulator Executive, Flowsheet Unit and Thermo.



**Figure 2-7 Use Cases on Simulator Executive system with "human" actors involved**



**Figure 2-8 Use Cases on Flowsheet Unit system by "non-human" actors**

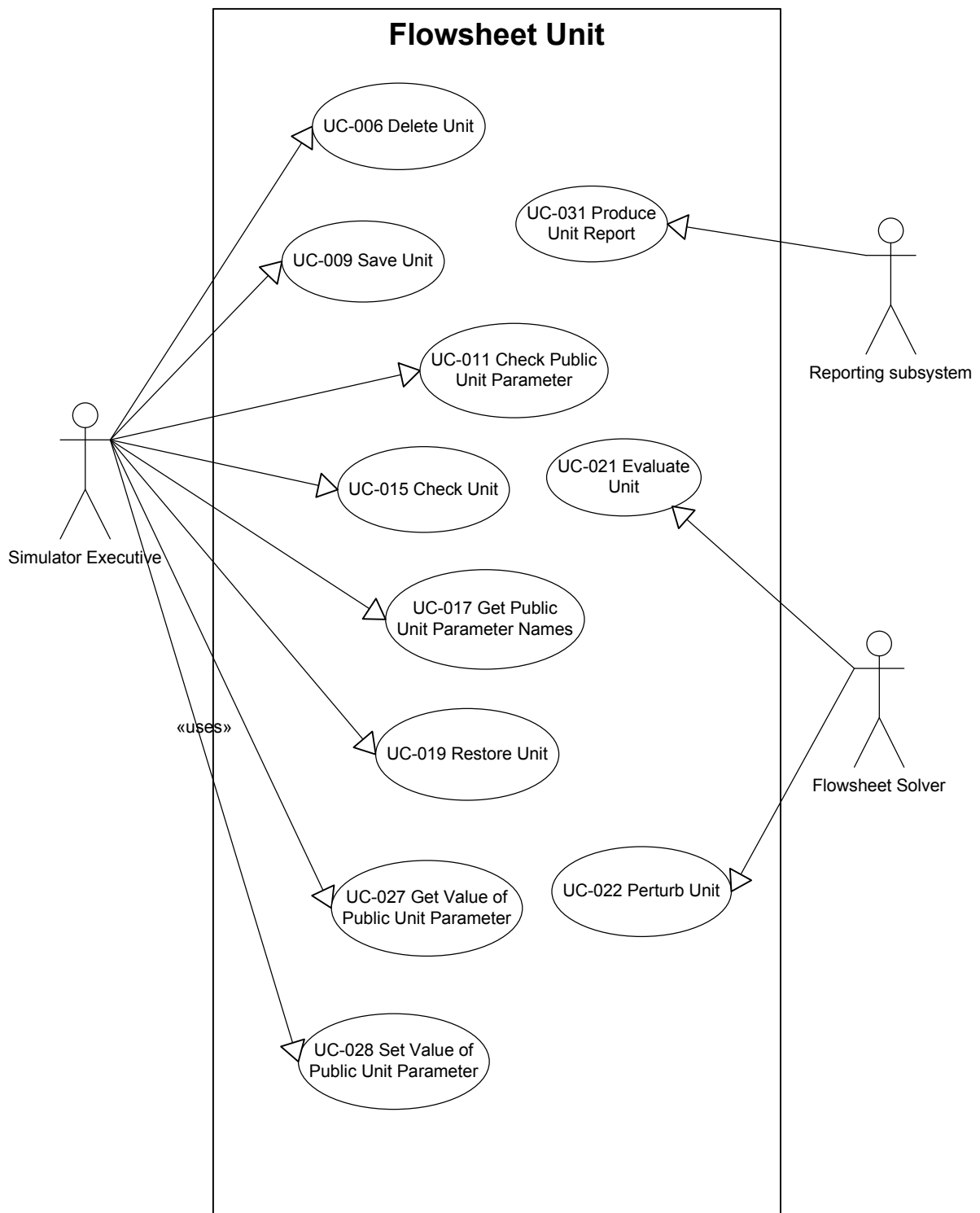
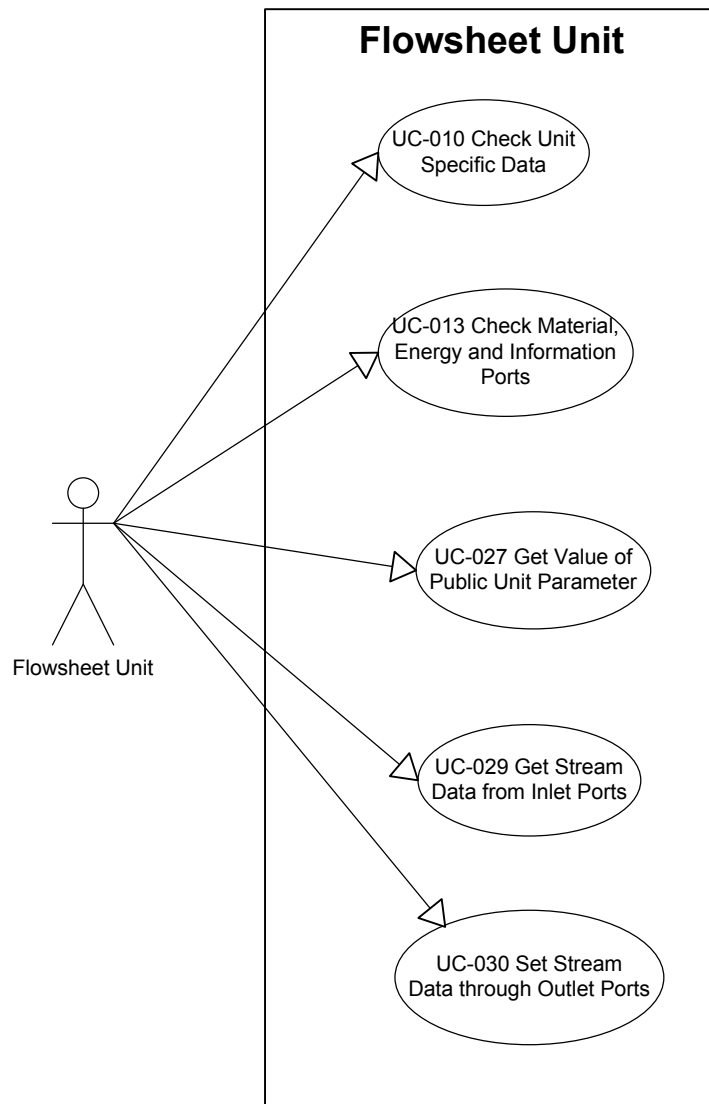
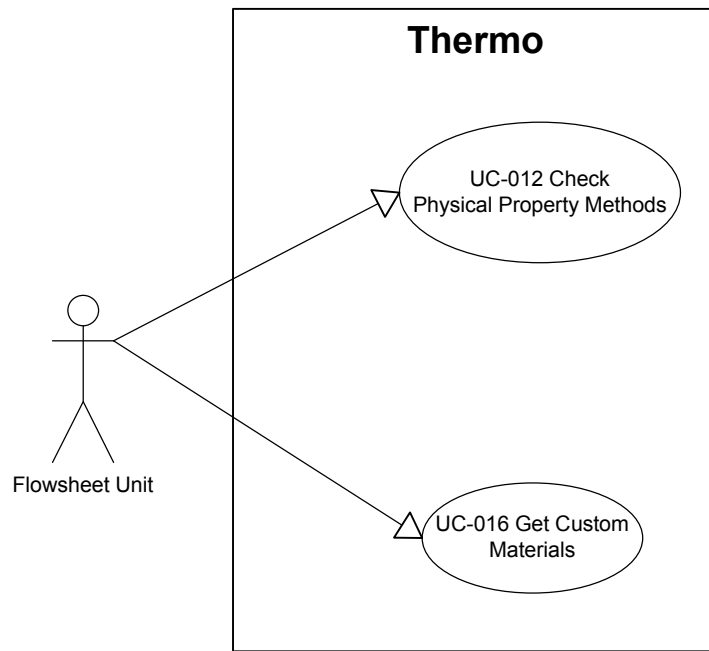


Figure 2-9 Use Cases on Flowsheet Unit system by Simulator Executive and its sub-systems



**Figure 2-10 Use Cases on Flowsheet Unit system by Flowsheet Unit**



**Figure 2-11 Use Cases on Thermo system by Flowsheet Unit**

## 2.2.6 Use Cases

This subsection describes all the Use Cases that are relevant to the Unit Operations Interface Specification. It also provides links to the methods identified in each Use Case.

### UC-001 ADD UNIT TO FLOWSHEET

Actors: <Flowsheet Builder>

Priority: <High>

Classification: <Unit Use Case>, <Simulation Context Use Case>

Context: A process model is represented by a Flowsheet made of one or several connected Flowsheet Units.

Status: <The first part of the Use Case is fulfilled by the normal operation of process simulators>

Pre-conditions: <Flowsheet must exist>

Flow of events: Selects a Flowsheet Unit type and adds an instance of it to the flowsheet.

The Flowsheet Builder gets the list of the available types of Flowsheet Units from the Unit Manager and selects one Flowsheet Unit from the list. For COM, the Unit Manager can obtain such a list from the Windows Registry by enumerating the objects that have the CAT ID for CAPE-OPEN Unit Operations listed (see Methods & Tools Integrated Guidelines documentation). The Unit Manager creates the instance of the Flowsheet Unit. If the creation is successful, the Unit Manager must initialize the Flowsheet Unit, by calling InitNew (if implemented) and Initialize, in this order. If the Simulation Context is available, it should be set after the Initialize call. If the creation fails, the Flowsheet Builder is informed of this, and the Unit Manager takes no further action.

Post-conditions: <Flowsheet Unit is successfully initialized>

Errors: <Initialize Flowsheet Unit fails>, <Failure to present the Flowsheet Unit in the Flowsheet>

Uses: none

Extends: none

## UC-002 SPECIFY MATERIAL CONNECTION OF UNIT

Actors: <Flowsheet Builder>

Priority: <High>

Classification: <Unit Use Case>, <Specific Unit Use Case>

Context: A process model represented by a flowsheet describes the flow of material through a number of Flowsheet Units. Hence there is a need to link Flowsheet Units through streams carrying material (represented by Material Objects).

Status: <This Use Case is fulfilled by the following methods: ICapeUnit::GetPorts, ICapeCollection::Count, ICapeCollection::Item, ICapeUnitPort::GetPortType, ICapeUnitPort::GetDirection, ICapeUnitPort::Connect, ICapeUnitPort::Disconnect>

Pre-conditions: <There are material streams to connect to the Flowsheet Unit, or they can be created when the connection is to be performed>, <Flowsheet can present information on material streams to the Flowsheet Unit in the appropriate form (i.e. Material Objects)>, <The Flowsheet Unit has material ports to be connected (i.e. input or output)>

Flow of events: Flowsheet Unit connections are specified by connecting or disconnecting input or output material Ports. The connection and disconnection of inlet and outlet material Ports may be performed in any order.

### *Connecting a Material Port*

The Flowsheet Builder asks the Simulator Executive to get the list of unconnected inlet or outlet material Ports on the Flowsheet Unit. If not done before, the Simulator Executive then asks the Flowsheet Unit for its Port Collection. The Simulator Executive iterates over the Port Collection items to obtain a list of unconnected inlet or outlet material Ports, and passes the list to the Flowsheet Builder. The Flowsheet Builder selects a material stream to connect.

The Flowsheet Builder selects an inlet or outlet Port from the list obtained and asks the Simulator Executive to make a connection between the selected stream and the selected Port. The Simulator Executive asks the Flowsheet Unit to make the requested connection between the selected stream and Port.

### *Disconnecting a Material Port*

The Flowsheet Builder asks the Simulator Executive to get the list of connected inlet and outlet material Ports on the Flowsheet Unit. If not done before, the Simulator Executive then asks the Flowsheet Unit for its Port Collection. The Simulator Executive iterates over the Port Collection items to obtain a list of connected inlet and outlet material Ports, and passes the list to the Flowsheet Builder. The Flowsheet Builder selects the Port to disconnect and asks the Simulator Executive to disconnect the Port. The Simulator Executive asks the Flowsheet Unit to disconnect the Port.

Post-conditions: <Port has been successfully connected or disconnected>

Errors: <Connection can not be made>

Uses: none

Extends: none

## UC-003 SPECIFY ENERGY CONNECTION OF UNIT

Actors: <Flowsheet Builder>

Priority: <Medium>

Classification: <Unit Use Case>, <Specific Unit Use Case>

Context: A process model describes material and energy transfer between a number of Flowsheet Units. Energy exchange in a process model is described by energy objects.

Status: <This Use Case is fulfilled by the following methods: ICapeUnit::GetPorts, ICapeCollection::Count, ICapeCollection::Item, ICapeUnitPort::GetPortType, ICapeUnitPort::GetDirection, ICapeUnitPort::Connect, ICapeUnitPort::Disconnect>

Pre-conditions: <There are energy streams to connect to the Flowsheet Unit, or they can be created when the connection is to be performed>, <Flowsheet can present content of energy streams to the Flowsheet Unit in the appropriate form, i.e. via an Energy Object, see 2.1.7 Notes on Conceptual Operation>, <The Flowsheet Unit has energy Ports to be connected (i.e. inlet or outlet)>

Flow of events: Flowsheet Unit connections are specified by connecting or disconnecting input or output energy Ports. The connection and disconnection of inlet and outlet energy Ports may be performed in any order.

### *Connecting an Energy Port*

The Flowsheet Builder asks the Simulator Executive to get the list of unconnected input or output energy Ports on the Flowsheet Unit. If not done before, the Simulator Executive then asks the Flowsheet Unit for its Port Collection. The Simulator Executive iterates over the Port Collection items to obtain a list of unconnected input or output energy Ports, and passes the list to the Flowsheet Builder. The Flowsheet Builder selects an energy stream to connect.

The Flowsheet Builder selects an inlet or outlet Port from the list obtained and asks the Simulator Executive to make a connection between the selected stream and the selected Port. The Simulator Executive asks the Flowsheet Unit to make the requested connection between the selected stream and Port.

### *Disconnecting an Energy Port*

The Flowsheet Builder asks the Simulator Executive to get the list of connected inlet and outlet energy Ports on the Flowsheet Unit. If not done before, the Simulator Executive then asks the Flowsheet Unit for its Port Collection. The Simulator Executive iterates over the Port Collection items to obtain a list of connected inlet and outlet energy Ports, and passes the list to the Flowsheet Builder. The Flowsheet Builder selects the Port to disconnect and asks the Simulator Executive to disconnect the Port. The Simulator Executive asks the Flowsheet Unit to disconnect the Port.

Post-conditions: <Port has been successfully connected or disconnected>

Errors: <Connection can not be made>

Uses: none

Extends: none

## UC-004 SPECIFY INFORMATION CONNECTION OF UNIT

Actors: <Flowsheet Builder>

Priority: <Low> if this can be achieved by other means (e.g. using the Flowsheet Unit Parameters)

Classification: <Unit Use Case>, <Specific Unit Use Case>

Context: A process model represented as a Flowsheet may involve the representation of information between Flowsheet Units or between the Simulator Executive and the Flowsheet Unit.

Status: <This Use Case is fulfilled by the following methods: ICapeUnit::GetPorts, ICapeCollection::Count, ICapeCollection::Item, ICapeUnitPort::GetPortType, ICapeUnitPort::GetDirection, ICapeUnitPort::Connect, ICapeUnitPort::Disconnect>

Pre-conditions: <There are information streams to connect to the Flowsheet Unit, or they can be created when the connection is to be performed>, <Flowsheet can present content of information streams to the Flowsheet Unit in the appropriate form, i.e. via an Information Object, see 2.1.7 Notes on Conceptual Operation>, <The Flowsheet Unit has information Ports to be connected (i.e. inlet or outlet)>

Flow of events: Flowsheet Unit connections are specified by connecting or disconnecting inlet or outlet information Ports. The connection and disconnection of inlet and outlet information Ports may be performed in any order.

### *Connecting an Information Port*

The Flowsheet Builder asks the Simulator Executive to get the list of unconnected inlet or outlet information Ports on the Flowsheet Unit. If not done before, the Simulator Executive then asks the Flowsheet Unit for its Port Collection. The Simulator Executive iterates over the Port Collection items to obtain a list of unconnected inlet or outlet information Ports, and passes the list to the Flowsheet Builder. The Flowsheet Builder selects an inlet or outlet Port from the list obtained. The Flowsheet Builder selects an information stream to connect or asks the Simulator Executive to create an information stream and connect it to the selected Port. The Simulator Executive creates the new information stream.

If the stream is not yet connected on either side, the Simulator Executive obtains a description for the input or output information from the Port (see 2.1.7 Notes on Conceptual Operation). The Simulator Executive then arranges the content of the information stream (number, type, name and dimensionality of the Parameter Objects) to match that of the Port that it is about to connect to. The Simulator Executive asks the Flowsheet Unit to make the requested connection between the selected stream and Port.

The connected inlet or outlet Port then becomes assigned, and the information stream and the Parameter it contains then become available within the Flowsheet. It is recommended that each information Port exposes just one real Parameter.

It is the responsibility of the information Port to test whether the Parameter(s) that is (are) available on the information stream matches the demands of the Port.

### *Disconnecting an Information Port*

The Flowsheet Builder asks the Simulator Executive to get the list of connected inlet and outlet information Ports on the Flowsheet Unit. If not done before, the Simulator Executive then asks the Flowsheet Unit for its Port Collection. The Simulator Executive iterates over the Port Collection items to obtain a list of connected inlet and outlet information Ports, and passes the list to the Flowsheet Builder. The Flowsheet Builder selects the Port to disconnect and asks the Simulator Executive to disconnect the Port. The Simulator Executive asks the Flowsheet Unit to disconnect the Port.

Post-conditions: <Port has been successfully connected or disconnected>

Errors: <Connection can not be made>

Uses: none

Extends: none

## UC-005 DELETE UNIT FROM FLOWSHEET

Actors: <Flowsheet Builder>

Priority: <High>

Classification: <General Purpose Use Case>

Context: During the course of the development of a process model represented as a Flowsheet, it may be necessary to modify the topology by removing a Unit from the Flowsheet.

Status: <The Use Case is fulfilled by the normal operation of process simulators>

Pre-conditions: <[Add Unit to Flowsheet] has been used and successfully passed, or [Restore Unit] has been used and successfully passed>

Flow of events: The Flowsheet Builder selects a Flowsheet Unit and asks the Simulator Executive to remove it from the flowsheet.

The Flowsheet Builder selects a Flowsheet Unit in the flowsheet and uses the [Delete Unit] Use Case. The Simulator Executive then removes the Flowsheet Unit from the flowsheet.

Post-conditions: <The Flowsheet Unit has been removed from the flowsheet>

Errors: <The Flowsheet Unit can not be removed from the flowsheet>

Uses: [Delete Unit]

Extends: none

## UC-006 DELETE UNIT

Actors: <Simulator Executive>

Priority: <High>

Classification: <Unit Use Case>, <Specific Unit Use Case>, <Simulation Context Use Case>

Context: All Flowsheet Units that are created are eventually deleted

Status: <This Use Case is fulfilled by the following methods: ICapeUnit::GetPorts, ICapeCollection::Count, ICapeCollection::Item, ICapeUnitPort::Disconnect, ICapeUtilities::Terminate>

Pre-conditions: <[Add Unit to Flowsheet] has been used and successfully passed, or [Restore Unit] has been used and successfully passed>

Flow of events: This deletes the instance of a Flowsheet Unit.

The Flowsheet Builder requests that the Unit Manager deletes the selected Flowsheet Unit instance. The Unit Manager asks the Flowsheet Unit to remove every Port connection. The Unit Manager then Terminates the Unit by calling the ICapeUtilities::Terminate method (at which point the Flowsheet Unit releases all its references to Flowsheet objects such as the Simulation Context). The Unit Manager then deletes the Flowsheet Unit instance.

Post-conditions: <Any connection between the Flowsheet Unit and other flowsheet objects has been removed>, <Termination proceeded successfully>, <The Flowsheet Unit has been deleted>

Errors: <The Flowsheet Unit can not be disconnected>, <The Flowsheet Unit cannot be deleted>, <Termination failed>

Uses: none

Extends: none

## UC-007 DEFINE UNIT REPORT

Actors: <Flowsheet Builder>

Priority: <Low>

Classification: <Unit Use Case>, <Specific Unit Use Case>

Context: A Flowsheet Unit may provide results of its calculations not only through values of PUPs and values in Objects connected to its outlet Ports, but also by a set of reports than can be requested by the Simulator Executive for incorporation in its own reporting mechanism.

Status: <This Use Case is fulfilled by the following methods: ICapeUnitReport::GetReports, ICapeUnitReport::Get/SetSelectedReport>

Pre-conditions: <[Add Unit to Flowsheet] has been used and successfully passed, or [Restore Unit] has been used and successfully passed>, <Flowsheet Unit implements report handling>

Flow of events: The Flowsheet Builder asks the Unit Manager to get the list of available report formats from the Flowsheet Unit. If there are some available report formats the Unit Manager displays them, so that the Flowsheet Builder can select one of them. The Unit Manager asks the Flowsheet Unit to set the selected report format for output. The Flowsheet Unit sets this format as its output format.

Post-conditions: <A report format has been selected>

Errors: none

Uses: none

Extends: none

## UC-008 CONFIGURE UNIT

Actors: <Flowsheet User>

Priority: <High>

Classification: <Unit Use Case>, <Specific Unit Use Case>.

Context: When a Flowsheet Unit is created, it has no connection with other elements of the Flowsheet and has default values for its PUPs. So the Flowsheet Unit needs to be configured in order to reflect the specifics of each situation, i.e. each process model it is used in.

Status: <This Use Case is fulfilled by the methods listed in the various Use Cases in the Subordinate list below>

Pre-conditions: <[Add Unit to Flowsheet] has been used and successfully passed, or [Restore Unit] has been used and successfully passed>

Flow of events: The function of this Use Case is to group together Flowsheet Unit parameter editing Use Cases into one as a simulator is likely to have a function to edit Flowsheet Unit data. This Use Case allows any data item to be edited.

The Flowsheet User uses the [Set Unit Specific Data], [Specify Material Connection of Unit], [Specify Information Connection of Unit], [Specify Energy Connection of Unit] and [Define Unit Report] Use Cases in any order and any number of times. When the Flowsheet User completes his/her edits, the Simulator Executive may use the [Check Unit] Use Case.

Post-conditions: none

Errors: <Check Unit fails>

Uses: <Set Unit Specific Data>, <Specify Material Connection of Unit>, <Specify Information Connection of Unit>, <Specify Energy Connection of Unit>, <Define Unit Report>, <Check Unit>

Extends: none

## UC-009 SAVE UNIT

Actors: <Simulator Executive>

Priority: <Medium>

Classification: <Unit Use Case>, <Specific Unit Use Case>

Context: Considering the amount of time and resources used frequently to develop a process model represented by a Flowsheet, it is reasonable to store the model and all its elements.

Status: <This Use Case is fulfilled by the method Save of Persistence Common Interface specification>

Pre-conditions: <[Add Unit to Flowsheet] has been used and successfully passed, or [Restore Unit] has been used and successfully passed>

Flow of events: The Unit Manager will use the Save method of the Persistence Common Interfaces to store the Flowsheet Unit specific data (PUPs and any private data which may or may not be accessible through Edit). Connectivity data must be persisted by the Simulator Executive, including port names. The Simulative Executive typically also persists names (and maybe descriptions) of Flowsheet Unit. It is however advised for each Flowsheet Unit to save its name and description in order to be restored in the same state as it was saved.

Post-conditions: <Save succeeded>

Errors: <Fails to save>

Uses: none

Extends: none

## UC-010 CHECK UNIT SPECIFIC DATA

Actors: <Flowsheet Unit>

Priority: <Medium>

Classification: <Unit Use Case>, <Specific Unit Use Case>

Context: none

Status: <This Use Case is fulfilled by the following methods: ICapeUnit::Validate and ICapeUnit::GetValStatus>

Pre-conditions: none

Flow of events: This is a check on the validity of the whole Flowsheet Unit specific data. The Flowsheet Unit checks some or all of the items supplied to it as specific data to ensure that they are: within the valid range, consistent with other items of data, plus ANY other test the Flowsheet Unit wishes to perform. The Flowsheet Unit may also use default values where items are missing/inappropriate. Finally, the Flowsheet Unit returns whether it is valid or not. If not valid the Flowsheet Unit returns also a text message describing the reason.

Post-conditions: <If validation is successful GetValStatus must return CAPE\_VALID, otherwise CAPE\_INVALID>

Errors: none

Uses: none

Extends: none

## UC-011 CHECK PUBLIC UNIT PARAMETER

Actors: <Simulator Executive>

Priority: <Medium>

Classification: <Unit Use Case>

Context: A PUP may be manipulated by Actors external to the Flowsheet Unit while the Flowsheet Unit will have to make use of the value stored in a PUP in order to perform its calculation. It is then necessary to ensure that the PUP value is fit for the purpose of calculating the Flowsheet Unit.

Status: <This Use Case is fulfilled by method `getparameters` of the Utilities Common Interface and by methods of the `ICapeCollection`, `ICapeParameter`, `ICapeParameterspec`, `ICapeRealParameterSpec`, `ICapeArrayParameterSpec`, `ICapeIntegerParameter-Spec`, `ICapeOptionParameterSpec` and `ICapeBooleanParameterSpec` interfaces. >

Pre-conditions: <[Add Unit to Flowsheet] has been used and successfully passed, or [Restore Unit] has been used and successfully passed>

Flow of events: The Simulator Executive obtains the Collection of Public Unit Parameters of the Flowsheet Unit. From this Collection it obtains the intended Public Unit Parameter. The Simulator Executive can then check the mode (input or output) and data type (real, integer, option, array). Then the Simulator Executive can obtain the parameter specification. Using the parameter specification, it can check the dimensionality of the parameter. And using the type specific part (e.g. `ICapeRealParameterSpec`) of the parameter specification, it can obtain parameter constraints like minimum, maximum values and valid options or check if a particular value is valid for the parameter (using the `Validate`).

Note: this facilitates use by systems/optimisers/controllers etc., which want to use the Public Unit Parameter and have no other means of performing this check ahead of a run-time use of the variable.

Post-conditions: none

Errors: none

Uses: none

Extends: none

## UC-012 CHECK PHYSICAL PROPERTY METHODS

Actors: <Flowsheet Unit>

Priority: <Medium>

Classification: <Unit Use Case>, <Specific Unit Use Case>, <Boundary Use Case>

Context:

Status: <This Use Case is fulfilled by the calls to methods from Thermodynamic and Physical Properties interface such as GetPropList.>

Pre-conditions: <Flowsheet Unit has access to thermodynamic and physical property methods (either through Material Objects or directly accessing a thermodynamic package), <The thermodynamic and physical property methods are associated with material definitions>

Flow of events: The Flowsheet Unit checks that all required physical properties are available by calling methods of the Material Object that list supported properties.

Post-conditions: <If check is unsuccessful the state of the Flowsheet Unit CAPE\_INVALID>

Errors: <Some thermodynamic data, a Physical Property (e.g. chemicalFormula) or a thermodynamic calculation (e.g. flash) is unavailable, this will cause the state of the Flowsheet Unit to be set to invalid (i.e. ValStatus = CAPE\_INVALID)>

Uses: none

Extends: none

## UC-013 CHECK MATERIAL, ENERGY AND INFORMATION PORTS

Actors: <Flowsheet Unit>

Priority: <Medium>

Classification: <Unit Use Case>, <Specific Unit Use Case>, <Boundary Use Case>

Context: none

Status: <This Use Case is fulfilled by the GetConnectedObject method of the ICapeUnitPort interface>

Pre-conditions: <Flowsheet Unit has material, information and/or energy ports to check>

Flow of events: The Flowsheet Unit checks that all essential Ports are connected. An essential Port is a Port that must be present for the Flowsheet Unit to function e.g. a pump must have one input and one output.

Post-conditions: <If check is unsuccessful the validation state of the Flowsheet Unit CAPE\_INVALID>

Errors: <Essential Ports are not connected. This will cause the state of the Flowsheet Unit to be set to invalid (i.e. ValStatus = CAPE-INVALID)>

Uses: none

Extends: none

## UC-014 SET UNIT SPECIFIC DATA

Actors: <Flowsheet User>

Priority: <High>

Classification: <Unit Use Case>, <Specific Unit Use Case>.

Context: none

Status: <This Use Case is fulfilled by the Edit method of the ICapeUtilities interface, or by the get\_Parameters method of the ICapeUtilities interface, which returns an ICapeCollection of ICapeParameter objects to the extent Unit Specific Data may be modified by the Flowsheet User>

Pre-conditions: <Flowsheet Unit has specific data whose values need to be specified>

Flow of events: the Flowsheet User asks the Unit Manager to start editing the Flowsheet Unit. Typically the Unit Manager will attempt to invoke ICapeUnit::Edit on the Flowsheet Unit. If the Property System is configured at that time, the Simulation Executive should provide access to the functionality of the Property System via the Material Objects connected to the Ports (if any Material Port is connected). For Flowsheet Units which have not a proper user interface, the Unit Manager should allow the Flowsheet User to change the values of Public Unit Parameters.

The Simulation Executive should re-obtain the available Ports and Public parameters from the Unit, as they may have changed during Edit.

As the Port Collection may have changed, the PME should not assume that Ports that had been present before Edit and Ports that are present after Edit are the same ports; during Edit, Ports may be removed and created. Even Ports with a given name before Edit may not be the same Ports as Ports with the same name after Edit. The proper way to check equality of Ports before and after Edit is to check which object is connected to the Port using GetConnectedObject.

Post-conditions: <Unit specific data have been specified>

Errors: <Flowsheet 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: none

## UC-015 CHECK UNIT

Actors: <Simulator Executive>

Priority: <High>

Classification: <Unit Use Case>, <Specific Unit Use Case>

Context: After configuration of the Flowsheet, the Flowsheet Unit must declare itself as ready to be calculated.

Status: <This Use Case is fulfilled by the following methods: ICapeUnit::Validate and ICapeUnit::GetValStatus>

Pre-conditions: <[Add Unit to Flowsheet] has been used and successfully passed, or [Restore Unit] has been used and successfully passed>

Flow of events: The Simulator Executive may call an external CAPE-OPEN Flowsheet Unit and ask it to use the [Check Unit Specific Data], [Check Material, Information and Energy Ports], [Check Physical Property Methods] Use Cases. These checks are provided by the Flowsheet Unit, without knowledge of how or when they may be called.

During validation, the Flowsheet Unit may typically obtain information regarding the configuration (not the state) of the objects connected to the Ports. For example, it may ask for a list of compounds on each Port to see whether a) the Flowsheet Unit can deal with this list and b) the compounds on in- and outlet Ports are consistent. Therefore, a call to Validate may be relatively time consuming, and it is not recommended to call Validate before each call to Calculate, unless the configuration of the Unit has changed since the last call to Calculate.

The state of objects connected to Ports should not be validated, e.g. the temperature on Material Objects connected to Material Ports, or values of parameters on objects connected to information Ports. It will be up to the Calculate method to return an appropriate error if the model is not valid for the given inlet conditions.

The Validate method *must* be called before Calculate when any (one or more) of the following have occurred since the last calculation:

- inserting the Flowsheet Unit into the flowsheet
- restoring the Flowsheet Unit from file
- connecting a object to a Port
- disconnecting n object from a Port
- calling the Edit method
- changing a parameter on the Flowsheet Unit via a parameter interface
- changing property package configuration (i.e. changing number of compounds, phases, thermodynamic properties, ...)

Post-conditions: <If Validation is unsuccessful the state of the Flowsheet Unit CAPE\_INVALID>

Errors: <Validation failed. This will cause the state of the Flowsheet Unit to be set to invalid (i.e. ValStatus = CAPE\_INVALID)>

Uses: [Check Unit Specific Data], [Check Material, Energy and Information Ports], [Check Physical Property Methods]

Extends: none

## UC-016 GET CUSTOM MATERIALS

Actors: <Flowsheet Unit>

Priority: <High>

Classification: <Unit Use Case>, <Specific Unit Use Case>, <Boundary Use Case>, <Simulation Context Use Case>

Context: Flowsheet Unit needs access to Material Object of types that are different from those connected to feed or product Ports (If a Material of a type equal to one connected to a feed or product Port, a duplicate of a connected Material Object can simply be used).

Status: <This Use Case is fulfilled by the services provided by put\_SimulationContext if this exposes ICapeMaterialTemplateSystem>

Pre-conditions: <Simulation Context has been set>

Flow of events: If a Flowsheet Unit has for example an internal stream, and requires thermodynamic calculations on it, the Simulator Executive may provide Material Objects for this purpose. The Simulator Executive may expose these Material Objects via the Material Template System which may be implemented by the Simulation Context.

Post-conditions: <None> ([Check Unit] will assure that the Flowsheet Unit has been able to get access to the required Material Objects)

Errors: <Requested Property Package not available>

Uses: none

Extends: none

## UC-017 GET PUBLIC UNIT PARAMETER NAMES

Actors: <Simulator Executive>

Priority: <High>

Classification: <Unit Use Case>

Context: none

Status: <This Use Case is fulfilled by using the following methods: getParameters (see Utilities Common Interface specification), Count and Item (see Collection Common Interface Specification), Name (see Identification Common Interface specification)>

Pre-conditions: <Flowsheet Unit has a collection of Public Unit Parameters>

Flow of events: The Simulator Executive requests the list of Public Unit Parameter names for the Flowsheet Unit. A pointer is obtained through ICapeUtilities::getParameters method. Count and Item methods are used to iterate over each parameter. The name of each parameter is obtained using the GetComponentName method of the ICapeIdentification interface.

Post-conditions: < Public Unit Parameter names supplied>

Errors: <Failed providing names>

Uses: none

Extends: none

## UC-018 SAVE FLOWSHEET

Actors: <Simulator Executive>

Priority: <Medium>

Classification: <General Purpose Use Case>

Context: none

Status: <This Use Case is fulfilled by the normal operation of process simulators>

Pre-conditions: <[Add Unit to Flowsheet] has been used and successfully passed, or [Restore Unit] has been used and successfully passed>

Flow of events: The Simulator Executive saves its native information in its own native format (this is outside CAPE-OPEN). Flowsheet Unit uses the [Save Unit] Use Case to save its data. Note: Connectivity data must be persisted by the simulator, including Port names.

Post-conditions: <Flowsheet Unit data location has been successfully stored with the simulation file>

Errors: <Failure saving>

Uses: [Save Unit]

Extends: none

## UC-019 RESTORE UNIT

Actors: <Simulator Executive>

Priority: <Medium>

Classification: <Unit Use Case>, <Specific Unit Use Case>, <Simulation Context Use Case>

Context: none

Status: <This Use Case is fulfilled by using method Load of the Persistence Common interface>

Pre-conditions: <Flowsheet Unit has been saved before or a Flowsheet Unit of the same type has been saved before>

Flow of events: The Simulator Executive has stored sufficient information for creating an instance of the Flowsheet Unit of the proper type. For COM operation this could be the CLSID or PROGID value. The Unit Manager will create the Flowsheet Unit. The Simulator Executive will use the interfaces described in "Persistence Common Interface" to restore the Flowsheet Unit data.

After restoring the Flowsheet Unit operation data, using Load, the Simulator Executive must Initialize the Unit. If the Simulation Context is available, it should be set after the Initialize call.

Post-conditions: <Flowsheet Unit is restored in the state a Flowsheet Unit of the same type was previously saved>

Errors: <Data not found>, <Bad data>

Uses: none

Extends: none

## UC-020 RETRIEVE FLOWSHEET

Actors: <Simulator Executive>

Priority: <Medium>

Classification: <General Purpose Use Case>

Context: none

Status: <This Use Case is fulfilled by the normal operation of process simulators>

Pre-conditions: <The simulation case to be retrieved exists>

Flow of events: The Simulator Executive retrieves the native Flowsheet Units in its usual way. For each CAPE-OPEN Flowsheet Unit in the flowsheet, it recovers the Flowsheet Unit type. The Flowsheet Unit is created. The Flowsheet Unit's data is restored as in [Restore Unit].

The Simulator Executive recovers details of the Flowsheet Unit's stream connections (which are saved in the simulator's native format) and asks each CAPE-OPEN Flowsheet Unit to connect their Ports to those specific streams.

The Simulator Executive asks the Flowsheet Unit to validate the information using the [Check Unit] Use Case. If the Flowsheet Unit fails to restore, the Flowsheet Builder is notified.

Post-conditions: <Flowsheet Unit has been appropriately created and initialised>, <Flowsheet Unit Ports have been connected to the appropriate Material, Energy or Information Objects>, <Flowsheet Unit has recovered its data>, <The state of the Flowsheet Unit is Dirty = FALSE>

Errors: <Failed to retrieve the flowsheet>, <Failed to restore the Flowsheet Unit data>, <Failure to connect Flowsheet Unit>, <Corrupted Flowsheet Unit data>

Uses: [Restore Unit], [Check Unit]

Extends: none

## UC-021 EVALUATE UNIT

Actors: <Flowsheet Solver>

Priority: <High>

Classification: <Unit Use Case>, <Specific Unit Use Case>, <Boundary Use Case>

Context: none

Status: <This Use Case is fulfilled by the following method: ICapeUnit::Calculate>

Pre-conditions: <The input and output ports (material, energy or information) have been connected>, <Flowsheet Unit has the CAPE\_VALID validation status>

Flow of events: The Flowsheet Solver tells the Flowsheet Unit to Calculate. The Flowsheet Unit then requests its Ports to get its input stream data using the [Get Inlet Material Objects from Inlet Ports] and [Get Inlet Energy or Information Objects from Inlet Ports] Use Cases. The Flowsheet Unit then attempts to perform its calculations. It may also make several requests for physical property using the [Thermo: Request Physical Properties] Use Case during the evaluation. Upon a successful partial or complete evaluation, the Flowsheet Unit sends the partial or complete definition of its output streams to its output ports using the [Set Outlet Material Objects Through Ports] and [Set Outlet Energy or Information Objects Through Ports]. It also updates any changed values of Public Unit Parameters. If the Flowsheet Unit cannot calculate, it returns the appropriate error to the Flowsheet Solver.

Post-conditions: <The Flowsheet Unit finished its calculations successfully or not>

Errors: <The Flowsheet Unit does not solve successfully>

Uses: [Thermo: Request Physical Properties], [Get Inlet Material Objects from Inlet Ports], [Get Inlet Energy or Information Objects from Inlet Ports], [Set Output Material Objects Through Ports], [Set Output Energy or Information Objects Through Ports]

Extends: none

## UC-022 PERTURB UNIT

Actors: <Flowsheet Solver>

Priority: <High>

Classification: <Unit Use Case>, <Specific Unit Use Case>, <Boundary Use Case>

Context: none

Status: <This Use Case is fulfilled by the following method: ICapeUnit::Calculate>

Pre-conditions: <The state of the Flowsheet Unit is Dirty = FALSE and Valid = TRUE>

Flow of events: Simply uses [Evaluate Unit] with some perturbed input values

Post-conditions: <The Flowsheet Unit finished its calculations successfully or not>

Errors: <Evaluate fails>

Uses: [Evaluate Unit]

Extends: none

## UC-023 GET INPUT MATERIAL DATA FROM INLET PORTS

Actors: <Flowsheet Unit>

Priority: <High>

Classification: <Unit Use Case>, <Specific Unit Use Case>, <Boundary Use Case>

Context: none

Status:

Pre-conditions: <Port is connected to a valid Material Object>

Flow of events: As in [Get Stream Data from Inlet Ports] : The objects requested are Material Objects and the Ports are material Ports.

Post-conditions: <Flowsheet Unit has retrieved data successfully>

Errors: <Stream data unavailable>

Uses: none

Extends: [Get Stream Data from Inlet Ports]

UC-024 GET INPUT ENERGY OR INFORMATION DATA FROM INLET PORTS

Actors: <Flowsheet Unit>

Priority: <Medium>

Classification: <Unit Use Case>, <Specific Unit Use Case>

Context: none

Pre-conditions: <Port is connected to a valid energy or information Object>

Flow of events: As in [Get Stream Data from Inlet Ports] except: the objects requested are Energy or Information Objects and the Ports are energy or information Ports.

Post-conditions: <Flowsheet Unit has retrieved data successfully>

Errors: <Stream data unavailable>

Uses: none

Extends: [Get Stream Data from Inlet Ports]

## UC-025 SET OUTLET MATERIAL OBJECTS THROUGH PORTS

Actors: <Flowsheet Unit>

Priority: <High>

Classification: <Unit Use Case>, <Specific Unit Use Case>, <Boundary Use Case>

Context: none

Status: <This Use Case is fulfilled by the various methods in the objects connected to material Ports>

Pre-conditions: <Port is connected to a valid Material Object>

Flow of events: As in [Set Stream Data through Outlet Ports] except: the streams requested are material streams represented by Material Objects, and the Ports are material Ports. The Flowsheet Unit must specify overall composition and total flow (or component flows) and instruct the thermodynamic services to calculate the thermodynamic phase equilibrium on the outlets by specification of two additional variables (e.g. temperature and pressure, or pressure and enthalpy).

Post-conditions: <Values have been set correctly>

Errors: <Values could not be set>

Uses: none

Extends: [Set Stream Data through Outlet Ports]

## UC-026 SET OUTLET ENERGY OR INFORMATION OBJECTS THROUGH PORTS

Actors: <Flowsheet Unit>

Priority: <Medium>

Classification: <Unit Use Case>, <Specific Unit Use Case>

Context: none

Status: <This Use Case is fulfilled by the various methods in the objects connected to Ports (i.e. Energy and Information)>

Pre-conditions: <Port is connected to a valid energy or Information Object>

Flow of events: As in [Set Stream Data through Outlet Ports] except: the objects requested are Energy or Information Objects, and the Ports are energy or information Ports.

Post-conditions: <Values have been set>

Errors: <Values could not be set>

Uses: none

Extends: [Set Stream Data through Output Ports]

## UC-027 GET VALUE OF PUBLIC UNIT PARAMETER

Actors: <Simulator Executive>

Priority: <High>

Classification: <Unit Use Case>

Context: none

Status: <This Use Case is fulfilled by the GetParameters method of ICapeUtilities, the Item method of ICapeCollection and the property Value of ICapeParameter>

Pre-conditions: <The Simulator Executive knows the identification used by the Flowsheet Unit for the required Parameter, e.g. by using <Get Public Unit Parameter Names> Use Case.

Flow of events: The Simulator Executive obtains the Parameter Collection from the Flowsheet Unit. The Simulator Executive obtains the Parameter from the Parameter Collection using its identification. The Simulator Executive asks the Parameter for its value. The Parameter returns a value to the Simulator Executive. (Note: the value could be of an unset variable e.g. "EMPTY").

Post-conditions: <Value has been obtained>

Errors: <Identification not recognised by Flowsheet Unit>, <Failed retrieving value>

Uses: none

Extends: none

## UC-028 SET VALUE OF PUBLIC UNIT PARAMETER

Actors: <Simulator Executive>

Priority: <High>

Classification: <Unit Use Case>

Context: context information needed to understand the flow of events

Status: <This Use Case is fulfilled by the GetParameters method of ICapeUtilities, the Item method of ICapeCollection and the property Value of ICapeParameter>

Pre-conditions: <The Simulator Executive knows the identification used by the Flowsheet Unit for the required Parameter, e.g. by using <Get Public Unit Parameter Names> Use Case.

Flow of events: The Simulator Executive obtains the Parameter Collection from the Flowsheet Unit. The Simulator Executive obtains the Parameter from the Collection using its identification. The Simulator Executive passes a value to the Parameter. If the Parameter cannot update its value (e.g. because the value passed is of the wrong type), it informs the Simulator Executive of this.

Post-conditions: <Value has been set correctly>

Errors: <Identification not recognised or is incomplete/bad>, <Cannot update (value of wrong type, value out of range, or variable is read only....)>

Uses: none

Extends: none

## UC-029 GET STREAM DATA FROM INLET PORTS

Actors: <Flowsheet Unit>

Priority: <High>

Classification: <Unit Use Case>, <Specific Unit Use Case>

Context: none

Status: <This Use Case is fulfilled by the various methods in the objects connected to Ports (i.e. Material, Energy and Information)>

Pre-conditions: <Port is connected to a valid object>

Flow of events: The Flowsheet Unit asks each of its input Ports to get the current stream data (energy, material or information) from the Simulator Executive.

Each inlet Port then gets the current values of the connected stream data from the Simulator Executive in CAPE-OPEN format. If required, the Port converts the stream data into the Flowsheet Unit's native format and passes it back to the Flowsheet Unit.

Post-conditions: <Data retrieved>

Errors: <Failed retrieving data (e.g. data not calculated or initialised)>

Uses: none

Extends: none

## UC-030 SET STREAM DATA THROUGH OUTLET PORTS

Actors: <Flowsheet Unit>

Priority: <High>

Classification: <Unit Use Case>, <Specific Unit Use Case>

Context: none

Status: <This Use Case is fulfilled by the various methods in the objects connected to Ports (i.e. Material, Energy and Information)>

Pre-conditions: <Port is connected to a valid object>

Flow of events: The Flowsheet Unit passes its output stream data (information, material or energy) to its outlet Ports (in its own native format) and asks its outlet Ports to update the Simulator Executive's data. The outlet Ports then (if required) perform any necessary conversion of the stream data into CAPE-OPEN format, and pass the output stream data to the Simulator Executive.

Post-conditions: <Data is set>

Errors: <Failed setting data>

Uses: none

Extends: none

## UC-031 PRODUCE UNIT REPORT

Actors: <Reporting sub system>

Priority: <Low>

Classification: <Unit Use Case>, <Specific Unit Use Case>

Context: none

Status: <This Use Case is fulfilled by the following method: ICapeUnitReport::ProduceReport>

Pre-conditions: <The <Define Unit Report> Use Case has been passed>

Flow of events: Produce a report on this Flowsheet Unit. The Reporting sub system requests the Flowsheet Unit to produce a report. The Flowsheet Unit produces the report including the information requested by the [Define Unit Report] Use Case.

Post-conditions: <Report has been produced>

Errors: <The Flowsheet Unit does not produce the report>, <The Flowsheet Unit cannot produce the report currently>

Uses: none

Extends: none

### 3. Analysis and design

#### 3.1 Overview

This chapter introduces the analysis models and the interface design. It contains a number of sequence diagrams as well as a textual description for each method of each interface.

#### 3.2 Sequence diagrams

This section lists the sequence diagrams. These Analysis Sequence Diagrams are included to illustrate how the definitions of some of the critical methods were achieved and to divide the textual description of the more complex Use Cases into smaller actions that occur in a specific order. To keep them as simple as possible, only exceptions that result in non-trivial behaviour are shown and no returning responses from messages sent from one object to another are included.

##### 3.2.1 Retrieve Flowsheet

The following sequence diagram illustrates Use Case UC-020 [Retrieve Flowsheet]. It emphasizes the correct sequence of Load and Initialize calls.

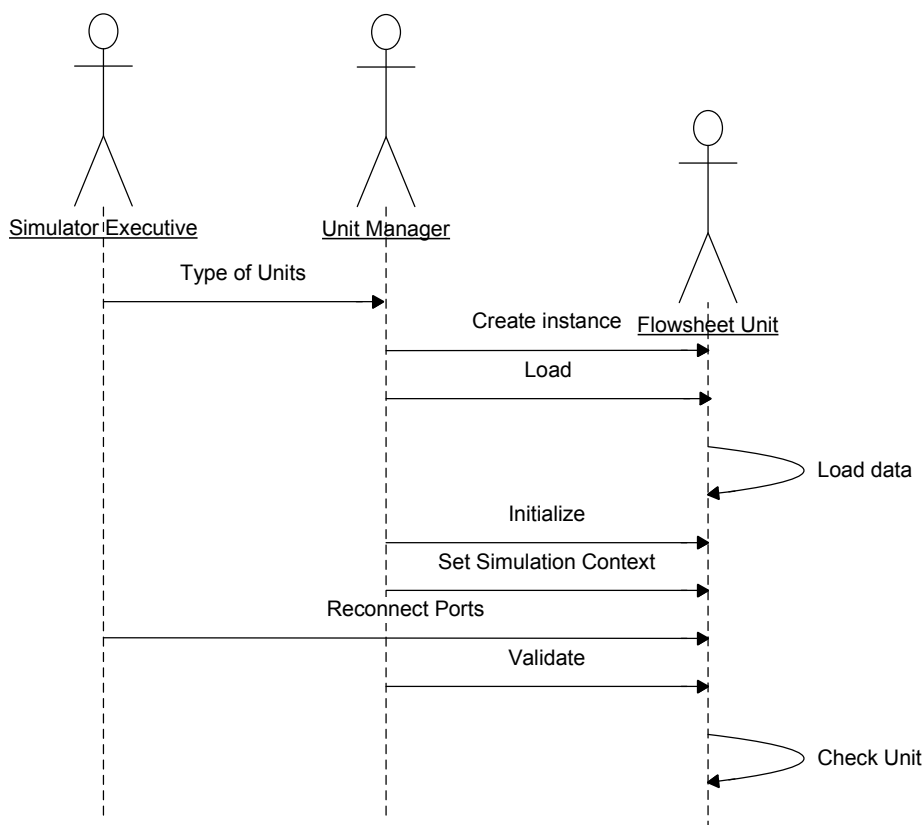


Figure 3-1 SQ-001 Retrieve Flowsheet

### 3.2.2 Add Unit to Flowsheet

The following sequence diagram illustrates Use Case UC-001 [Add Unit to Flowsheet]. The diagram shows in particular the sequence of calls between persistence methods such as `InitNew` and the `Initialize` method of the `ICapeUtilities` interface. The `InitNew` call happens only if the `IPersistStreamInit` interface is implemented by the `Flowsheet Unit`.

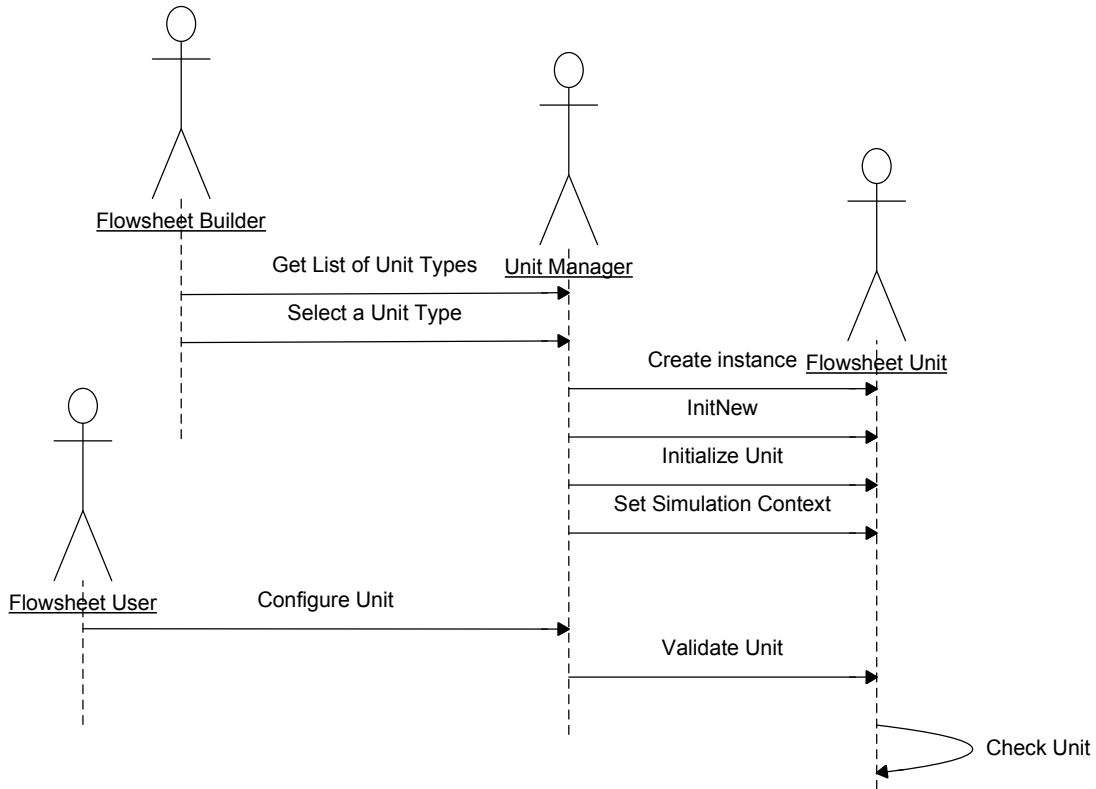


Figure 3-2 SQ-002 Add Unit to Flowsheet

### 3.2.3 Evaluate Unit

The following sequence diagram illustrates Use Case UC-021 [Evaluate Unit]. It indicates in particular the sequence where the Unit retrieves stream data from inlet ports before the core calculations begin and then the Unit stores the results of these calculations as stream data through outlet Ports.

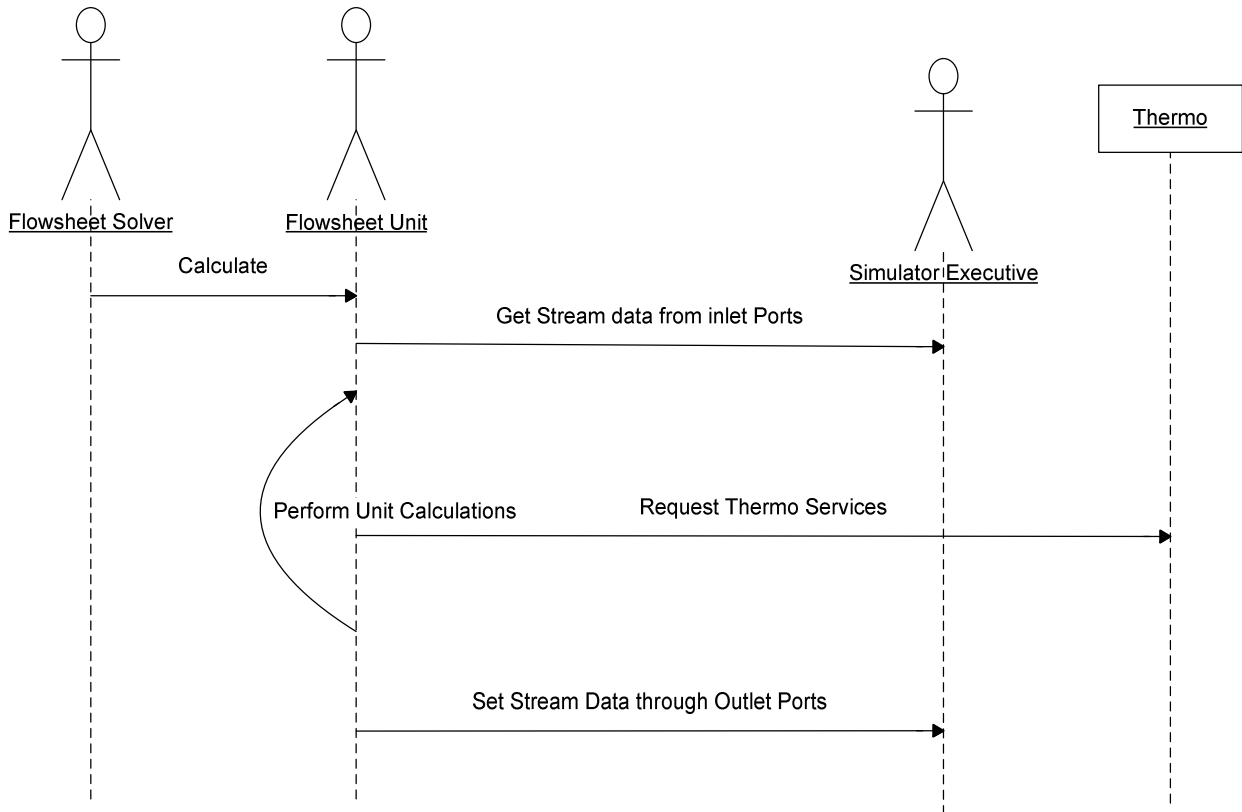


Figure 3-3 SQ-003 Evaluate Unit

### 3.2.4 Reporting

The following sequence diagram illustrates the Use Cases that deal with the reporting mechanism, i.e. the choice of which report to use by selection among the reports proposed by the Unit and then obtain the report when needed.

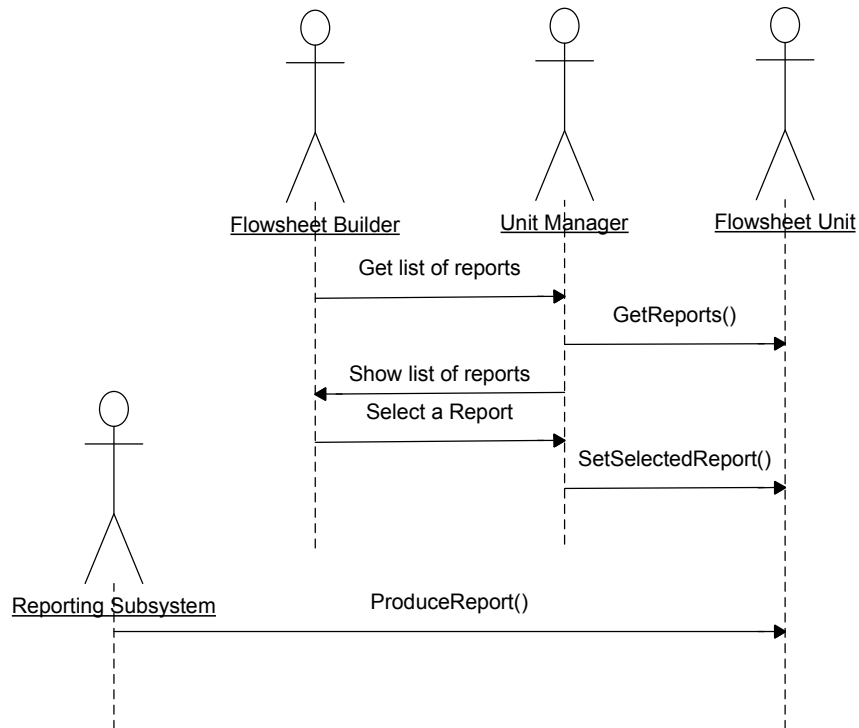


Figure 3-4 SQ-004 Selecting and producing a Report

### 3.3 Interface diagrams

This section presents all interface diagrams.

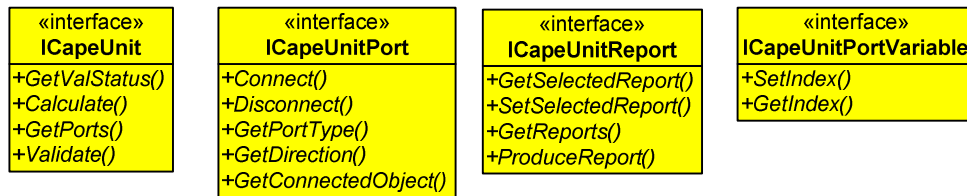
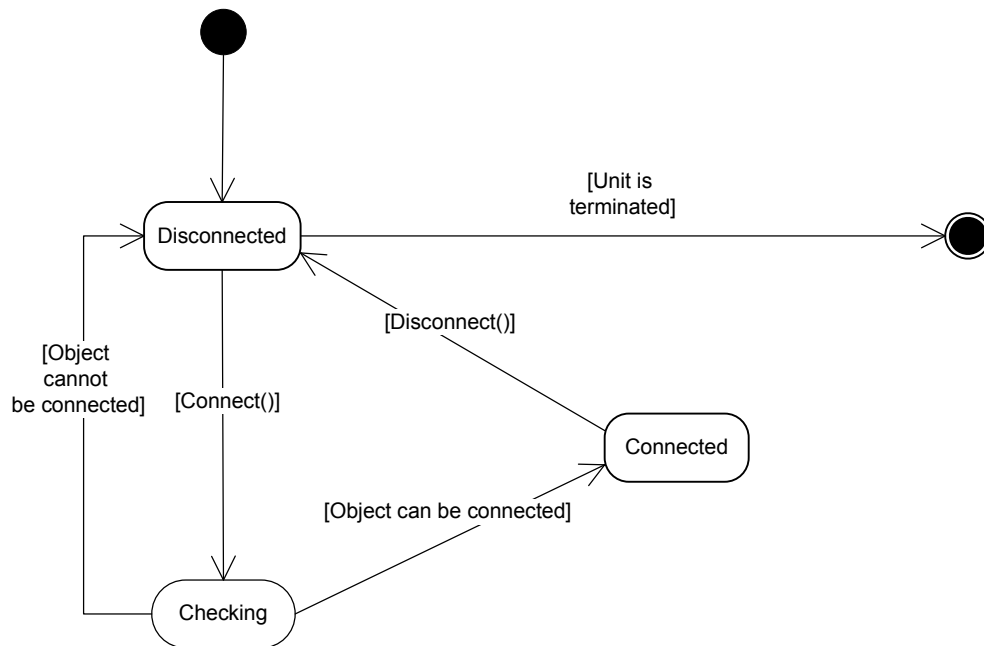


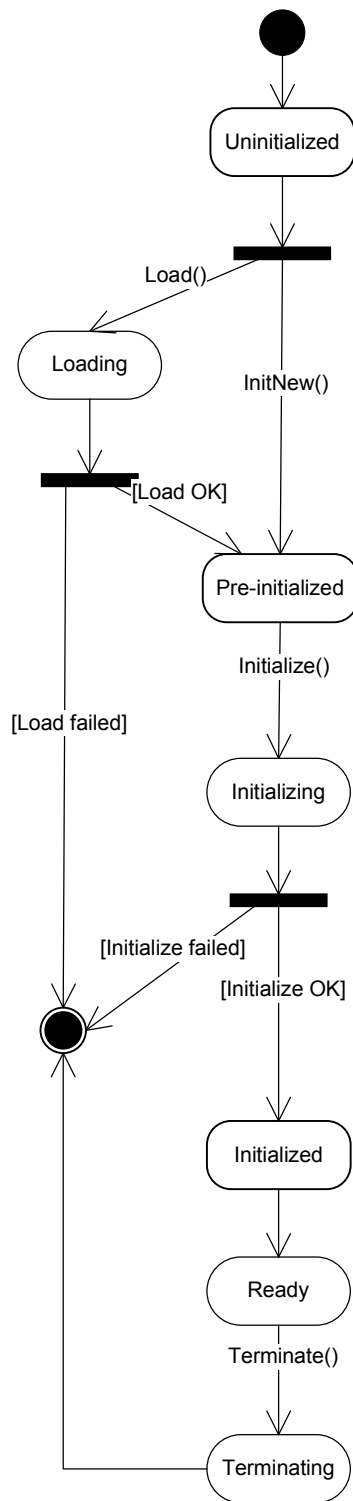
Figure 3-5 Interface diagram for UNIT

### 3.4 State diagrams

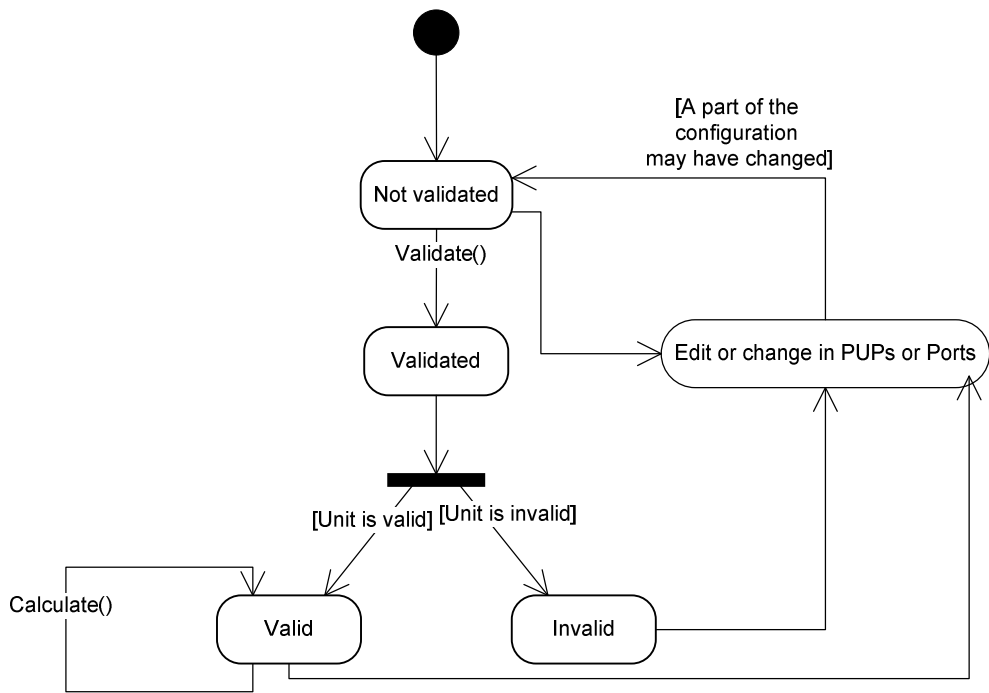
This section presents the Port State diagram, the Initialization State diagram and the Unit Validation State diagram.



**Figure 3-6 Port State diagram**



**Figure 3-7 Initialization State Diagram**



**Figure 3-8 Unit Validation State Diagram**

### 3.5 Other diagrams

A collaboration diagram has been drawn in order to describe which the interfaces on each object are.

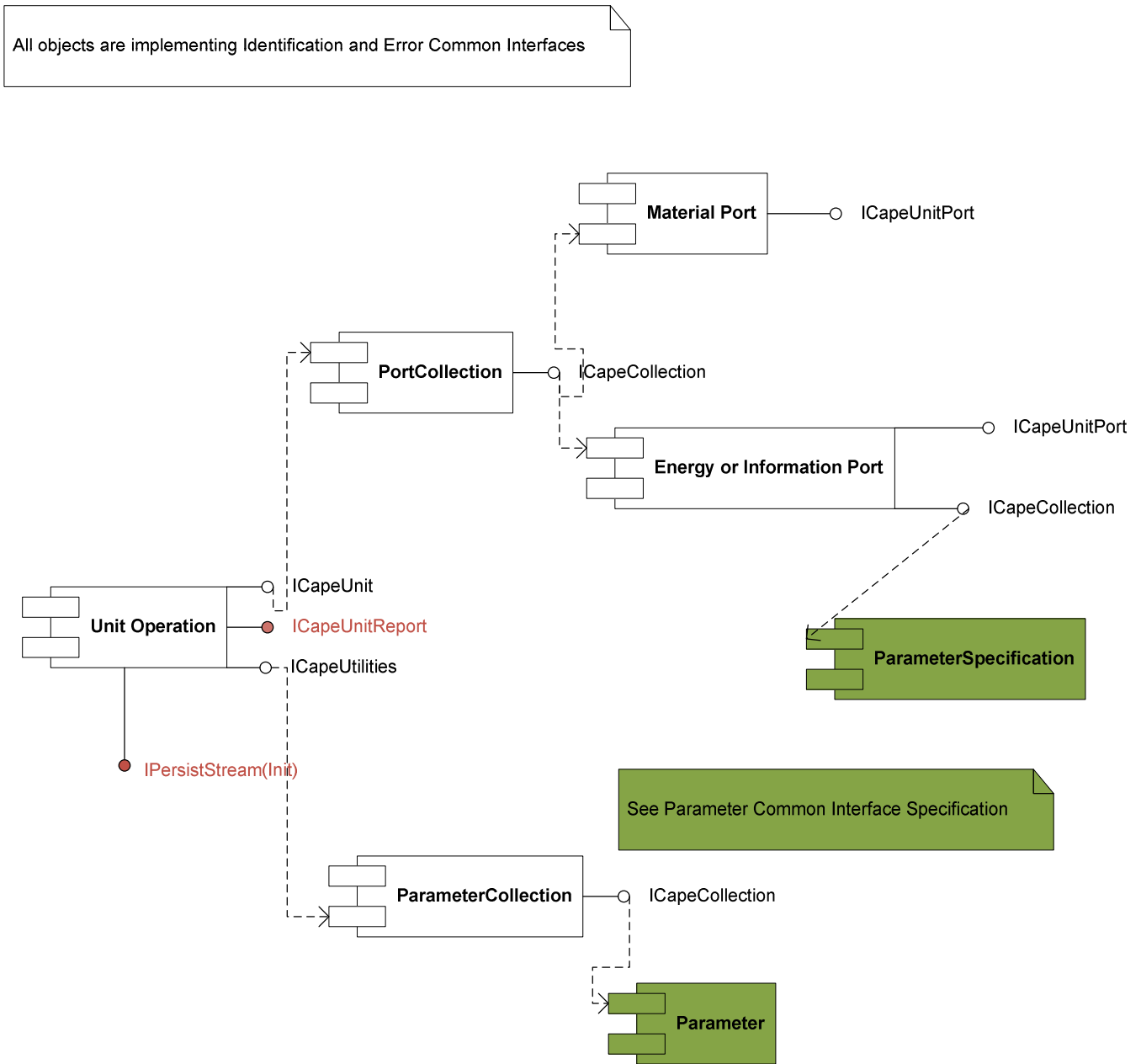


Figure 3-9 Collaboration diagram

The interfaces mentioned in red (ICapeUnitReport and IPersistStream(Init)) are non-mandatory interfaces. The only PMC Primary object in Figure 3.9 is the Unit Operation.

### 3.6 Interfaces descriptions

Each interface is presented together with its corresponding methods. The error list of each method reflects specific use of errors for each method. These lists are not limitative. Any generic error can be returned.

The UNIT interfaces are:

- ❑ **ICapeUnit.** This interface handles most of the interaction with the Unit Operation object. Any Unit Operation exposes at least the interfaces ICapeUnit, ICapeUtilities, ICapeIdentification and error common interfaces corresponding to the error codes returned by the Unit Operation methods.
- ❑ **ICapeUnitPort.** This interface provides access to a Port of a Unit Operation. Ports provide in turn access to Material, Energy and Information Objects provided by the host simulator. Any Port object exposes at least the interfaces ICapeUnitPort, ICapeIdentification as well as appropriate error common interfaces..
- ❑ **ICapeUnitReport.** This interface is optional for a Unit Operation and provides access to the reporting facilities of the Unit Operation.
- ❑ **ICapeUnitPortVariables.** This interface is optional and would be implemented by a Port object. It is intended to allow a Port to describe which equation-oriented variables are associated with it and should only be implemented for the Ports contained in a Unit Operation which supports the ICapeNumericESO interface described in “CAPE-OPEN Interface Specification – Numerical Solvers”.

### 3.6.1 ICapeUnit

Interface Name	ICapeUnit
Method Name	GetValStatus
Returns	CapeValidationStatus

#### Description

Gets the flag that indicates whether the Unit Operation is valid (e.g. if some parameter values have changed but they have not been validated by using Validate, this method will return CAPE\_NOT\_VALIDATED).

#### Arguments

Name	Type	Description
[out, return] ValStatus	CapeValidation Status	This argument has only three possible values: CAPE_VALID means the Validate method returned success CAPE_INVALID means the Validate method returned failure CAPE_NOT_VALIDATED means that the Validate method needs to be called to determine whether the Flowsheet Unit is valid or not.

#### Errors

No specific error

Interface Name	ICapeUnit
Method Name	Calculate
Returns	-

## Description

The Unit Operation performs its calculation, that is, computes the variables that are missing at this stage in the complete description of the objects connected to Outlet Ports and any public parameter value that needs to be displayed.

## Arguments

None

## Errors

ECapeUnknown

ECapeSolvingError: to be used when there is a solving error and to be used also if the calculations are not valid for the given inlet conditions.

ECapeBadCOPparameter: to be used if a Unit Operation parameter is invalid.

EcapeBadInvokationOrder : to be used if Validate has not been called before Calculate.

## Notes

See the Use Case UC-21 Evaluate Unit for an overview of the responsibilities of the Unit during Calculate.

Unit Operations must perform a suitable flash calculation on all Material Objects connected to Outlet Ports.

Validate method must be called at least once before calculation. For further information see Notes of Validate method.

<b>Interface Name</b>	<b>ICapeUnit</b>
Method Name	GetPorts
Returns	CapeInterface

## Description

Returns an interface to a collection containing the Ports (e.g. ICapeCollection) of the Unit Operation.

## Arguments

Name	Type	Description
[out, return] portsInterface	CapeInterface	A reference to the interface on the collection containing the ports

## Errors

ECapeUnknown

<b>Interface Name</b>	<b>ICapeUnit</b>
Method Name	Validate
Returns	CapeBoolean

## Description

Sets the flag that indicates whether the Flowsheet Unit is valid by validating the ports and parameters of the Flowsheet Unit. For example, this method could check that all mandatory Ports have connections and that the values of all parameters are within bounds.

## Arguments

Name	Type	Description
[out, return] isValid	CapeBoolean	TRUE if the Flowsheet Unit is valid.
[out] message	CapeString	An optional message describing the cause of the validation failure

## Errors

ECapeUnknown

ECapeBadCOPparameter

## Notes

During Validate, the Flowsheet Unit may typically obtain information regarding the configuration (not the state) of the objects connected to its ports. For example, it may ask for a list of compounds on each port to see whether a) the unit can deal with this list and b) the compounds on in- and outlet ports are consistent. Therefore, a call to Validate may be relatively time consuming, and it is not recommended to call Validate before each call to Calculate. However, Validate must be called before Calculate in case any (one or more) changes to the following configuration aspects of the Unit Operation have changed since the last call to Calculate:

- inserting the Flowsheet Unit into the flowsheet
- restoring the Flowsheet Unit from file
- connecting an object to a port
- disconnecting an object from a port
- calling the Edit method
- changing a parameter on the Flowsheet Unit via a parameter interface
- changing property package configuration (i.e. changing number of compounds, phases, thermodynamic properties, ...)

The configuration changes that call for Validation do not include the state of object connected to ports, such as the temperature on Material Objects connected to Material Ports, or values of parameters on objects connected to Information Ports. If the calculations are not valid for the given inlet conditions, the Calculate() method is expected to return an appropriate error.

### 3.6.2 ICapeUnitPort

Interface Name	ICapeUnitPort
Method Name	GetPortType
Returns	CapePortType

#### Description

Returns the type of the Port. The returned value has to be one of the constants defined as of the type CapePortType (e.g. CAPE\_MATERIAL, CAPE\_ENERGY or CAPE\_INFORMATION). The type of the Port allows the client to know what type of Objects the Port is expecting.

#### Arguments

Name	Type	Description
[out, return] portType	CapePortType	The type of the port

#### Errors

No specific error

#### Notes

The CapePortType CAPE\_ANY is reserved for future use and should not be used.

<b>Interface Name</b>	<b>ICapeUnitPort</b>
Method Name	GetDirection
Returns	CapePortDirection

## Description

Returns the direction of the Port. This attribute of a Port indicates whether an Object connected to the Port provides data to the Unit or the Unit provides data to the Object connected to the Port: CAPE\_INLET for a Port from which the Unit reads data, CAPE\_OUTLET for a Port to which the Unit writes data

## Arguments

Name	Type	Description
[out, return] portDirection	CapePortDirection	The direction of the port

## Errors

No specific error

## Notes

The CapePortDirection CAPE\_INLET\_OUTLET is reserved for future use and should not be used.

<b>Interface Name</b>	<b>ICapeUnitPort</b>
<b>Method Name</b>	<b>GetConnectedObject</b>
<b>Returns</b>	<b>CapeInterface</b>

## Description

Returns the object that is connected to the Port. A client is provided with the Material, Energy or Information object that was previously connected to the Port, using the Connect method.

## Arguments

Name	Type	Description
[out, return] connectedObject	CapeInterface	The object that is connected to the port. It can be either a Material, Information or Energy Object.

## Errors

No specific error

## Notes

If the Port is not connected, either the connected object is returned as UNDEFINED (NULL or Nothing) or an ECapeUnknown error is returned.

<b>Interface Name</b>	<b>ICapeUnitPort</b>
<b>Method Name</b>	Connect
<b>Returns</b>	-

## Description

Method used by a client, when the client requests that a Port connects itself to the object that is passed in as the argument of the method.

## Arguments

Name	Type	Description
[in] objectToConnect	CapeInterface	The object that is to be connected to the port (i.e. Material, Energy or Information Object).

## Errors

ECapeUnknown

## Notes

Probably, before accepting the connection, a Port will check that the Object sent as argument is of the expected type and according to the value of its attribute portType.

<b>Interface Name</b>	<b>ICapeUnitPort</b>
Method Name	Disconnect
Returns	-

## Description

Disconnects the Port from any object connected to it.

## Arguments

None

## Errors

ECapeUnknown

## Notes

The caller can safely assume after using Disconnect that the Port is no longer connected to any object whether an ECapeUnknown error is returned or not. The only reason for the method to return an error is in the situation where the Port is not connected before the Disconnect method is called. But it is not mandatory for the method to return an error when this situation happens.

### 3.6.3 ICapeUnitReport

Interface Name	ICapeUnitReport
Method Name	GetSelectedReport
Returns	CapeString

#### Description

Returns the name of the active report in the Flowsheet Unit.

#### Arguments

Name	Type	Description
[out, return] selectedReport	CapeString	The name of the active report

#### Errors

ECapeBadInvOrder: the error to be raised if no report has been selected and no default report is available.

#### Notes

If no report has been selected and no default report is available, the method can return an error or UNDEFINED.

<b>Interface Name</b>	<b>ICapeUnitReport</b>
Method Name	SetSelectedReport
Returns	-

## Description

Sets the name of the active report in the Flowsheet Unit.

## Arguments

Name	Type	Description
[in] selectedReport	CapeString	The name of the active report. The name should be in the list of names returned by GetReports.

## Errors

ECapeInvalidArgument: error to be raised when the specified report does not exist.

<b>Interface Name</b>	<b>ICapeUnitReport</b>
Method Name	GetReports
Returns	CapeArrayString

## Description

Returns the list of possible report names.

## Arguments

Name	Type	Description
[out, return] reports	CapeArrayString	The list of possible report names

## Errors

ECapeUnknown

Interface Name	ICapeUnitReport
Method Name	ProduceReport
Returns	CapeString

## Description

Produces the selected report. If no value has been set, the default report is produced if a default report exists

## Arguments

Name	Type	Description
[in, out] report	CapeString	The name of the currently selected report.

## Errors

ECapeBadInvOrder: the error to be raised if no report has been selected and no default report is available.

### 3.6.4 Interface ICapeUnitPortVariables

Interface Name	ICapeUnitPortVariables
Method Name	SetIndex
Returns	-

#### Description

SetIndex is used to set the indices of the variables occurring in a Port.

#### Arguments

Name	Type	Description
[in] variable_type	CapeString	The type of the variable whose index is to be set. The value must be one of the recognised CAPE-OPEN Physical Property names.
[in] Component	CapeString	Component Id if the variable type is requiring component selection (such as fraction), otherwise UNDEFINED.
[in] Index	CapeLong	The index that identifies this variable in the ESO associated with the Unit Operation

#### Errors

ECapeUnknown

ECapeInvalidArgument

#### Notes

Typically the SetIndex method is only used within a Unit Operation. It should not be used by any external software to change the indices of the port variables.

It is expected that a Port will typically contain variables that represent a temperature, a pressure, an enthalpy, a flow rate, and a molar volume, with the composition represented by the molar fraction for each component. Other possibilities are allowed but it is the Simulator Executive responsibility to determine whether it can make a connection to a Port based on the variables that it contains.

Interface Name	ICapeUnitPortVariables
Method Name	GetIndex
Returns	CapeLong

## Description

GetIndex is used to get the index of a specific variable occurring in a Port.

## Arguments

Name	Type	Description
[in] variable_type	CapeString	The type of the variable whose index is to be set. The value must be one of the recognised CAPE-OPEN Physical Property names.
[in] Component	CapeString	Component Id if the variable type is requiring component selection (such as fraction), otherwise UNDEFINED.
[out,return] Index	CapeLong	The index that identifies this variable in the ESO associated with the Unit Operation

## Errors

ECapeUnknown, ECapeInvalidArgument

## Notes

Typically the client of a Unit Operation uses this method when it builds an equation-oriented representation of a simulation problem. The indices of the variables occurring in a Port need to be known in order to create the equations needed to represent connections to the Unit Operation.

It is expected that a Port will typically contain variables that represent a temperature, a pressure, an enthalpy, a flow rate, and a molar volume, with the composition represented by the molar fraction for each component. Other possibilities are allowed but it is the Simulator Executive responsibility to determine whether it can make a connection to a Port based on the variables that it contains.

## 3.7 Scenario

This section describes the scenario of actions that was used to validate the set of interfaces for Unit Operations. This scenario exercises the most important methods of the UNIT interfaces, and so guarantees the core functionality expected from CAPE-OPEN Unit Operations interfaces. It is based on a mixer/splitter component, since this exhibits the interface behaviours of more complex Flowsheet Unit operations, but is simple to implement.

The purpose of this scenario is to focus the scope of the interface specifications onto what is required to implement the mixer-splitter prototype. It provides the overall functionality required for this prototype, rather than an engineering specification of a mixer-splitter.

**For the purposes of this exercise** the following assumptions are made:

- The Unit Operation can ask a material for the value of its enthalpy and get it. The Simulator Executive may not actually store enthalpy with its streams, but it is assumed that the CAPE-OPEN interface supplies it by some means. In other words, we assume that we do not have to understand the normal internal arrangements of the Simulator Executive to obtain any CAPE-OPEN interface attributes.
- The Unit Operation can set the attributes of materials (e.g. T, P etc.) in “batch mode”, that is, an attribute can be set without triggering a recalculation of the material’s internal state. When the Unit Operation has finished setting up a material, it then asks it to recalculate. This is done to make the operation of the Unit Operation efficient by avoiding unnecessary calculations.
- The Unit Operation does not need to inquire about the availability of a thermodynamic server. It just asks the materials to perform actions and the material is assumed to invoke the thermodynamic server internally, if it needs to. In a production simulator, of course, the Unit Operation would need to do this. It is quite possible that there might be no physical property system connected at all. This is just a simplifying assumption for this initial prototype.
- All streams, both inlet, outlet and internal, have the same material template.

The mixer-splitter allows the user to specify the split factors used to direct flow to the outlet streams. It also has the ability to specify heat input.

### **Add a Unit Operation to the Flowsheet**

- Flowsheet Builder selects a mixer-splitter from a palette of unit operations and places it on the flowsheet.
- Simulator Executive asks the Unit Operation to initialise itself.
- The Flowsheet Builder selects or creates the streams and requests the Simulator Executive to connect them, one at a time, to the ports of the Unit Operation, possibly by using a GUI to drag the stream to a port on the Unit Operation icon or by interpretation of a text input file.

### **Enter Unit Operation Specific Data**

- The Simulator Executive does not know if a user interface (UI) is available from the Unit Operation, so that it must query the Unit Operation for one. If a UI is available, it is displayed. If not, the Simulator Executive attempts to construct one from the Unit Operation’s list of Public Unit Parameters.
- When any Unit Operation specific data has changed, the Simulator Executive must validate the Unit Operation before calculating the Unit Operation.

### **Define Unit Operation Report**

- Flowsheet User asks the Unit Operation which reports it can produce.
- The Unit Operation gives a list from which the Flowsheet User selects.

### **Simulation Run**

- Simulator Executive asks the Unit Operation to calculate itself
- The Unit Operation retrieves the Material Objects associated with its streams by requesting them from its Ports.

- The Unit Operation creates an internal Material Object. This is to contain the mixed feeds. It is to be known as the mixed stream. The Unit Operation creates an internal array to contain the mixed composition. This is done to minimise calls to the Material Object to set compositions.
- The Unit Operation retrieves the component flows of each feed from the associated materials and adds them to the mixed stream component flow array. When complete, the composition and total flow are assigned to the mixed stream Material Object.
- The Unit Operation calculates molar enthalpy (J/mol) for each feed stream using a duplicate Material Object of a Material Objects connected to one of the Inlet Ports. Molar enthalpy is multiplied by feed flow rate (mol/s); this is summed up for all inlet streams and added to the heat input (J/s) specified as input parameter. The total molar enthalpy of the mixed stream is calculated by division of the mixed flow rate (mol/s) to give mixed stream enthalpy (J/mol).
- The Unit Operation sets the pressure of the mixed stream Material Object to the minimum of the feed pressures.
- The Unit Operation asks the mixed stream Material Object to calculate its temperature (PH flash).
- The Unit Operation assigns molar flows to the Material Objects connected to the Outlet Ports in accordance with the specified split factors.
- The Unit Operation assigns the composition of the mixed stream to the Material Objects connected to the Outlet Ports.
- The Unit Operation assigns the pressure and temperature of the mixed stream to the Material Objects connected to the Outlet Ports.
- The Unit Operation requests the Material Objects connected to the Outlet Ports to calculate equilibrium for specified conditions (TP).

## **Reporting**

- Simulator Executive asks the Unit Operation to produce its report.

## **4. Interface specifications**

The developers of CAPE-OPEN compliant components should use the formal libraries from implementation specification category available on the CO-LaN website.

## **5. Notes on the interface specifications**

There is no note recording the rationale for the decisions made in designing the interfaces, methods and attributes.

## 6. Prototype implementation

A prototype in Microsoft Visual Basic 6 and a prototype in Microsoft Visual C++ 2005 are available from the CO-LaN website.

## 7. Guidelines on versioning

Experiences with interoperability testing revealed a shortcoming of the current mechanism used to communicate which version of the CAPE-OPEN interfaces (IDL/type library) an object supports. For example, there is no way for a process modeling environment (PME) to know which versions of the thermodynamic specification that a process modeling component (PMC) such as a Unit Operation supports. As a result, a unit operation PMC that verifies the type of Material Object being connected to its ports would refuse to accept the connection if the Material Object does not support the correct version of thermodynamics.

Currently, the version of CAPE-OPEN supported by a PMC is indicated through the CapeVersion registry key in Microsoft COM implementations. While this key provides general information about the overall CAPE-OPEN support of an object, it does not provide the granularity needed to indicate support for multiple versions of CAPE-OPEN, as in the case presented by the release of Thermodynamics version 1.1. As a result, a new versioning scheme needs to be developed to provide information regarding which versions of individual standards an object supports.

The 'Implemented Categories' registry key is a mechanism to provide more granular information about version support to the PME and other objects. Currently, 'Implemented Categories' is used primarily to distinguish between types of PMCs; for instance, a unit operation PMC will include Category Unit GUID (CATID) of {678C09A5-7D66-11D2-A67D-00105A42887F}. This is sufficient to inform the PME that the PMC supports CAPE-OPEN version 1.0 Unit Operation specifications. However, this does not provide the PME, or other objects, with any information regarding whether this Unit Operation is capable of supporting Thermodynamics version 1.0 or 1.1. Using additional CATID values, as described below, will enable the object to provide more granular version information. For example, exposing the proper CATID combinations will enable a Unit Operation PMC to clearly indicate that it only supports version 1.1 of the CAPE-OPEN Thermodynamics standards.

It is useful for the PME to know whether a Unit Operation needs access to the thermodynamic subsystem. In order to provide this information, the Unit Operation should be registered with the following CATID:

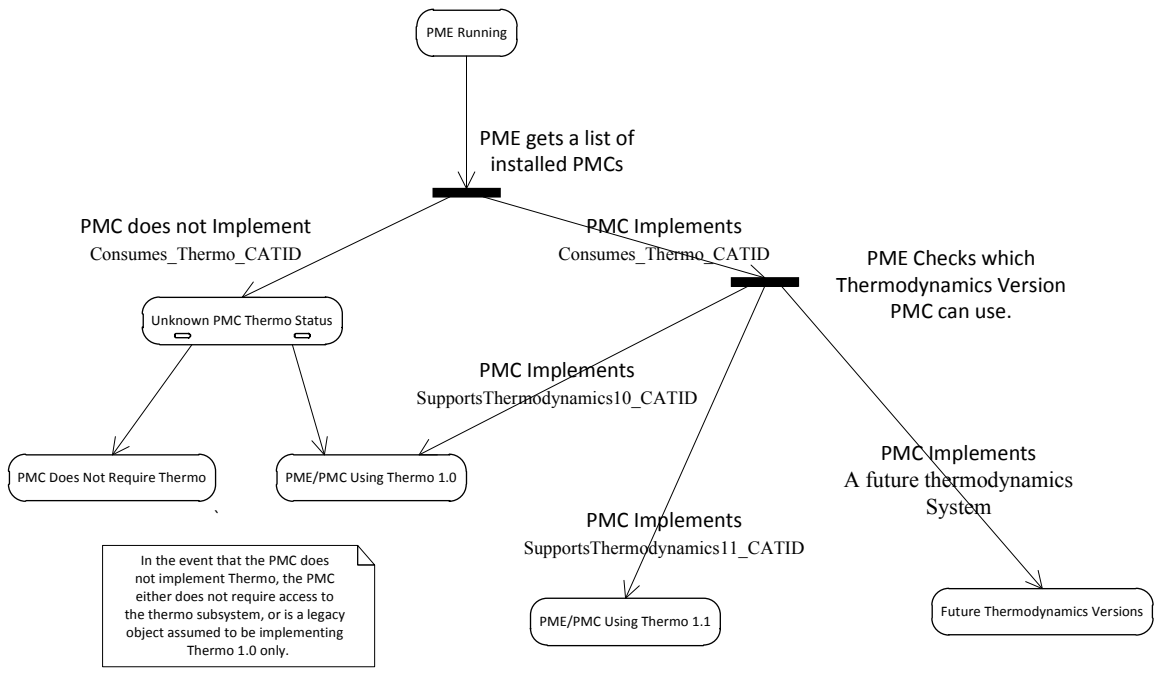
Consumes\_Thermo\_CATID: {4150C28A-EE06-403f-A871-87AFEC38A249}

Presence of Consumes\_Thermo\_CATID informs the PME that the Unit Operation will require thermodynamic interfaces specified further by the CATIDs listed below. In the event that the Unit Operation does not indicate that it consumes thermodynamics, the Unit Operation either will not require access to the thermodynamic subsystem, or is a legacy object that will only access thermodynamics using interfaces under version 1.0 of the CAPE-OPEN standards. Unit Operations that consume thermodynamics will use the following two new CATIDs to indicate which version(s) of the thermodynamic interfaces are supported:

SupportsThermodynamics10\_CATID: {0D562DC8-EA8E-4210-AB39-B66513C0CD09}

SupportsThermodynamics11\_CATID: {4667023A-5A8E-4CCA-AB6D-9D78C5112FED}

Based upon which registry key is present, the PME can choose which version of the thermodynamic interfaces is most appropriate to interact with the Unit Operation through, as shown on **Erreur ! Source du renvoi introuvable.**



**Figure 7-1 State Chart Showing Thermodynamic Specification Usage for Different Implemented Category Combinations**

## 8. Specific glossary terms

The terms used in this document are in accordance with the CAPE-OPEN Project Glossary.

**Material Object:** Object provided by the Simulator Executive, to be connected to Unit Operation Material Ports, and additionally to be used by the Unit Operation to perform thermodynamic and physical property calculations or thermodynamic phase equilibrium calculations. The major task of the Material Object is storing properties and calculation results. Material Objects connected to Material Ports typically represent a material stream on a flowsheet level.

**Information Object:** Object provided by the Simulator Executive, to be connected to Unit Operation Information Ports. The major task of the Information Object is storing information content. Information Objects connected to Information Ports typically represent an information stream on a flowsheet level.

**Energy Object:** Object provided by the Simulator Executive, to be connected to Unit Operation Energy Ports. The major task of the Energy Object is storing energy content. Energy Objects connected to Energy Ports typically represent an energy stream on a flowsheet level.

**Simulator Executive:** this is the core part of any Process Modelling Environment. It provides services that a Unit Operation is likely to need from any process simulator, such as stream information and the Simulation Context.

**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.

## 9. Bibliography

The bibliography is organised in three sections as shown below. In the process simulation references are listed papers from various organizations who have implemented the CAPE-OPEN Unit Operation interface specification: Alstom Power, IFP Energies Nouvelles, TOTAL SA, Università degli studi di Trieste, ANSYS Inc., Aspentech Inc., National Energy Technology Laboratory, US Environmental Protection Agency, etc...

### 9.1 Process simulation references

Barrett, William M. and Yang, Jun (2005), "Development of a chemical process modeling environment based on CAPE-OPEN interface standards and the Microsoft .NET framework". *Computers and Chemical Engineering* 30, pp. 191-201

Belaud J.-P., Roux P. and Joulia X., Opening Unit Operations For Process Engineering Software Solutions, Chemical Engineering Transactions, Vol. 3, pp. 1069-1074, 2003. ISBN 88-900775-2-2. Edited by Sauro Pierucci. AIDIC Servizi S.r.l. Publisher. Printed in Italy.

Domancich A., Perez V., Hoch P. and N. B. Brignole, Systematic generation of a CAPE-OPEN compliant simulation module from GAMS and FORTRAN models, Chemical Engineering Research & Design, 2010, vol. 88, n°4, pp. 421-429

Fermeglia, M., Longo, G., and Toma, L., 2008, "COWAR: A CAPE OPEN Software Module for the Evaluation of Process Sustainability". *Environmental Progress* 27, No.3.

Fermeglia, M., Longo, G. and Toma, L., 2009, "Computer Aided Design for Sustainable Industrial Processes: Specific Tools and Applications". *AIChE Journal*, Vol. 55, No. 4, 1065.

Floquet P., Joulia X., Vacher A., Gainville M. and Pons M. (2009) Numerical and Computational Strategy for Pressure-Driven Steady-State Simulation of Oilfield Production. *Computers & Chemical Engineering*, vol. 33 (n° 3). pp. 660-669. ISSN 0098-1354.

Gainville M., Roux P., M. Pons, Ricordeau A. and D. Paen (2007). Integrated Flow Modeling Platform Using CAPE-OPEN standard. Paper 110864-MS, SPE Annual Technical Conference and Exhibition, 11-14 November 2007, Anaheim, CA.

Gassies M., Ricordeau A, Mohan R.S.and O. Shoham (2008). GLCC CAPE-OPEN compliant Unit Operation. Paper 397, *AIChE Annual Meeting 2008*, Philadelphia, PA.

Morales-Rodriguez R., Gani R., Dechelotte S. Vacher A. and Baudoin O., Use of CAPE-OPEN standards in the interoperability between modelling tools (MoT) and process simulators (Simulis Thermodynamics and ProSim Plus), Chemical Engineering Research and Design, Volume 86, Issue 7, July 2008, Pages 823-833

Osawe, M.O, P. Felix, M. Syamlal, I. Lapshin, K.J. Cleetus, and S.E. Zitney, "An Integrated Process Simulation and CFD Environment Using the CAPE-OPEN Interface Specifications," Presented at the *AIChE 2002 Annual Meeting*, November 3-8, Paper No. 250c, Indianapolis, IN (2002).

Pons M., Industrial Implementations of the CAPE-OPEN Standard, AIDIC Conference Series, Vol. 6, pp. 253-262, 2003. ISBN 0390-2358. Reed Business Information Publisher. Printed in Italy.

Ricordeau A., Sarica C. and A. Mathieu (2007). Development of TUWAX as a CAPE-OPEN compliant process modelling environment. Paper 459c, *AIChE Annual Meeting 2007*, Salt Lake City, UT.

Sloan, D.G., W.A. Fiveland, S.E. Zitney, and M. Syamlal, "Software Integration for Power Plant Simulations," *Proc. of the 27th International Technical Conference on Coal Utilization & Fuel Systems*, March 4-7, Clearwater, FL (2002).

Sloan, D.G., W.A. Fiveland, S.E. Zitney, and M. Syamlal, "Power Plant Simulations Using Process Analysis Software Linked to Advanced Modules," *Proc. of the 29th International Technical Conference on Coal Utilization & Fuel Systems*, April 18-22, Clearwater, FL (2004).

Van Baten, J. and R. Szczepanski, A thermodynamic equilibrium reactor model as a CAPE-OPEN unit operation, *Computers & Chemical Engineering*, doi:10.1016/j.compchemeng.2010.07.016

Zitney, S.E., M.T. Prinkey, M. Shahnam, and W.A. Rogers, "Coupled CFD and Process Simulation of a Fuel Cell Auxiliary Power Unit," *Proc. of the ASME Second International Conference on Fuel Cell Science, Engineering, and Technology*, Eds. R. Shah and S.G. Kandlikar, June 13-16, Rochester NY, Paper 2490, pp. 339-345 (2004).

Zitney, S. E. and M. Syamlal, "Integrated Process Simulation and CFD for Improved Process Engineering," *Proc. of the European Symposium on Computer Aided Process Engineering-12, ESCAPE-12*, May 26- 29, The Hague, The Netherlands, eds. J. Grievink and J. van Schijndel, pp. 397-402 (2002).

## **9.2 Computing references**

## **9.3 General references**

## 10. Appendix: Equation-oriented simulation

### 10.1 Introduction

In this appendix, we consider the extension of the Unit Operation interfaces to handle equation-oriented simulation, both steady-state and dynamic.

This analysis was developed in close collaboration between the Unit Work Package and the Numerical Work Package during the CAPE-OPEN project and confirmed that only minor extensions would be needed to the work already done by both Work Packages. However, time has not permitted us to build a prototype demonstration, which is why this chapter is contained in an appendix.

The chapter summarises an analysis of the equation-oriented approach, maps it onto existing Unit Operations and Numerical concepts and then presents a mixer/splitter scenario.

The analysis provides an overview of the types of equations set up by a typical EO unit, the functionality of the Equation Set Object (ESO) defined by the Numerical work-package (NUMR) and the way that an EO simulator brings together the individual unit operations to create a complete flowsheet.

### 10.2 An Equation-oriented Unit Operation

This appendix is developed around a simple unit operation, a mixer/splitter example. As before, this provides good insights into the issues involved, without becoming distracted by the complexities of the unit operation itself.

#### 10.2.1 Equations for a typical EO Unit Operation

For a simple tank, with an inlet flow, and an outlet flow governed by the liquid head, then a typical set of equations would be:

$$\frac{dM}{dt} = F_{in} - F_{out}$$

$$\rho g h = \frac{4 f L v^2}{2 D}$$

$$\rho A h - M = 0$$

$$F_{out} = \rho \frac{\pi D^2}{4} v$$

for laminar flow:

$$f = \frac{16}{\text{Re}}$$

for turbulent flow:

$$\frac{1}{\sqrt{f}} = -4 \log_{10} \left( \frac{1}{\text{Re} \sqrt{f}} + \dots \right)$$

with the switch from laminar to turbulent flow occurring at  $\text{Re}=2100$ , and the reverse at  $\text{Re}=2000$ .

## 10.2.2 The Equation Set Object (ESO)

All the features of this set of equations, including the discontinuity in the equation for the friction factor,  $f$ , can be handled by the ESO defined by CAPE-OPEN Numerical interfaces. The ESO allows the description of a set of equations:

$$f(x, \dot{x}, t) = 0$$

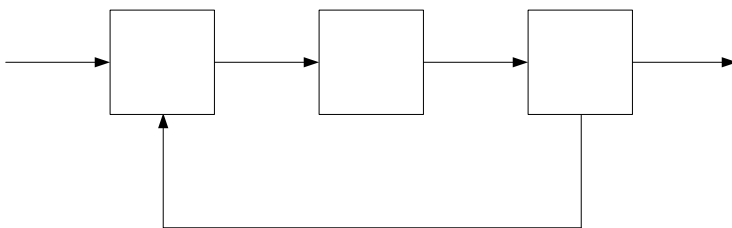
and provides methods for (amongst others):

- Get the number of variables
- Get the number of equations
- Get / set values of  $x$ ,  $\dot{x}$  and  $t$
- Get values of functions
- Get number of non-zero derivatives
- Get structure of derivatives (Jacobian)
- Get / Set values of derivatives

Note that values of  $x$  are simply stored as a vector; individual values are accessed using the location within the vector.

## 10.2.3 Operation of an EO simulator

For the simple flowsheet below:



an EO simulator would generate the overall set of variables and equations by looking at each unit operation in turn to identify the variables and equations associated with the units themselves, then would add the connectivity equations and the user specification equations.

For example, in an EO simulator (here assumed to be working in steady-state, but extendible to dynamics) a single Unit Operation (hereafter simply called Unit) will describe a set of equations:

$$\underline{f}(\underline{x}) = \underline{0}$$

where  $\underline{x}$  is the vector of all the variables associated with the Unit. This vector can be sub-divided into 3 categories:

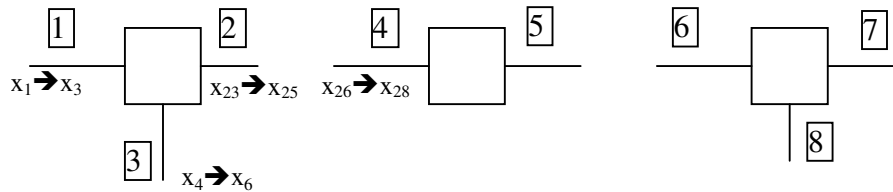
- The ‘input’ variables  $\underline{u}$ . These would include variables such as feed stream data (global variables) and Unit input data (Public Unit Parameters).
- The ‘output’ variables  $\underline{y}$ . These would include variables such as product stream data (global variables) and calculated results that are made available to the Simulator Executive (Public Unit Parameters).
- Internal variables  $\underline{x}'$ , not visible to the Simulator Executive, other than as an element of the vector  $\underline{x}$ .

Thus:

$$\underline{x} = [\underline{u}, \underline{x}', \underline{y}]$$

Note again that the ESO simply stores  $\underline{x}$  as a vector, and there is no information stored as to the type or meaning of a given element of the vector. Furthermore, in general for a single Unit there will be fewer equations than there are variables, as many of the equations involving the input variables  $\underline{u}$  can only be defined at the flowsheet level.

Returning to our simple flowsheet, let’s assume that each stream has 3 variables associated with it and that each unit operation has just 1 item of input data. Let’s further assume that the first Unit has a total of 25 variables ( $x_1$  to  $x_{25}$ ). Of these,  $x_1$  to  $x_6$  are the variables associated with the feed streams,  $x_{23}$  to  $x_{25}$  are the variables associated with the product stream, whilst  $x_{10}$  is the Unit input data. Finally, the Unit itself is described by a total of 18 equations. A similar analysis can be performed for each of the other Units in the flowsheet. Let’s say that the 2<sup>nd</sup> Unit has a total of 15 variables and 11 equations, whilst the 3<sup>rd</sup> has 40 variables and 36 equations.



A typical EO Simulator Executive will obtain this information from the ESO and will construct variable and equation arrays for the overall flowsheet:

Variables	1-25	26-40	41-80
Equations	1-18	19-29	30-65 66-80

Thus, in this case, the Simulator Executive will add a further 15 equations (equations 66-80) to describe the overall flowsheet connectivity and the Unit Operation input data. These equations are derived from:

stream 1 = plant feed stream data (e.g.  $x_1 = \text{value}$ )

stream 4 = stream 2 (e.g.  $x_{26} = x_{23}$ )

stream 6 = stream 5

stream 3 = stream 8

and

$x_j = \text{Unit input data}$  (e.g.  $x_{10} = \text{value}$ )

where there are 3 equations for each stream and 1 for the input data for each Unit Operation.

### 10.2.4 Proposal

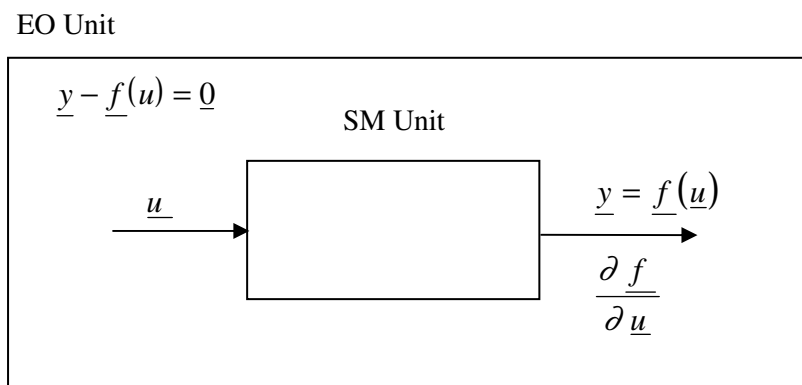
It is proposed that the variables and equations for each Unit can be described by a ESO generated by that Unit. Furthermore, the Simulator Executive will be able to generate an overall ESO by amalgamating the individual Unit ESO's with the additional equations required to describe the flowsheet connectivity and Unit input data. However, as each individual ESO contains a simple vector representation of the variables for the Unit, methods must be available to determine the association of the Unit variables with the flowsheet streams and with the Unit input data.

Thus an EO unit will consist of the following:

- An ESO generated by the Unit, but exposed via the standard ESO interfaces.
- Methods to allow the user to provide Unit input data (as per current SM Unit interfaces)
- Methods to allow production of reports (as per current SM Unit interfaces)
- Methods to allow the association of an internal ESO variable to a Global Variable (from the Material Object).
- Methods to allow the association of an internal ESO variable to a Public Unit Parameter. Note that all internal Unit variables which can be specified by the user must be available as PUPs. Other internal variables are at the discretion of the Unit developer.

### 10.2.5 Interoperability of SM and EO Unit Operations

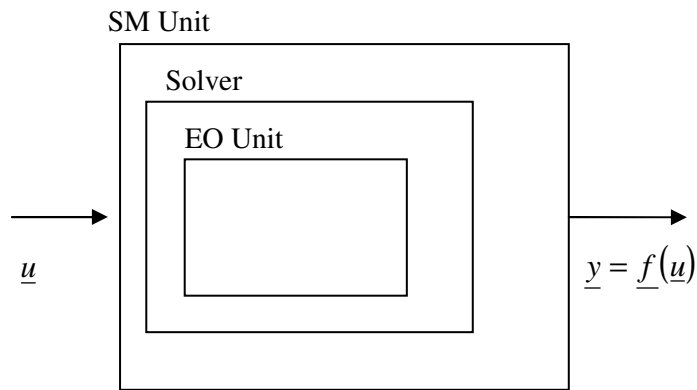
A CAPE-OPEN compliant SM unit operation can be used in an EO simulator as follows:



This implies that the SM CAPE-OPEN interface specification must make provisions for the derivatives to be accessed, if available. This is a new method, which is not covered by the current specification. Of course, individual implementations of such interfaces may choose not to provide such derivative information; in this case, the EO wrapper object will be unable to provide derivatives to its clients.

Furthermore, a legacy SM Unit operation can be used within an EO simulator by first wrapping as a CAPE-OPEN compliant SM unit operation, then using the method above.

Using a CAPE-OPEN compliant EO Unit Operation with a SM simulator is also possible, as below:



### 10.2.6 Other issues

EO thermodynamics and sub-flowsheets can both be handled as sub-ESO's, which can then be amalgamated into the overall flowsheet ESO.

The requirement to be able to identify the name of each of the variables within an ESO has been identified. While this is not strictly necessary mathematically, it enables a solver to give more meaningful error messages and is thus essential from a practical point of view.

### 10.2.7 Extra methods required for UNIT and NUMR

In summary, the analysis has shown that equation-oriented simulation can be handled by the existing Unit Operation and Numerical interfaces, with the addition of a small number of extra methods. These are:

- ESO: get names of variables
- SM Unit: get derivatives  $df/du$
- EO unit: get internal variable number for given Global Variable
- EO unit: get internal variable number for given Public Unit Parameter

Note that this also implies changes to the ESO to deal with the variable names.

### 10.2.8 Mixer-splitter design scenario for steady-state EO Unit and Simulator

We are now able to revise the design scenario used earlier for steady-state, sequential modular simulation for steady-state EO simulation. As before, the purpose of the scenario is to focus the scope of the interface specifications onto what is required to implement a steady-state mixer-splitter prototype in an EO simulator. It provides the overall functionality required for this prototype, rather than an engineering specification of a mixer-splitter.

**For the purposes of this exercise** the following assumptions are made:

- The Unit relies on the use of a “conventional” physical properties package to compute any physical properties (e.g. enthalpies) that appear in its equations. In a CAPE-OPEN context, this is done via the Material Object.
- The Unit can set the attributes of materials (e.g. T, P etc.) in “batch mode”, that is, an attribute can be set without triggering a recalculation of the material's internal state. When the Unit has finished

setting up a material, it then asks it to recalculate. This is done to make the operation of the Unit efficient by avoiding unnecessary calculations.

- The Unit does not need to inquire about the availability of a thermodynamic server. It just asks the materials to perform actions and the material is assumed to invoke the thermodynamic server internally, if it needs to. In a production simulator, of course, the unit would need to do this. It is quite possible that there might be no physical property system connected at all. This is just a simplifying assumption for this initial prototype.
- All the Material Objects connected to the Unit Ports derived from the same material template.
- The mixer-splitter allows the user to specify the split factors used to direct flow to the outlet streams. It also has the ability to specify heat input.

### **Add a Unit to the Flowsheet**

- User selects a mixer-splitter from a palette of unit operations and places it on the flowsheet.
- The Simulator Executive asks each unit to initialise itself passing to it a material template. The Unit uses the latter to create one or several Material Objects and interacts with one of them to obtain the number of components in the system. It uses this information to set up an ESO (Equation Set Object) describing the mathematical system of equations that models the Unit's operation.
- The user selects or creates the streams and requests the simulator to connect them, one at a time, to the ports of the Unit, possibly by using a GUI to drag the stream to a port on the Unit icon or by interpretation of a text input file.

### **Enter Unit Specific Data**

- The Simulator Executive does not know if a user interface (UI) is available from the Unit, so that it must query the Unit for one. If a UI is available, it is displayed. If not, the Simulator Executive attempts to construct one from the Unit's list of Public Unit Parameters.
- When any Unit Operation specific data has changed, the Simulator Executive must validate the Unit Operation before calculating the Unit Operation. Note that an EO Unit does not necessarily require all the possible data to be specified, unlike a SM unit.

### **Define Unit Report**

- User asks the Unit which reports it can produce.
- The Unit gives a list from which the user selects.

### **Run a simulation**

#### *Building the mathematical problem*

- The Simulator Executive asks the ESO of each Unit in the flowsheet for the numbers of equations and variables in it and uses them to construct the global vectors of equations and variables for the entire flowsheet. The simulator also asks the Unit ESO for the number of non-zero partial derivatives, the sparsity structure of the partial derivative matrix (i.e. the row and column numbers of each element) and the possibility of computing the exact value of each element.
- The Simulator Executive sets up the equations that describe the connectivity of the Units in the flowsheet and the corresponding rows of the partial derivative matrix. To do this, the Simulator Executive asks the Unit to identify which of its variables are associated with each of the connected streams.

- The Simulator Executive sets up the specification equations and the corresponding rows of the partial derivative matrix. To do this, the Simulator Executive asks the Unit to identify which of its variables are fixed and at what values.

#### *Analyzing and validating the mathematical problem*

- The Simulator Executive checks that the mathematical problem assembled at the previous step is well-posed (e.g. square, structurally non-singular, etc.). If not, the Simulator Executive issues some form of diagnostic information to the user.

#### *Solving the mathematical problem*

- The Simulator Executive creates an ESO (Equation Set Object) which incorporates all the information on the mathematical problem assembled.
- The Simulator Executive calls an NLASystemFactory, passing to it the above ESO, to create an NLASystem. The NLASystem is the combination of a solver and the specific mathematical problem.
- The Simulator Executive invokes the NLASystem to solve the simulation problem.
- During the numerical solution, each of the following interactions may occur in any order and frequency, depending on the design of the solver:
  - The NLASystem asks its ESO for its current variable values. The ESO obtains these by querying each unit ESO in turn for its own variables, and then assembling these into a global vector which it returns to the NLASystem. *Note:* this is likely to be one of the first actions performed by any NLASystem, since it can only obtain initial estimates for the variables from its ESO. Thus these initial estimates must be present in, or be generated by, the units before the solution process commences. (The exact point in the sequence at which these estimates are produced will depend on the implementation of the component).
  - The NLASystem provides new variable values to its ESO. The ESO then distributes these to each unit ESO in turn, updating the local values formerly held by the units.
  - The NLASystem requests the current residual values from its ESO. The ESO obtains most of these by requesting them from the ESOs of its constituent units, and assembles the responses into a global vector. It then appends to this vector the values of the residuals for the connectivity and specification equations, before returning the global vector to the NLASystem.
  - The NLASystem requests all the current partial derivative values that can be computed by each ESO. Again these are obtained from the individual units and assembled into a global vector, with information for the connectivity and specification equations appended to it. The ESO then returns this global vector to the NLASystem. (It is assumed that the NLASystem will approximate those partial derivatives that cannot be computed by the units).

#### **Report results**

- The Simulator Executive asks the Unit to produce its report.