WP Validation Deliverable D 521: Report on Integration

CAPE-OPEN
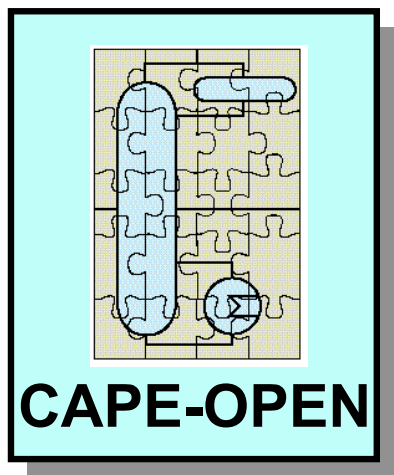
S Artlich
C Kuhlmann
P Roux

Archival Information

Reference	CO-VALI-BY-03
Authors	S. Artlich
	C. Kuhlmann
	P. Roux
Date	June 23, 1999
Number of Pages	35
Version	Version 2
Filename, short	D521 Integration Report.doc
Filename, long	
Location:	
Electronic	
Hardcopy 1	
Hardcopy 2	
Reviewed by	C. Kuhlmann, P. Roux, Ph. Muller
Approved by	
Distribution	VALI
Originating Organisations	
Reference Number	

Summary

In a standardization project like CAPE-OPEN a number of validation tasks have to be completed to demonstrate that a standard has been defined which can be accepted without further revision. For CAPE-OPEN, the main validation tasks include:

- Prepare a suite of test problems

- Test these problems using existing simulators

- Repeat tests with CAPE-OPEN compliant interfaces

- Test ability of independent authors to write conforming components

- Prepare test harnesses to validate basic interface conformance

- Test components against harnesses (and vice-versa)

- Test ability to interface components from independent authors

- Test (through scenarios based on Use Cases) that CAPE-OPEN essential requirements are met through the interfaces defined.

Further tasks include the review of the interface documentation which were allocated to the three standardization work packages. In this sense, validation is not just the task of one individual Validation work package but a collaborative over-all project issue. The current document concentrates on the last four tasks from the list above: test harnesses, test components, test from independent authors, and relating tests to basic User Requirements. For the set up of the more detailed validation programme, it has been decided to focus the effort on the definition of the minimum set of tasks necessary to interchange a vapour-liquid mixer-splitter unit together with relevant thermodynamics. Further tasks, which are desirable before any standard can be considered adequate, are deferred to appendices. Although concentrating on the mixer-splitter example, similar tasks have to be undertaken for the validation of those CAPE-OPEN interfaces which are not used in the presentation example. In this sense, the document has broader application than the mixer/splitter application alone.

In this document, we concentrate only on those tasks essential to demonstrate that CAPE-OPEN has developed a workable standard, not on those necessary to demonstrate that the standard can be accepted without further revision. Due to resource limitations, tasks such as employing independent authors to verify clarity of definition had to be reduced. The validation programme was designed to achieve two primary objectives. The first objective is to verify that the defined CAPE-OPEN interfaces are capable of transmitting all the information necessary to perform a correct simulation. The second objective is to verify that the standard interfaces are unambiguously defined and sufficient to support those additional features, such as error tracking, which are essential to the end user.

Contents

1 Introduction

Any software that has not been tested probably does not work. The same goes for software standard interfaces. Standard interface testing cannot be left entirely to the original conforming software authors. These authors must, of course, make the initial tests. They, however, will have both sides of the interfaces available to them and, despite any shortcomings in the standard, will inevitably produce code that interoperates successfully. The objective of a formal validation exercise is to take code produced by different authors, working independently, and test it for interoperability. The test programme is designed to verify both that the standards are unambiguously defined and that they meet the users requirements. The important tests are first that the interfaces are capable of transmitting all the required information, and secondly that independent authors are able to prepare components that interoperate successfully. This document describes the steps to be taken to achieve these goals.

The document is broken down as follows. In Chapter 2, we describe the major software components. In Chapter 3, we explain the major interfaces that need to be standardized and relate these back to the Use Cases and User Requirements that give rise to them. In Chapter 4, we describe the general validation strategy. Chapters 5 develops the concept of test harnesses. Chapter 6 outlines the steps to be taken in progressing from test harnesses to integrated simulations employing software components from a variety of sources. The whole validation activity is iterative, the results of each step of validation can lead to revisions of the CAPE-OPEN standards. The project is organized to minimize extensive revisions, but not so rigidly that software cannot be written before every detail is defined.

CAPE-OPEN has been developing two complementary standard formats, one built around Microsoft COM technology and the other around the OMG CORBA technology. The validation programme described in this report concentrates exclusively on the COM technology. The standards are, however, designed for easy migration.

2 Software Components.

The following software components have been identified by the Work Packages:

- Simulator executive

- Physical Properties Package

- Physical Properties Routine

- Unit Operation

- Numerical Routines

- Neutral file systems

These components are illustrated in Figure 1.
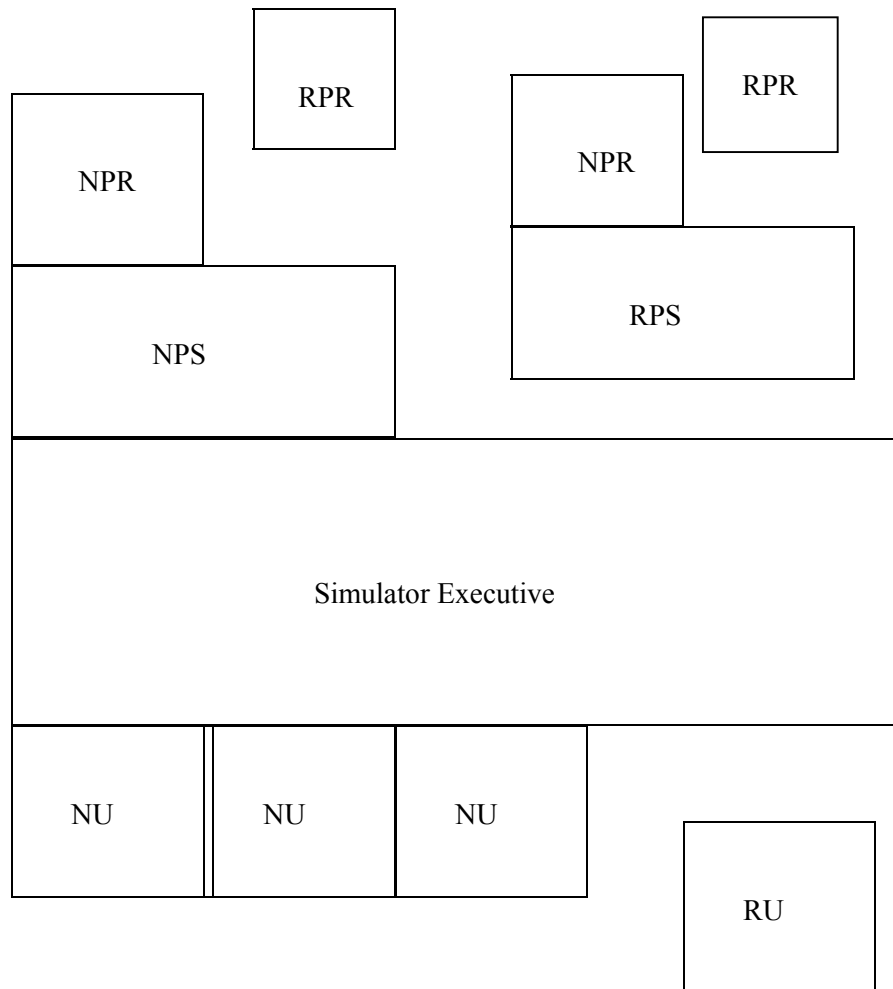
KEY: **NPR - Native Props Routine**   **RPR - Replaceable Props Routine**
         NPS - Native Properties System   **RPS - Replaceable Properties Syst.**
         NU - Native Unit Model   **RU - Replaceable Unit Model**

**Figure 1. Major Components of CAPE-OPEN compliant simulator
(numerical routines not shown)**

A typical commercially available, non-CAPE-OPEN, simulator consists of a Simulator Executive with a library of Unit Models, a Physical Properties System and a library of Physical Properties routines. The Unit models represent such units as reactors, distillation columns, mixers, and splitters. They are linked to the Simulator Executive using interfaces that are proprietary to the simulator supplier. The Physical Properties System either is an integral part of the simulator, or is linked to it by a proprietary interface. The Physical Property Routines represent methods for calculating individual properties, such as viscosity, or sets of properties as, for example, provided by an equation of state. There will be a variety of

alternative routines for calculating each property. The physical properties system provides means for selecting chemical components, routines and data appropriate for a particular simulation. We call this selection a physical properties package.

The Simulator Executive provides means for selecting a set of Unit Models appropriate for a given simulation and specifying the process topology (the logical connexions between the units). The Executive further permits the allocation of property packages to individual units, streams or sections of the flowsheet. The particular options available depend on the particular simulator.

For simplicity, Figure 1 does not show numerical routines. Simulator Executives also provide a library of numerical methods for solving algebraic, or differential and algebraic, equations in order to solve the specified simulations. Most simulators provide default methods for organizing and solving the resulting equations, with the option of alternative methods for the user to select if the speed or precision of the simulation is inadequate. We consider neither numerical routines nor interfaces to neutral file systems further in this document which focuses on validation of those interfaces necessary to support a mixer-splitter example, i.e. an example flowsheet consisting of mixer and splitter unit operations. This example is designed to show that plug&play interoperability for Unit Operations and Physical Properties Systems is achieved.

CAPE-OPEN is defining interfaces that enable the major components shown in Figure 1 to be interchanged. Thus, in addition to the Simulator native unit models, it will be possible to add replaceable CAPE-OPEN unit models. Similarly, it will be possible to add CAPE-OPEN physical properties routines alongside native routines. It will also be possible to add to, or replace, entire physical properties systems. These new physical properties systems will come with their own native libraries of property routines. CAPE-OPEN will allow routines to be added to these systems in the same way as routines could be added to the original native physical property systems.

It is emphasized that the major software components described here do not necessarily correspond to software components as defined by COM or CORBA. Thus, it may be convenient conceptually to break a system (or package) into a number of subsidiary software components, each of which presents a set of logically related interfaces. The structure illustrated in Figure 1 does, however, provide a convenient and meaningful grouping of the interfaces that are standardized in order to achieve the CAPE-OPEN goals of interoperability. These interfaces are described in the following Chapter 4 of this report.

It is further emphasized that it is expected that commercial simulators will continue to provide their own native interfaces alongside the CAPE-OPEN interfaces. In a typical simulation, it is likely that the majority of unit models will be native unit models. Only one or two special units may be added using CAPE-OPEN interfaces. Similarly, it is expected that commercial simulators will provide the great majority of physical property methods and data required by end users. End users may then add only one or two properties using CAPE-OPEN interfaces. Note that it is not expected that commercial simulators will initially offer all CAPE-OPEN interfaces. The sub-set offered may be just unit models, just complete physical properties systems, or just the ability to exchange property routines. It is hoped that numerical method interfaces and interfaces to neutral files systems (e.g. databases) will be added in due course.

3 Standardized Interfaces.

There are essentially three sets of interfaces that are discussed in this document and that need to be validated.

- Interfaces between property routines (RPR) and physical properties systems

- Interfaces between physical properties systems (RPS) and simulator executives

- Interfaces between simulator executives and unit models (RU)

The set of **properties routine** interfaces transmits requests to the replaceable property routines and return property values. Some property routines are capable of returning several properties as a consequence of one computation. To employ such routines efficiently, the interfaces provide options to request several properties at one call. The property routines need not be self-standing. Thus, they may themselves require property values. For example, a thermal conductivity routine may require values for viscosity and heat capacity. Such routines may then themselves generate requests for properties, the values of which are returned by the property systems.

In addition to the set of interfaces to the replaceable routines, **Physical Property Systems** have a set of interfaces for communicating with the simulator executive. The interfaces transmit requests from the executive for particular properties, and return the values required. The requests may also be made for sets of chemical components and for parameters for these components (for example, estimated critical constants).

The **Unit Models** have interfaces for requesting input values (both material flows and non-material information) and receiving the values requested. They also have corresponding interfaces for setting output values of material flows and other information.

4 Validation Strategy.

An iterative approach to validation as outlined in the Conceptual Design Document [1997] has been adopted.

The steps are essentially as follows:

(i) The Users Requirements for CAPE-OPEN are elicited. These have been recorded in the Conceptual Design Document [1997]. No revisions have been issued, but a revised prioritization has been agreed. Thus the first priority is to define and test just those unit and physical properties interfaces needed to support interchange of a mixer-splitter unit and its associated physical properties.

(ii) The User Requirements are elaborated into Use Cases. These Use Cases enable the specific behaviours required of CAPE-OPEN software components to be defined in more detail. These have been reviewed within

the relevant work packages (Numerical, Thermodynamic and Unit Operations).

(iii)   Further steps in the UML methodology are followed, generating Sequence Diagrams, Class Diagrams and Interface Diagrams. All of these UML steps have been validated by the relevant work packages.

(iv)   Interface Specifications. Based on the above activities, prototype standard interfaces have been defined. In the course of these definitions, revisions to some of the earlier steps of the UML process have been made.

(v)   Generation of prototype code. Each work package has generated prototype code. This code has generally been generated by (one or more of) the authors of the interface specifications. It has been an integral part of the activity to ensure that the interfaces are adequately defined. The specifications have been iteratively updated as part of the code generation and, where appropriate, earlier steps in the UML process have also been updated. Prototype code is generally not a deliverable from the project as it may make extensive use of valuable commercial software. Its primary purpose is to refine the interface specification. A second, but very important, purpose is to provide a first demonstration that the interfaces work.

(vi)   Consistency checking. Consistency between the work packages is being achieved by joint reviews and meetings. The first priority has been to establish consistency between the Unit and Physical Property proposals. As a consequence of collaboration between the groups, a number of gaps and inconsistencies have been identified. These have been addressed in a set of new Use Cases which have been jointly developed, and revisions to the individual interface specifications.

(vii)   Maintenance of records. UML is a means to an end. It is not an end in itself. It has, therefore, not been considered to be a high priority to maintain consistency at every step of the documentation. In the end, only that documentation necessary to define and maintain the standard interfaces will be retained. Thus, where relevant documentation has been found to be in error, it has been updated. Redundant documentation has not, however, been rigorously eliminated. The limited resources have been concentrated on the active documentation needed to support the prototype activity. This approach is consistent with that recommended by Fowler and Scott [1997].

(viii)   Preparation of test harnesses. This activity is described in the following chapter. Experience with these harnesses was intended to lead to further revisions of the interface specifications. The test harnesses were planned to be updated as CAPE-OPEN progresses, and as deliverables from the project they are freely available to other workers wishing to produce CAPE-OPEN components.

(ix)   Integration testing: In the case that a CAPE-OPEN-conforming simulator executive prototype is delivered in addition to the one or the other prototype component, tests will be performed using this prototype components with their native interfaces in their original environment as well as with CAPE-OPEN interfaces in the CAPE-OPEN-conforming environment. Comparison

of the results should not reveal major differences, and the loss in performance should be limited.

(x)   Interchange of prototype components. Simulations have been undertaken with prototype code to demonstrate that the standards allow to work together in a "plug and play" manner.

5 Design of Test Harnesses.

Testing CAPE-OPEN interfaces against example processes alone would be a time-consuming activity with results that may be difficult to interpret. Furthermore, although the test schedule is designed to exercise all features of all interfaces, there can be no assurance that all features will be so exercised. An initial test programme has, therefore, been devised that will permit all the interface features to be tested qualitatively. Thus, we will be able to test that the right number of pieces of information is transmitted and that each is organized in the correct way. For example, a set of double length floating point numbers is transmitted across the test interface, in the correct direction, in response to a defined request. This initial test programme envisages the development of a set of test harnesses. These harnesses are intended to detect most logical errors before complete processes are simulated.

The objective of the test components (harnesses) is not to perform chemical engineering computations, but to test that all components and interfaces are unambiguously defined. It is intended that the test suite will mainly pass messages, but where appropriate, will also include some elementary computations.

The initial tests concentrate on the interfaces between the Simulator Executive and a Unit Model and between the Simulator Executive and the Physical Properties System. These two sets of interfaces are sufficient to demonstrate the interchangeability of Mixer Splitter models. Thus, it will be possible after successful testing to simulate a whole process with native unit models and physical property systems. It will be possible to extend this simulation by exchanging physical property systems using CAPE-OPEN standard interfaces. The exchanged physical property system will then be available to all the native unit models. As an alternative to interchanging the physical property system, it will be possible to repeat the simulation with any native mixer-splitter models replaced by CAPE-OPEN models. The native physical property system will then apply to all the native units and also to the CAPE-OPEN mixer splitter units. Finally, it will be possible to interchange both the mixer-splitter models and the physical property system. The interchanged physical property system will then apply both to the native units and to the interchanged CAPE-OPEN mixer-splitter models. At this level of interchange, the main features that were required of CAPE-OPEN will have been achieved and demonstrated. This chapter describes the more elementary steps that will be necessary before these demonstrations of complete process simulations can be achieved.

i) Basic interface tests. These tests interrogate components to ascertain that all the required interfaces are present. The test harness generates a report indicating missing interfaces. Note that, for any particular component, a missing interface may not be an error. For example, it might be impossible for a particular exception

condition to occur. Moreover, note that it is expected that many of the simulator supplier components will have additional interfaces to give direct access to their own native facilities. Such components with additional interfaces are, of course, still fully CAPE-OPEN compliant.

ii) Function tests. Messages will be passed across the interfaces (for example, a request to return a set of property values such as partial molar volumes) and the complementary test component will return appropriate values. The values will probably not be computed, but simply returned from a set of values stored. The test harnesses runs through a large number of such tests. The tests also run through chains of events. For example, a unit model is requested to operate, which in turn requests properties which request a property method. At each step, a check is made that the appropriate type of value is returned. It will be noted that, when applied to actual components rather than test components, there will be no prior knowledge of the correctness of any given property. It is thus practicable to test only the correctness of the type of value and whether it is within sensible bounds (e.g. absolute temperatures and pressures greater than zero). The harnesses also run through tests of the exception conditions. Thus, methods and interfaces used only to respond to or signal exceptions will be exercised. These exceptions will include both logical and numerical exceptions, and exceptions generated when missing interface behaviours are encountered. Also included will be valid range tests. These function tests will include setting unit model parameters from "information streams". Again, these components will be delivered to the simulator suppliers to check that the specifications are consistent with their understanding.

iii) Actual value tests. A very simple physical properties package will be written that returns just the values needed by a mixer-splitter, namely enthalpies and vapour pressures for an ideal mixture. This component will also be delivered to the simulator suppliers and to IFP. It will be necessary to provide such a component because confidentiality will prevent the direct exchange of commercial physical properties systems between suppliers. Similarly, one or more very simple mixer-splitter models will be prepared and supplied. (It is anticipated that one of the prototypes already written will serve this purpose).

iv) Performance tests. It is expected that performance tests will be undertaken as part of the PATH (complementary research) activity. Tests have already been undertaken to compare run times using direct calls, and calls via COM (in-process and out-of-process). Further tests will establish the intrinsic run-time efficiencies of the technologies that we are employing. We can then confirm that we have selected appropriate technologies, and an appropriate level of granularity, to give an acceptably small run-time overhead.

v) Clarity of definition tests. Wherever possible, test components will be written by at least two fully independent authors (usually one at IFP, one at QuantiSci). In this way, it can be verified that the standards have been sufficiently clearly defined that communicating components can be written without collaboration between the authors. The fact that prototypes are already being written by the commercial simulator suppliers goes some way towards meeting this goal. Their test harnesses will be delivered to the validation team.

6 Prototype conformant simulators.

AspenTech and Hyprotech have adapted their simulators to provide prototype CAPE-OPEN conformant interfaces. Within the resource constraints of the project, it was of course impossible that the prototype simulators provide all interfaces. Specifically, the following components have been delivered:

- Physical Properties Systems with CAPE-OPEN interfaces to Simulator Executives. AspenTech (COM), Hyprotech (COM), QuantiSci (Test harness, COM, and simple prototype, COM)

- Simulator Executive with CAPE-OPEN interface to Physical Properties System. AspenTech (COM), Hyprotech (COM)

- Unit Model (steady-state mudular model [group 1 model]) with CAPE-OPEN interface to Simulator Executive. Hyprotech (COM), AspenTech (COM), IFP (model and test harness, COM)

- Simulator Executive with CAPE-OPEN interface to Unit Model. Hyprotech (COM), AspenTech (COM),

7 Description of Test Harnesses.

During the CAPE-OPEN validation activity several software components and harnesses were developed. The purpose of this activity was to test and validate the interface specifications. It was focused on the Unit and Thermo specs. The harnesses were run with validation test components and other components plugged into it in order to identify the ambiguities and completeness of the specifications. All the software is available in source code form. The test environment was decided to be

Microsoft Windows NT4 Service Pack 4

Microsoft Visual Studio C++ 5.0 Service Pack 3

Microsoft Visual Basic 5 Service Pack 3

Apart from the components listed in the previous section, the following CAPE-OPEN Active-X components were produced for validation purposes by the Validation work package partners:

- a MaterialObject component for testing Property Packages was produced. It exposes all the functionality necessary to calculate properties using a CAPE-OPEN Property Package. For this purpose, the MaterialObject component implements the ICapeThermoIdentification and ICapeThermoMaterialObject interfaces together with a private interface to set the component up.

- a dummy Property Package component was developed to support the development of the test environment. This component supports the ICapeIdentification and the ICapeThermoPropertyPackage interfaces.
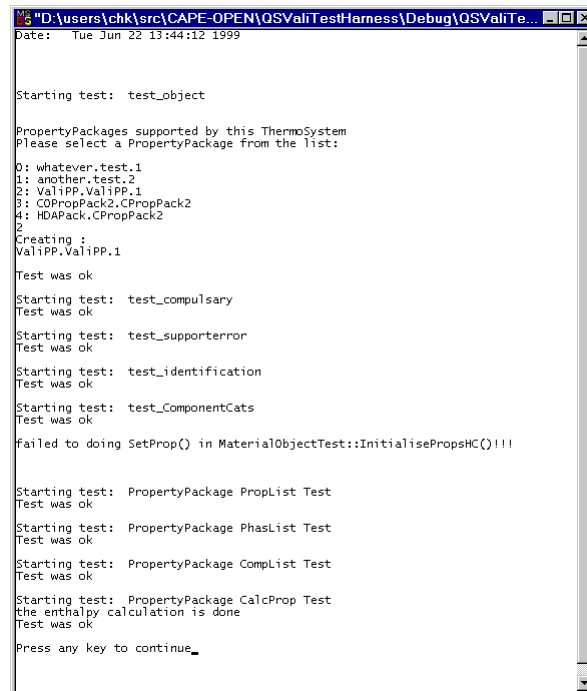
- an EquilibriumServer capable of performing ideal and non ideal TP-Flashes was developed to demonstrate the use of Equilibrium Server components. This Server implements the ICapeThermoIdentification and ICapeThermoEquilibrium interface.

- a CAPE-OPEN Thermo System component which allows the delivery of arbitrary standalone CAPE-OPEN Property Packages was developed to enable the test of standalone Property Packages. It implements the ICapeThermoIdentification and ICapeThermoSystem interface.

- a CAPE-OPEN Unit Operation component. It implements the ICapeIdentification, ICapeUnit, ICapeUnitEdit, and ICapeUnitReport interfaces.

- a CAPE-OPEN Parameter and a CAPE-OPEN Port collection component. Both implement the ICapeIdentification, ICapeUnitCollection interfaces.

- a CAPE-OPEN Parameter and a CAPE-OPEN Port component. Both implement the ICapeIdentification interface and respectively ICapeUnitParameter and ICapeUnitPort.

All these Active-X components were implemented using Microsoft Visual C++ ATL/COM. The source code is available from the CAPE-OPEN interface repository on the QuantiSci Business Collaborator Server at http://cobweb.quantisci.co.uk/bscw/bscw.cgi/0/1284720

7.1 Validation with respect to Thermo

Two Thermo validation programs were developed to test the use of the interface specification in different programming environments. All programs test the existence of the optional and mandatory CAPE-OPEN interfaces on the plugged in components. After this test, different calling sequences of functions on the interfaces are implemented or can be executed to demonstrate the viability of the interface specifications.

7.1.1 C++ console application.

Figure 1: Console output from the C++ validation program

This program consists of different C++ classes which execute methods on the CAPE-OPEN interfaces implemented on a ThermoSystem-PropertyPackage. A standalone PropertyPackage can be plugged into the program utilising the above ThermoSystem component.

Figure 2: ASCII report from the C++ validation program

The output of the test program is written to an ASCII file and can be viewed in any text file editor.

7.1.2 Visual Basic test program CO-PropCalc
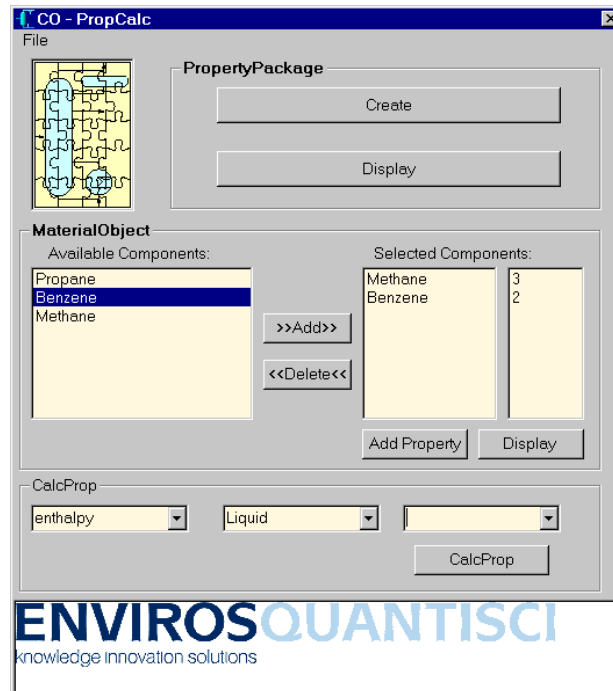
Figure 3: VB application to calculate different properties with a PropertyPackage

This program instantiates a PropertyPackage via a ThermoSystem component specified by its ProgId. The MaterialObject developed within this workpackage can be instantiated and user specified properties can be set through the validation software on this object. The user can select the components in the MaterialObject from the set of components provided by the PropertyPackage.
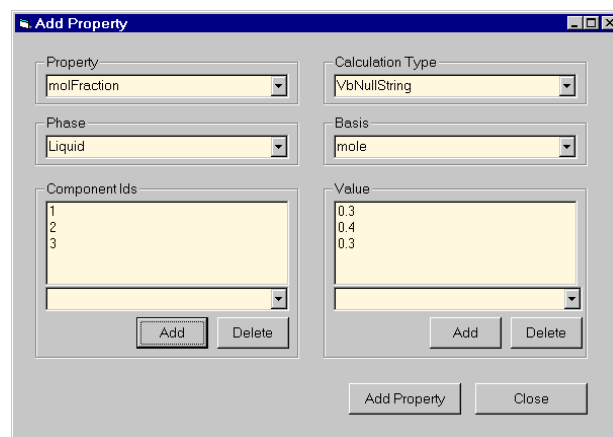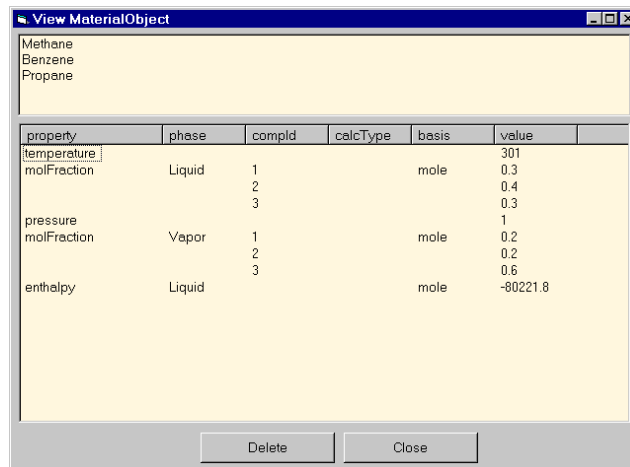

Figure 4: Adding properties to the MaterialObject

This MaterialObject is then passed into the instantiated PropertyPackage to calculate the requested property. The result can be viewed by inspecting the properties of the MaterialObject. after the calculation succeeded.

Figure 5: Displaying the components and properties of the MaterialObject

Both of these validation programs can be downloaded from the interface repository at

http://cobweb.quantisci.co.uk/bscw/bscw.cgi/0/1284720

7.2 Validation with respect to Unit Operations

The major steps performed during the Validation testing of Unit Operations fall into one of the following categories:

(i)   Check that CAPE-OPEN Components are registered under appropriate categories.

(ii) Check each of the components support the documented CAPE-OPEN interfaces according to specifications.

(iii) Check operation of each interface.

7.2.1 Test Harness Architecture

The following subsections describe the test harness architecture for the unit test:

Test Harness Use Case Model
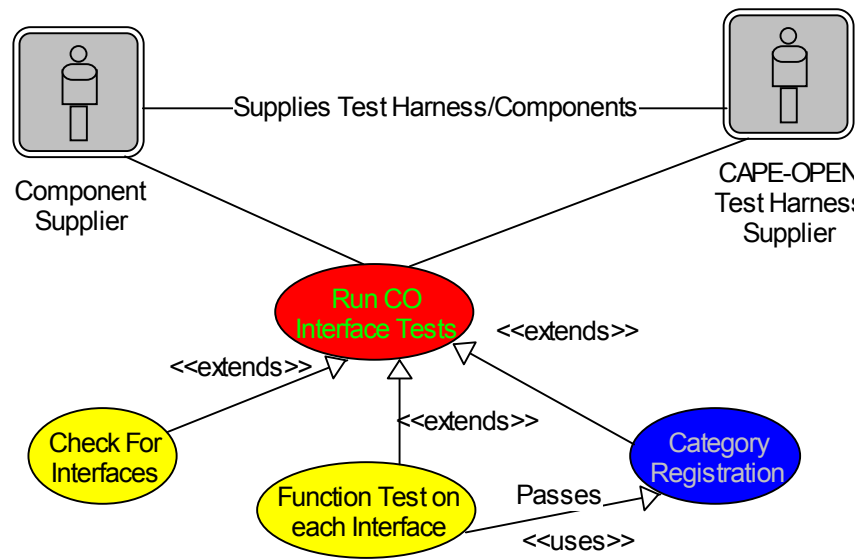
The test harness architecture is comprised of a web browser based test user interface. The test harness utilises HTML, Java Script, VB control, and ActiveX dlls to call various test modules.

Tests are available through a web browser in which we can find all the specifications of the interface definition.

CAPE-OPEN Test Harness Scenarios

Two basic test harness scenarios are followed:

Test 1

- Check if a component is registered correctly in the appropriate category.

- Interface Basic Test

- This tests whether the interfaces (outlined in the component models of the interface specifications) are available.

- Function Test

- Interrogates each of the methods on the interface to see whether it is functioning properly.

Test 2

- Interface Basic Test

- Function Test

- If tests succeed the test harness registers the component under the appropriate category.

Test Harness Component Diagram

The following component diagram describes the architectural approach of the CAPE-OPEN test harnesses. These test harnesses are intended to provide an easy-to-use approach for testing and validating CAPE-OPEN components through the web. It is expected that new test modules will be provided as the CAPE-OPEN interfaces mature. There will have to be a plan to maintain the consistency of the test harnesses, as the interfaces become more commercially available.

CAPE-OPEN
Unit/Thermo/Numr
Test Harness

Web Browser/
HTML / Java
Script

CAPE-OPEN
Test Module
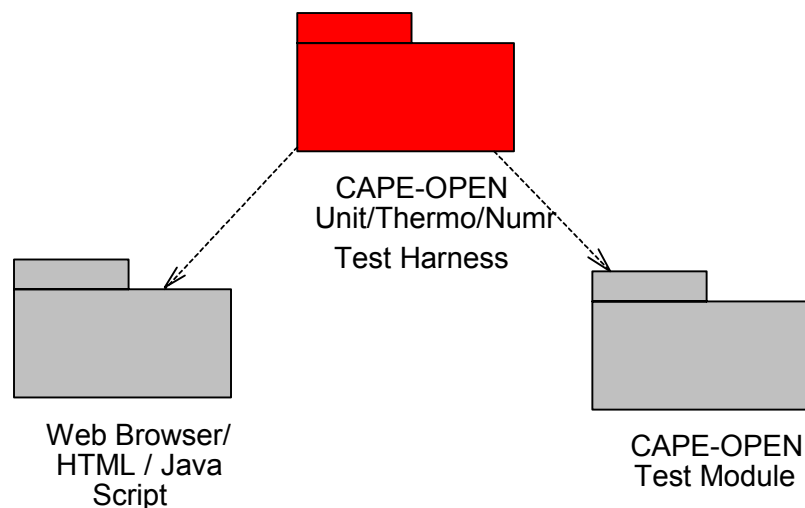
Key Functionality of Test Harness

- Test for Consistent Category Registration Approach

- Tests are built upon Cross Component Collaboration of the Use Case Model

- Components support CAPE-OPEN defined interfaces.

- Textual Reports are run on each of the documented tests outlining where key problems exist.

7.2.2 Specific Tests for the Feasibility Demo on COM Interoperability

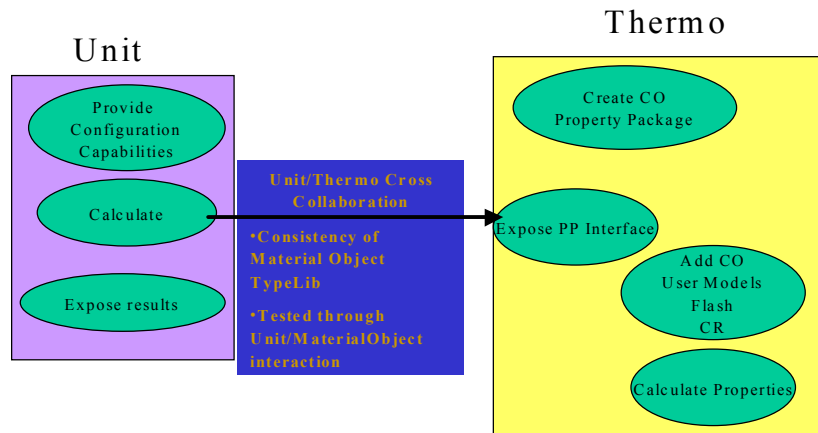### Use Cases Collaboration Diagram

The use case model listed below was agreed as the high level functional use case model. Use Case Collaboration diagrams are a UML stereotype model developed by Michael White as means to explicitly map the use cases to practical component tests. The Use Case Model below provides a simplified view of the key functionality of the CAPE-OPEN interfaces and components. The most important aspects of the test harnesses are described through more detailed use case collaboration diagrams.

Cross Collaboration Use Case Model Overview
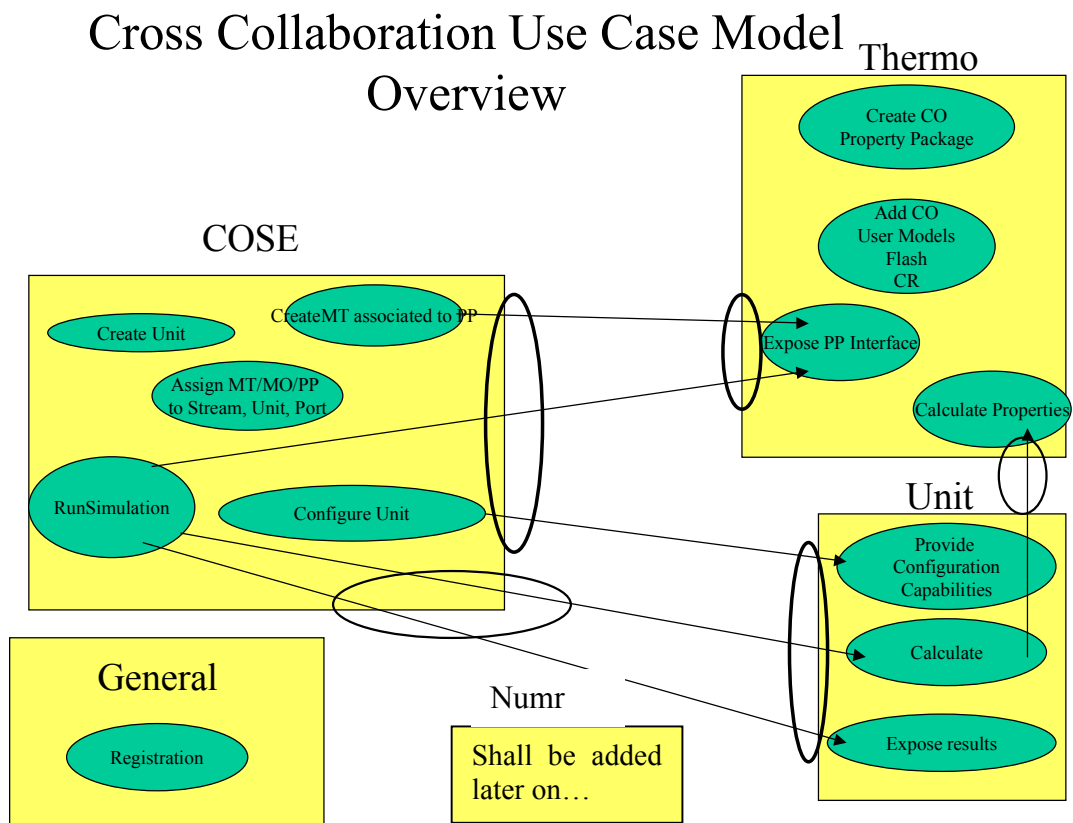
Unit & Thermo Collaboration Diagram

The following diagram describes key functional collaborations between the Unit component and the Thermo component. These collaborations are supported through Material Object and Unit collaboration and are not part of the test harnesses. Any simulator vendor who is providing a COSE component will be required to submit his Material object typeLib for testing by validation.

Unit/Thermo Cross Collaboration Use Case Model

COSE/Unit Collaboration Diagram

The test harnesses will test that Unit Operations can invoke the appropriate interfaces for configuring, calculating, and reporting aspects of a unit operation. The test harnesses will test the use cases by using the ICapeUnit, ICapeUnitReport and ICapeUnitEdit interfaces.

COSE/Unit Cross Collaboration Use Case Model
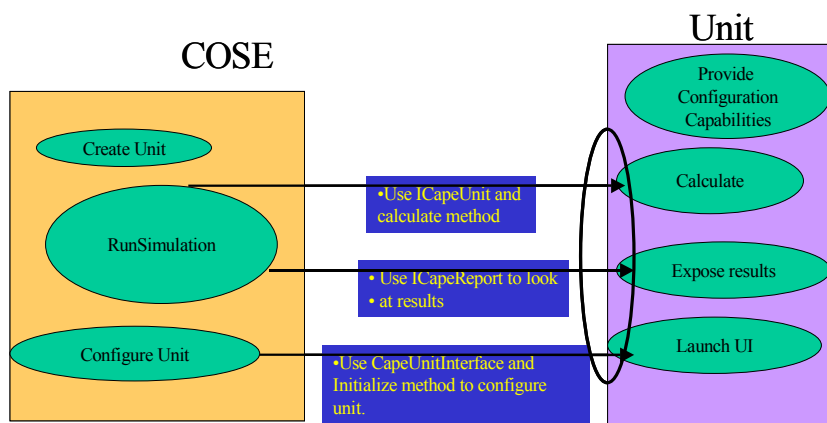
7.2.3 Basic Test Harness Sequence Diagram

The attached diagram outlines the way in which the test harnesses are checking for specified general CAPE-OPEN, Thermo, and Unit CAPE-OPEN interfaces.
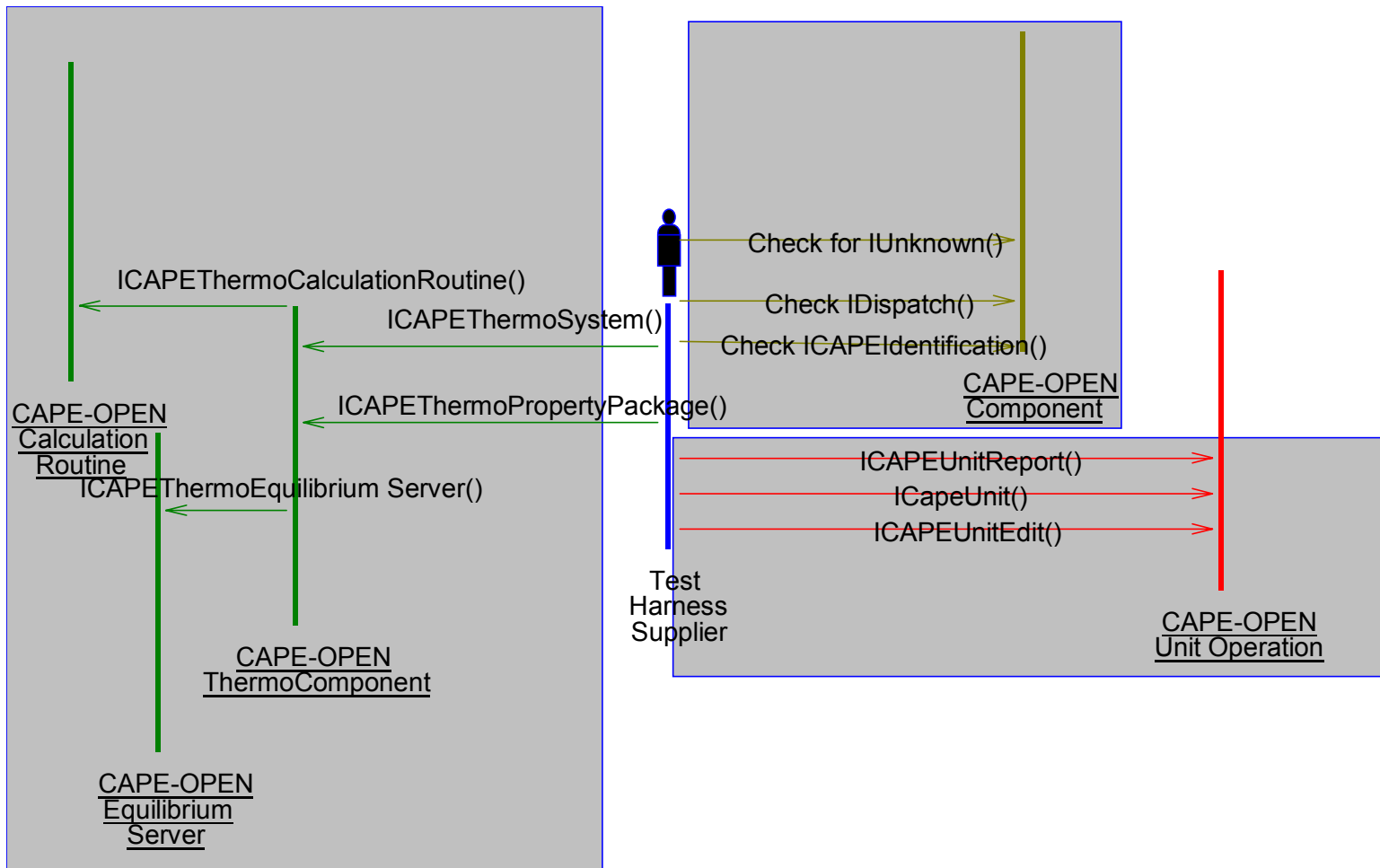
7.2.4 Integration tests

Once this last tests succeeded, we are, theoretically, able to plug different CAPE-OPEN components together.
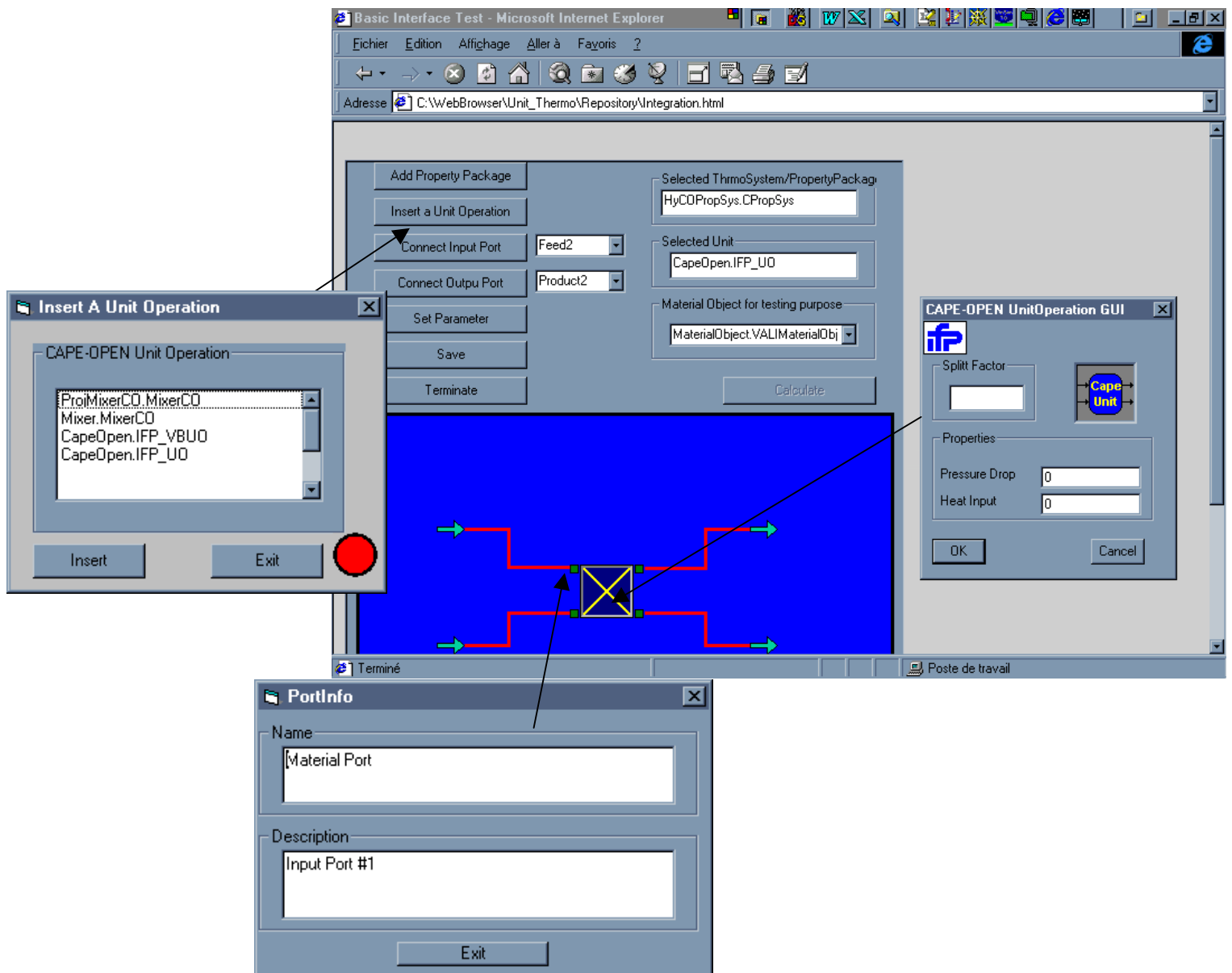
An Integration test has been designed in order to test the collaboration of components provided by different vendor.

This test acts as a CAPE-OPEN Simulator Executive, as shown below.

8 Conclusions.

This document describes the concept for a comprehensive validation activity in CAPE-OPEN. It concentrates on the priorities for meeting the needs to prove the goals of CAPE-OPEN by means of a mixer-splitter example problem. The same approach is, however, being adopted for more complex scenarios and the other components and interfaces.

In the second part, the document describes those tests which have in fact been performed out of the list described above. In the appendix, further validation tests which could be performed are contained as well as a detailed description of the output delivered by the test harnesses.

A1 Appendix: Further validation tests.

The validation programme described in the main text concentrates on the minimum set of tests to ensure that the standards have been defined to the extent that simulation components can be successfully interchanged. Before final acceptance as fully proven standards, a number of further validations will be required. Some of the further required tests are:

i) Logical consistency. For some components, very simple simulation environments will be built. For example, a simple simulation with one unit model, optionally using thermodynamics, including one recycle with equation solution. These environments enable us to make simple convergence tests.   Such tests may reveal some of the logical faults that might be included in specifying object interfaces (for example, either insufficient, or redundant, information required by a solver passed through a unit or thermodynamic interface).   These tests will be restricted to small test harnesses in which is easier to track faults than in large fully functional systems. (This test is of sufficiently high priority for it to be attempted within the CAPE-OPEN time frame if possible).

ii) Independence of compilers. If judged appropriate, software components will be produced in independent languages (e.g. C++ and Visual Basic). Independent compilers (e.g. Microsoft and Borland) should also be considered to ensure that the standard is fully portable and does not rely on an idiosyncrasy of a particular compiler.

iii) Independence of hardware. Where appropriate, each component should be tested on independent hardware. Thus COM is claimed to be fully available for Intel processors, Power PC RISC processors, and DEC Alpha RISC processors. CORBA is claimed to be available on a much wider range of processors. Tests should include remote tests with, where appropriate, components running on different processors across a network.

iv) Independence of operating systems. It is a desire of CAPE-OPEN that the standards are available at least on MSWindows, UNIX and Open VMS. These tests may be restricted to the CORBA implementation.

v) Independence of middleware. Where there are different sources for the middleware, the components will be tested as far as possible on the different options available. For example, CORBA is available from a number of independent suppliers.

vi) Communication between middleware. There is a neutral standard for communication between CORBA implementations, and a standard for communicating between COM and CORBA. Communication between components compiled in different flavours of CORBA, and between CORBA and COM should be tested.

A2 Appendix: Screenshots of Test Harness User Interface.

CAPE-OPEN Registration Tests
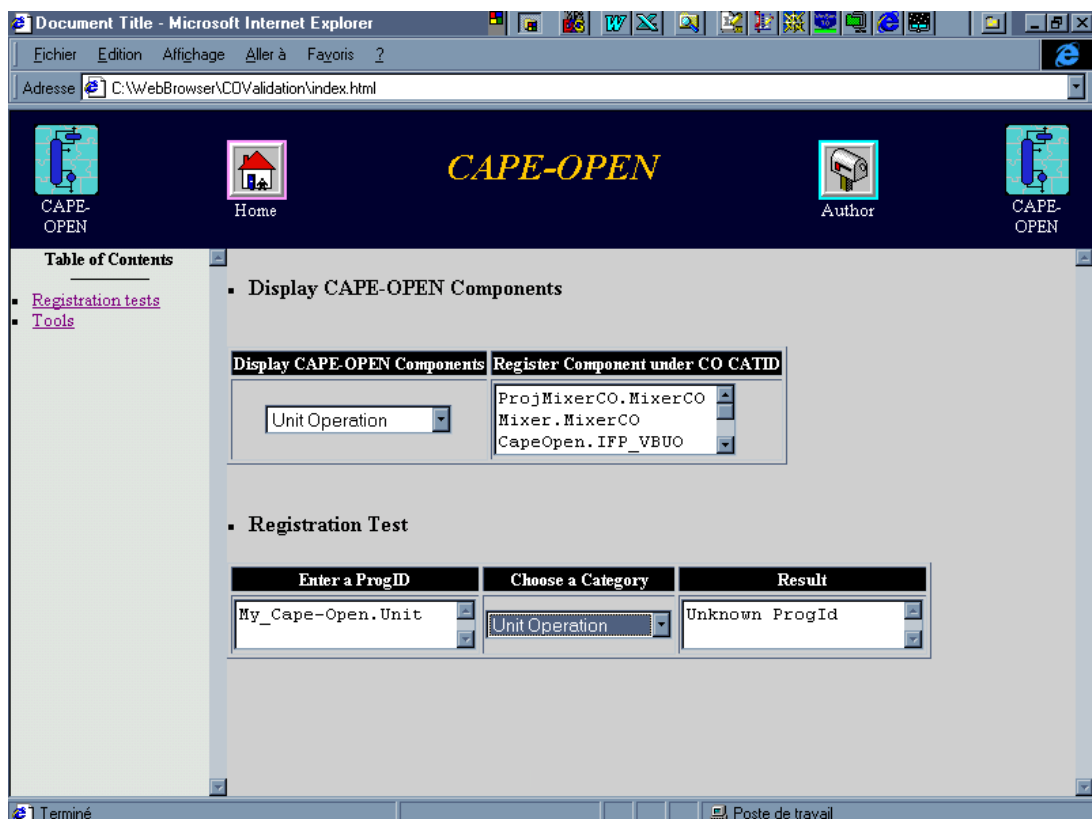
CAPE-OPEN Basic Test

Document Title - Microsoft Internet Explorer

Fichier   Edition   Affichage

Adresse C:\COValidation\

CAPE-OPEN

1. Unit
2. Port
3. Collection

- **Interface Description**

1. CAPE-OPEN Guids
2. IDL despciption

- **Unit Use Cases**

1. Actors
2. Creating a flowsheet
3. Runnig a flowsheet
4. Looking at result

- **Validation & Tests**

1. Basic Interface Tests
2. Function Tests
3. Test Cases

Steady State Mixer/Splitter Model

Specifications

Split Factors

port ID
material ID
split factors

Properties

Pressure Drop (kPa)

Heat Input (kJ/h)

Exit

Specifications   Worksheet

Author

CAPE-OPEN

Operation

Operation

CO.MixerCO

THERM ProgID	COMaterialObject.CMate
Results (ALL)	Not Performed
Results (RESTRICTED)	Not Performed

Result:

| Components | UnitOperation | PortCollection | ParameterCollection | Port | Parameter |

Interfaces   Th.   Prt.   Th.   Prt.   Th.   Prt.   Th.   Prt.   Th.   Prt.

Poste de travail

Document Title - Microsoft Internet Explorer

Fichier   Edition   Affichage   Aller à   Favoris   ?

Adresse C:\COValidation\index.html

CAPE-OPEN

Home

CAPE-OPEN

Author

CAPE-OPEN

1. Unit
2. Port
3. Collection

- **Interface Description**

1. CAPE-OPEN Guids
2. IDL despciption

- **Unit Use Cases**

1. Actors
2. Creating a flowsheet
3. Runnig a flowsheet
4. Looking at result

- **Validation & Tests**

1. Basic Interface Tests
2. Function Tests
3. Test Cases

Components	UnitOperation		PortCollection		ParameterCollection		Port		Parameter	
Interfaces	Th.	Prt.	Th.	Prt.	Th.	Prt.	Th.	Prt.	Th.	Prt.
IUnknown	×	YES	×	YES	×	YES	×	YES	×	YES
IDispatch	×	YES	×	YES	×	YES	×	YES	×	YES
ICapeIdentification	×	YES	×	YES	×	YES	×	YES	×	YES
ICapeUnit	×	YES								
ICapeUnitEdit	×	YES								
ICapeUnitReport	×	*NO*								
ICapeUnitCollection			×	YES	×	YES				
ICapeUnitPort							×	YES		

Poste de travail

A3 Appendix: Report generated through Basic Test.

```
*****************************************************************
********
                BASIC INTERFACE TEST

*****************************************************************
********
Tested Component:
UnitOperation

Description:
  Test will interrogate interfaces to a given component to
  ascertain that all the required interfaces are present.

Author:
  Pascal Roux

Date and Time:
  Tue Mar 23 18:18:51 1999
*****************************************************************
********
CoCreateIntance succed



IUnknown is supported

  IDispatch is supported
  ICapeIdentification is supported
  ICapeUnit is supported
  ICapeUnitEdit is supported
  ICapeUnitReport is supported
*****************************************************************
********
                BASIC INTERFACE TEST

*****************************************************************
********
Tested Component:
PortCollection

Description:
  Test will interrogate interfaces to a given component to
  ascertain that all the required interfaces are present.

Author:
  Pascal Roux

Date and Time:
  Tue Mar 23 18:18:52 1999
```

```
*****************************************************************
********
CoCreateIntance succed



IUnknown is supported

  IDispatch is supported
ICapeIdentification is supported

  ICapeUnitCollection is supported
*****************************************************************
********
                 BASIC INTERFACE TEST


*****************************************************************
********
Tested Component:
ParameterCollection

Description:
  Test will interrogate interfaces to a given component to
  ascertain that all the required interfaces are present.

Author:
  Pascal Roux

Date and Time:
  Tue Mar 23 18:18:52 1999
*****************************************************************
********
CoCreateIntance succed



IUnknown is supported

  IDispatch is supported
  ICapeIdentification is supported

  ICapeUnitCollection is supported


*****************************************************************
********
                 BASIC INTERFACE TEST


*****************************************************************
********
Tested Component:
Port
```

Description:
  Test will interrogate interfaces to a given component to
  ascertain that all the required interfaces are present.

Author:
  Pascal Roux

Date and Time:
  Tue Mar 23 18:18:53 1999
****
****
CoCreateIntance succed

IUnknown is supported

  IDispatch is supported
ICapeIdentification is supported

  ICapeUnitCollection is supported

****
****
                    BASIC INTERFACE TEST

****
****
Tested Component:
Port

Description:
  Test will interrogate interfaces to a given component to
  ascertain that all the required interfaces are present.

Author:
  Pascal Roux

Date and Time:
  Tue Mar 23 18:18:53 1999
****
****
CoCreateIntance succed

IUnknown is supported

  IDispatch is supported
  ICapeIdentification is supported
  ICapeUnitParameter is supported

A4 Appendix: Report generated through Function Test.

```
********************************************************************
********
                    FUNCTION INTERFACE TEST


********************************************************************
********
Tested Component:
  FunctionTest

Description:
  Test will interrogate interfaces to a given component to
  and test some of the methods and properties.

Author:
  Pascal Roux

Date and Time:
  Tue Mar 23 18:23:13 1999
********************************************************************
********
CoCreateIntance succeded

Get a pointer to IUnknown
  IUnknown is supported

*Checking for the specified interfaces:
  IDispatch is supported
  ICapeIdentification is supported
  ICapeUnit is supported
  ICapeUnitEdit is supported
  Unit has it own GUI
                        ICapeUnitReport is not supported


----------------------------------------------------------------------
=> Invoke ICapeUnit::Initialize() method

ICapeUnit Initialize suceeded
----------------------------------------------------------------------
=> Invoke ICapeUnit::get_Ports() property
  This method shoul return a Pointeur to IDispatch to a CapePortCollection.
  So, with this pointeur, we can ensure that all specified interface are available.
  Get a pointer to IDispatch of UnitPortCollection
  Get a pointer to IUnknown of ICapeUnitPortCollection too
  IUnknown is supported

*Checking for the specified interfaces
  IDispatch is supported
ICapeIdentification is supported
```

ICapeUnitCollection is supported

=> Invoke ICapeUnit::get_Ports() property
  This method shoul return a Pointeur to IDispatch to a CapeParameterCollection.
  So, with this pointeur, we can ensure that all specified interface are available.
Public parameters collection available.
  Get a pointer to IDispatch of ICapeUnitPortCollection
  Get a pointer to IUnknown of ICapeUnitPortCollection too
  IUnknown is supported

*Checking for the specified interfaces
  IDispatch is supported
  ICapeIdentification is supported

  ICapeUnitCollection is supported

=> Invoke ICapeUnitCollection::Count() and Item() Methods
  The item method shoul return a Pointeur to IDispatch to a CapeUnitPort.
  So, with this pointeur, we can ensure that all specified interface are available.
  Get a pointer to IDispatch of Port
  Get a pointer to IUnknown of Port too
  IUnknown is supported

*Checking for the specified interfaces
  IDispatch is supported
ICapeIdentification is supported

  ICapeUnitCollection is supported

=> Invoke ICapeUnitCollection::Count() and Item() Methods
  The item method shoul return a Pointeur to IDispatch to a CapeUnitParameter.
  So, with this pointeur, we can ensure that all specified interface are available.
=> Invoke ICapeUnitCollection::Count() ,count
  Get a pointer to IDispatch of Parameter
  Get a pointer to IUnknown of Parameter too
  IUnknown is supported

*Checking for the specified interfaces
  IDispatch is supported
  ICapeIdentification is supported
  ICapeUnitParameter is supported

Cannot connect Nothing !
=>Invoke ICapeParameter::put_Value method succeeded
=>Invoke ICapeParameter::Valid() method failed
=>Invoke ICapeUnit::Valid method
The Unit Operation is not able to calculate
****
*

Test Stop, Release all the varialbles