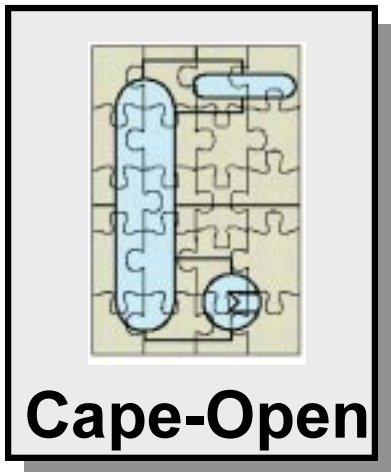


CAPE PATH Recommendations

A Window to the Future



**J. Köller
L. von Wedel
M. Jarke
W. Marquardt**

Januar 2000

Archival Information

Reference	
Authors	Jörg Köller Lars von Wedel
Date	
Number of Pages	2
Version	Version 1 (Draft 1)
Filename, short	
Filename, long	
Location: Electronic Hardcopy 1 Hardcopy 2	
Reviewed by 4/March/1997 4/March/1997 7/March/1997 7/March/1997	
Approved by	
Distribution	CO
Originating Organisations Reference Number	

Summary

The document describes the work of the PATH workpackage of the EC-funded project CAPE-OPEN. The objective is to explore conceptual and technical research issues related to process simulation and the use of software components. The prototypical simulation environment Cheops has been built to demonstrate the ideas. It will be described in this document. Integration with various CAPE-OPEN compliant prototypes will be described. Some remarks about the CAPE-OPEN interface definitions will be made. Finally, CORBA, as the implementation platform of Cheops, will be presented and discussed.

Contents

1.....	Introduction	5
2.....	Contribution to the CAPE-OPEN Interface Standardization Process	6
3.....	The Prototypical Simulator CHEOPS	7
3.1.....	A Conceptual Object Model for Process Simulation	7
3.1.1.	The Numerics Package	8
3.1.2.	Unit Operations	8
3.1.3.	The Thermodynamics Package	9
3.1.4.	The Executive Package	10
3.1.5.	Deriving Interface Definitions	10
3.2.....	Achievements and Outlook for CHEOPS	11
4.....	Components with CORBA	13
4.1.....	Implementing CORBA Interfaces	14
4.1.1.	The Invocation Mechanism	15
4.2.....	CORBA Services	16
4.2.1.	The Naming Service	16
4.2.2.	The Trader Service	17
Bibliography		18

1. Introduction

The report describes the results of the PATH workpackage within CAPE-OPEN. The objectives of PATH was the assessment of conceptual and technical issues as described in the proposal document.

Chapter 2 gives an overview of the contributions and recommendations made to the interface standardization process with respect to technical and conceptual issues. These issues have been explored during the work in the PATH work package.

A proposal about the interface definitions of an object-oriented thermodynamics package was made in CO-DSGN-RW-04, added as Appendix A. It shows how a conceptual object model as proposed in (Jouliá, Jourda, 1997) can be mapped to an appropriate set of interface definitions. Further, the use of a variable concept as a means of transparent data transfer within an architecture has been emphasized in this document. The document CO-DSGN-RW-03 (Appendix B) made an early contribution to the design of interfaces between different parts of a simulator.

In order to explore these ideas not only in theory, but in a more practical manner, the prototypical simulation environment CHEOPS was built (chapter 3). It is described in (von Wedel, 1997) and (von Wedel, Marquardt, 1999a,b). It is based on a conceptual object model that is subject of chapter 2.1. Further development tasks such as defining component interfaces, or developing data exchange formats, have been drastically simplified by this underlying conceptual model.

A major goal during the design of this environment was to explore how component software can contribute to the flexibility of tomorrow's simulation environments. A possible approach to further enhance the performance of simulation is the integration of different, specialized solvers for different parts of the overall problem to solve. This approach is an important means to take advantage of knowledge about the problem structure because solvers can be tailored to individual parts of a problem. This knowledge cannot be exploited by general-purpose solvers such as a Newton-based algorithm. More detailed examples can be found e.g. in (Biegler, 1999). The environment developed aims at the solution of problems in this area.

An advanced design of unit operation modules for an integration of different simulation approaches such as modular and equation-oriented has been proposed in (von Wedel, Marquardt, 1999b) and will also be described in this document.

The Cheops environment has been used to integrate and validate CAPE-OPEN-compliant prototypes on numerical solvers and equations set objects (in close collaboration with Imperial College, London), the IK-Cape thermodynamics package, and the SMST prototype (in close collaboration with INPT, Toulouse). This environment has been presented at the ESCAPE-9 conference (Budapest) and at the final meeting in Paris.

Finally, the chapter 4 presents CORBA as a middleware platform in general.

2. Contribution to the CAPE-OPEN Interface Standardization Process

This section will provide a brief review on the work of the mainstream CAPE-OPEN project from the perspective of PATH. It is not meant as a general criticism but rather describes some experiences that could be extracted from the work done in PATH.

The main issue concerns the overall reference architecture of a CAPE-OPEN compliant simulation environment. Because the CAPE-OPEN standardization process was strongly focused on the structure of existing tools, the flexibility of the resulting interface specifications was not fully exploited. As an example, the integration of modular and equation-oriented models can only be achieved by a holistic approach, not by processes based upon existing tools that do not provide this integration either. As a consequence, it is not possible to use CAPE-OPEN unit operation modules or thermodynamic packages standalone. They always have to be connected to a CAPE-OPEN compliant simulator executive, thereby reducing the openness of the approach.

This could have been overcome by agreeing on a conceptual object model at the beginning of the project, before starting the specification of interfaces. This object model should describe the basic entities participating in a simulation, their relationships as well as the most important attributes. Further, a basic interaction model should have been agreed upon. Such a process would have revealed concepts that can be reused across different work packages.

A good example for this situation is the lack of a generally agreed mechanism for exchanging data, e.g. process quantities, among components. A possible solution is the interface for a variable as defined in Cheops. Simpler solutions may be used, e.g. a standardized set of method definitions for setting and retrieving data from a component. Changes and extensions concerning the representation of variables have to be made in several interfaces. Instead, factoring out this issue into a common interface would have simplified extensibility and maintenance of the standard. To be more concrete, exchange of process quantities is defined 4 times in CAPE-OPEN, for the ICapeNumericAlgebraicESO, for the ICapeThermoMaterialObject, for the ICapeThermoPropertyPackage, for the ICapeParameter interface.

An object model as defined above would also have simplified related tasks such as the definition of a simulation database interface (and its schema), or a neutral CAPE exchange language as proposed for Global CAPE-OPEN.

A further important fact about the standard definition is that it lacks mechanisms for verification during the runtime phase. As an example, it is not possible for the simulator to check whether he can supply all types of values that a certain unit operation module requires. Errors regarding the data exchange occur during the simulation but they should occur during the setup phase.

We would recommend to iterate on these suggestions in the next evolution phase of the CAPE-OPEN interface definitions.

3. The Prototypical Simulator CHEOPS

Cheops has a component architecture which provides a lot of flexibility on different levels. On the unit level, the structure of a unit describing the different in- and outlet ports is separated from the behavior which actually defines how the unit model equations can be evaluated. Likewise, the structure on the flowsheet level, represented by a coupling, is separated from the actual solution of the flowsheet. The use of componentware enables the substitution of individual parts during runtime such as the solution strategy on the flowsheet or on the unit level.

Consequently, no monolithic simulator executive exists as a piece of software on its own, instead a simulator lives as a set of components on a network. In our vision the user is not only specifying a flowsheet but merely selecting all components participating in the simulation run. Setting up the simulation becomes more or less a process of putting together a new simulator from a set of components. The flexibility of this approach is of course accompanied by a set of difficulties: components are no longer selected from a fixed library but an open, virtual library from a network instead. The set-up of such a completely component-based environment must be supported by a tool to ensure that components are compatible with each other and correctly configured. As there is no executive blob that could take over this functionality different approaches to this problem have to be exploited. Today's middleware also does not offer mechanisms which support these tasks. For example, from a component's interface description one cannot determine whether any associations with other objects must be made before the component can calculate anything. This is due to a lack of encapsulation of the object oriented paradigm: Though objects are well encapsulated with regards to their servers, there is no mechanism that encapsulates the internal details from an object's environment. References made to the 'outside world' are not formally stated in an object's context. Support of the component set-up and configuration must be extended.

3.1. A Conceptual Object Model for Process Simulation

An important step during the development of Cheops was the conceptual definition of an implementation independent object model as the basis for further steps. It defines entities related to process simulation and their relationships in a conceptual manner. It can be used for different purposes:

- Interfaces and their relationships can be derived from the conceptual object model
- A general implementation model in the sense of a framework can be derived from the conceptual object model in order to speed up the implementation of e.g. unit operations where many implementations are required (as opposed to e.g. a solver)
- A model exchange format can be derived from the object model because it specifies attributes and which concepts are related among each other.

The remainder of this section describes the underlying object model of Cheops. It is structured in 4 packages. The numerics package defines a variable that is a basic entity in the overall model. It is used to describe any physical quantity occurring in the overall object model and can therefore be reused in many different situation as

can be seen in figure 1. The thermodynamics package defines a set of thermodynamic concepts like substances and phase systems. Their quantitative properties are described by variables. The unit package defines a unit operation and its ports which denote the interaction with other unit operations. Finally, the executive package defines concepts to represent the flowsheet topology and solvers working on the flowsheet level.

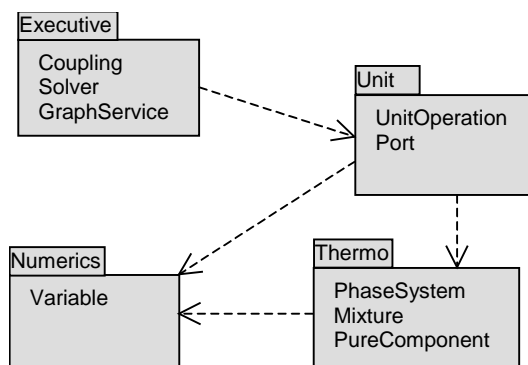


Figure 1: Overview of the Cheops Object Model

3.1.1. The Numerics Package

The numerics package introduces the `Variable` which abstracts all meaningful physical and thermodynamic quantities participating in a simulation. The ports for example expose their values as process quantities. Design degrees of freedom for the unit are also exposed as variables. Variables are a convenient means to group together different aspects of a variable such as their values, units, mathematical and physical dimension, or estimates.

Handling and performance of those fine-grained components is of course a crucial issue in process simulation. The experiences show that the additional amount of work for the programmer is quite high, whereas the performance problems can be reduced by intelligent design of various components: if the references to a process quantity components are retrieved only once and stored for the remainder of the simulation run it doesn't make a difference whether the unit or a variable is queried for a set of values.

During the development of the workbench the concept of a variable has been identified as a key concept for interoperability. It is a basic element that occurs over and over again for the quantitative description of materials, ports, unit operations etc. If such a component is used over and over again integration is facilitated through a common understanding. Solvers do no longer have to take care whether they are solving an equation system, a unit operation, or just some thermodynamic calculations.

3.1.2. Unit Operations

The core element in the Cheops architecture is the *unit operation* module. It represents a simulation model for a certain piece of equipment. On the general level of a unit operation, we do not distinguish between different formulations of the underlying model equations, e.g. a modular or an equation-oriented formulation.

A unit operation has an associated set of *ports* (cf. Fig. 2) that denote locations where a unit can be connected to other unit operations. More concrete, the port groups a set of variables that describe the exchange of material, energy, or information in a quantitative manner. Unit operation modules also have variables on their own to quantify equipment parameters and further information being computed by the module.

A variable represents a process quantity such as temperature, energy flow, or volume. It has properties like a value, an estimate, lower and upper boundaries, and a dimension. The variable interface allows arbitrary dimensions of vectors, matrices, etc.

Further, a material port as well as a unit operation are associated with a phase system representing the material flowing through the port or contained in the unit. This will be further discussed in the section about the thermodynamic package.

What we have just explored can be considered as the structure of a unit operation module. It is important that the unit operation module in this form is generally independent of a certain simulation approach.

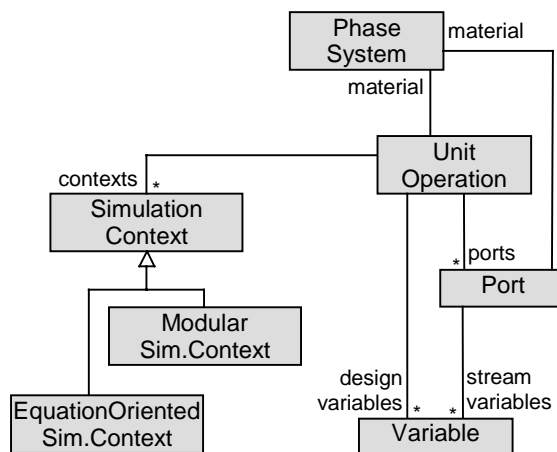


Figure 2: Conceptual Object Model for Unit Operations

In order to perform calculations within a simulation, we introduce a so-called simulation context that is associated to a unit operation. It implements the model equations in different representations. A modular simulation context for example provides a compute method to launch the calculation of output values if the input values of the unit are provided. This context may be employed for sequential and simultaneous simulation on a modular basis. On the other hand, an equation-oriented simulation context can evaluate the model equations to yield a set of residuals and derivatives, e.g. through an ESO interface. A formulation that is more in line with the Cheops architecture would be to use two variables, one denoting the vector of residuals, the other a matrix of derivatives.

3.1.3. The Thermodynamics Package

In order to represent the materials that are processed in a unit operation we define two different flavors of a material concept. The mixture and the pure component represent abstract material properties. A phase system on the other hand denotes the

concrete occurrence of a chunk of material in the plant and can be associated with a thermodynamic state.

Consequently, the mixture and pure component (with an obvious aggregation relationship) can be defined by specifying material properties such as molar weight or critical temperature. They are all described by a variable. Further, thermodynamic laws like mixing rules or an equation of state have to be defined for mixtures and pure components. A phase system will refer to a mixture (or a pure component) and define a context for a thermodynamic state.

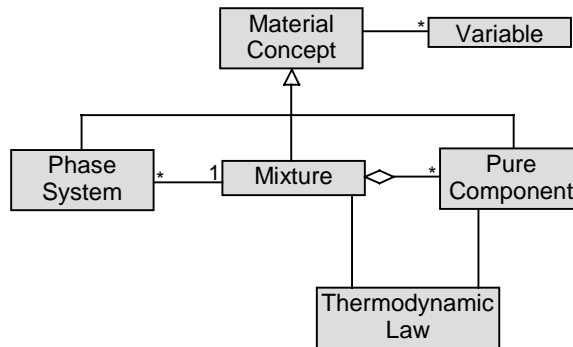


Figure 3: Conceptual Object Model for Thermodynamics

3.1.4. The Executive Package

The executive package deals with the flowsheet representation and its solution. In order to represent the flowsheet topology a coupling object is defined that denotes the coupling relationship that holds between two ports (Fig. 4). A solver object is related to a set of units and couplings which it is able to solve. In this case, the solver can be considered as an application of the strategy design pattern which encapsulates an algorithm that shall be easily substitutable.

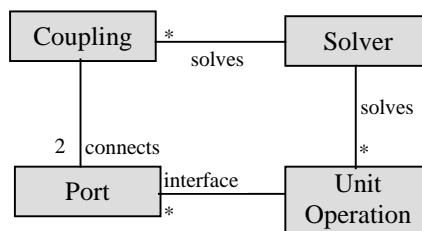


Figure 4: Conceptual Model of the Executive Package

3.1.5. Deriving Interface Definitions

The next step during the design of Cheops concerned the actual definition of interfaces according to the object model defined above. During this step, the packages defined above have been mapped to IDL modules in order to reflect the conceptual ideas in the design of the system. Objects have been equipped with interfaces for which methods had to be defined.

Part of those methods provide on the one hand functionality to navigate through the object graph. For example, a unit can be queried for its port, and each port can be queried for variables. This functionality is realized according to a simple pattern in order to simplify this task as much as possible.

Further methods concern the actual execution of a simulation such as methods to compute unit operation modules, to start the solver, or to propagate variables along a coupling. In order to verify the methods defined, sequence diagrams showing important object interaction have been developed. The diagram below for example shows the execution of a sequential-modular simulation on the flowsheet level: the solver is started via its solve method. It then analyses the flowsheet topology using the graph service. Then, all units and couplings are initialized for the simulation. Finally, the compute methods of the units is called according to the computational order of the flowsheet. Couplings are converged or propagated, depending on whether they have been torn or not.

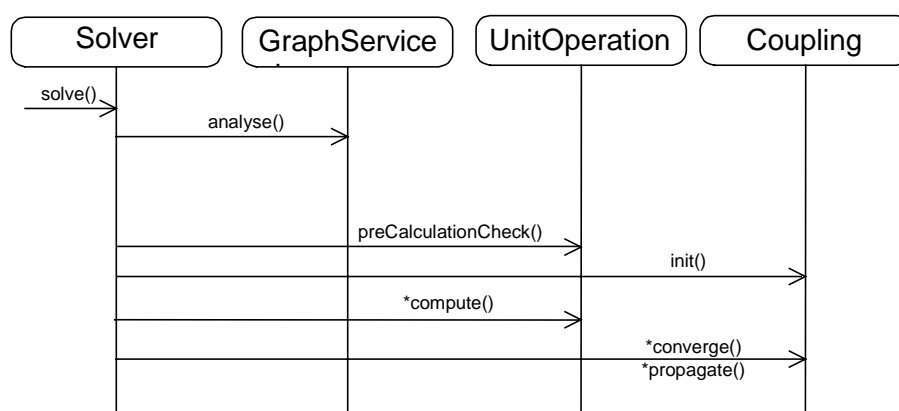


Figure 5: Execution of simulation on the flowsheet level (Sequence diagram)

For the actual writing of the CORBA IDL a set of naming convention and guidelines have been used.

3.2. Achievements and Outlook for CHEOPS

A fully component-based simulator has been developed. Modular simulation strategies simplify the use of legacy code and permit the integration of different simulation mechanisms. Many problems must be addressed using solvers which are highly optimized for a certain problem structure (e.g. adaptive methods, multi-grid solvers etc.). It is questionable, whether a one-fits-all-approach is feasible facing the increasing complexity of simulation tasks in chemical process engineering. Therefore, a flexible handling of simulation strategies is definitely a worthwhile feature to pursue in future research tasks.

In order to emphasize the feasibility of the approach, various existing pieces of software have been integrated in the Cheops environment. A set of FORTRAN-based unit operation modules for catalytic reforming have been wrapped. The SMST tool (INP Toulouse) has been integrated to calculate the computational order for sequential solving. The IK-Cape package has been wrapped with CAPE-OPEN interfaces to supply thermodynamic calculations. Finally, gPROMS has been integrated via the ICapeNumericAlgebraicESO interface. In this case, gPROMS supplies residuals and derivatives for a model that has been defined using the

gPROMS modeling language. An external newton-type solver is used to solve this model on the unit level.

A sequential-modular solver component has been developed. Further, a solver component for integrated simultaneous-modular and equation-oriented simulation has been built using Java (Nicholls, 1999). It uses the CAPE-OPEN compliant ICapeNumericNonlinearAlgebraicSystem as a Newton-type solver formulation to solve the overall flowsheet equation system. These two different approaches use a single interface definition for a solver component and are therefore exchangeable.

All these parts have been presented at demonstrations at ESCAPE-9 and the final meeting in Paris. The following two figures represent the set-up of the demonstration which consists of two parts. The first part focuses on solving a flowsheet, whereas the other part emphasizes tool integration. Both demonstrations use a heterogeneous environment, where a simple user interface and the SMST prototype run on a Windows PC, all remaining components are implemented on a Sun Unix workstation.

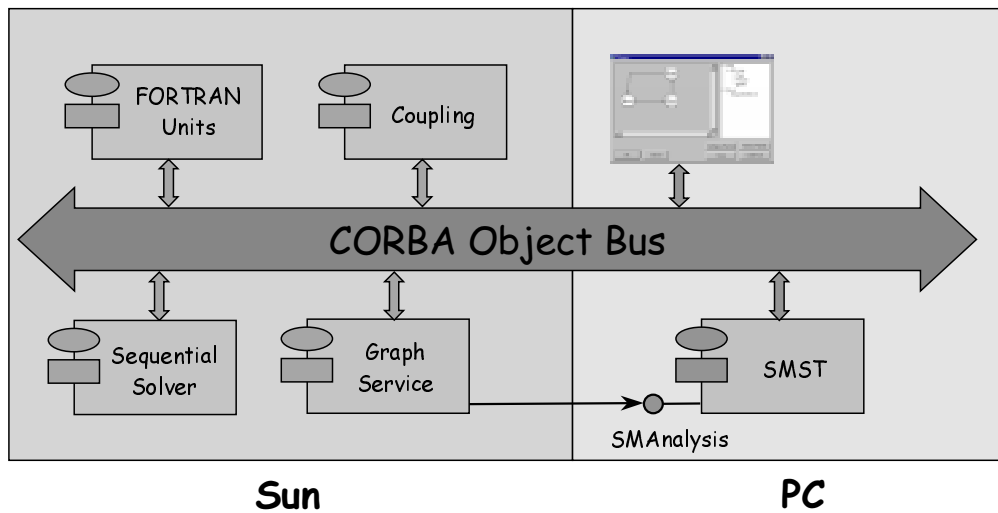


Figure 6: First part of the Cheops demonstration

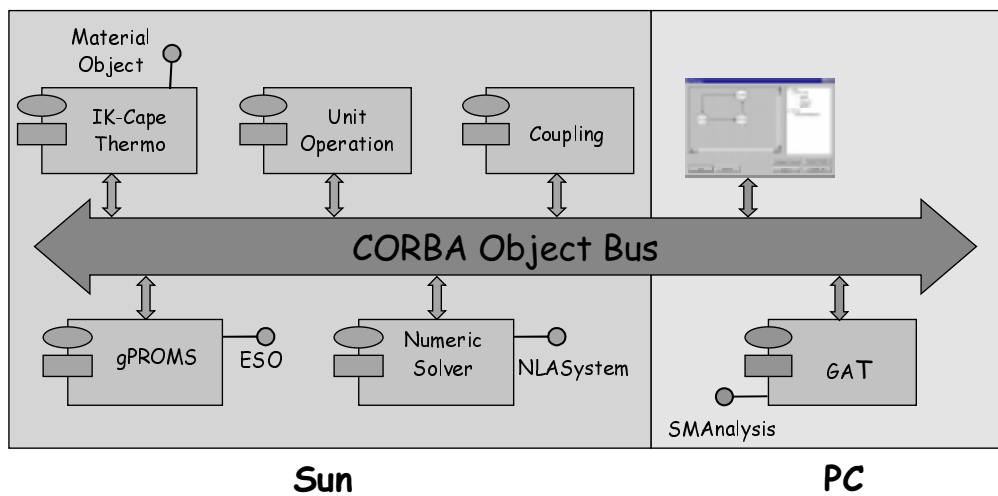


Figure 7: Second part of Cheops demonstration

4. Components with CORBA

The second part of the work in the PATH workpackage was the exploration of the CORBA middleware platform. For this purpose the explorative simulator workbench CHEOPS has been developed using CORBA. The idea was to use this workbench not only for technology exploration but also for experimental work during conceptual work as described in Chapter 2.

CORBA (Common Object Broker Request Architecture) is a standard for distributed objects proposed by the Object Management Group (OMG). The OMG is a consortium of more than 700 companies which actively develop middleware standards. The central piece of work is the Object Management Architecture (OMA), which defines the CORBA standard for the Object Request Broker together with a binary independent protocol (GIOP and IIOP). Interface definitions are expressed in IDL (interface definition language).

```
interface UnitOperation : Cheops::CheopsConcept {
    PortIterator getInputPortIterator ();
    PortIterator getOutputIterator ();
    void addInputPort (in Port p, in string name);
    void addOutputPort (in Port p, in string name);
    void deletePort (in sting name);
    string checkPortConfiguration ();

    VariableIterator getVariableIterator ();
    void addVariable (in Variable v, in string name);
    void deleteVariable (in string name);
    int dof(); // returns number of unspecified degrees of freedom

    attribute PhaseSystem Material;

    void preCalculationCheck() raises
(InvalidConfigurationException);
    void compute() raises (CannotConvergeException);
};
exception InvalidConfigurationException {
    string reason;
};
```

Figure 8: An example of the interface definition language (IDL) for CORBA

The IDL is a declarative language specifying all interfaces inside the OMA following an objectoriented approach. It provides no description of the behavior of an object and is independent of the language chosen for the actual implementation. See Fig. 8 for a simple example of IDL.

A central element in IDL is the *interface*: it defines the interface of a distributed object. Each interface may be derived from one or more other interfaces (single or multiple inheritance) and specifies the attributes (states) and methods (operations) of an object. The notion of an interface stresses that no assumptions concerning the implementation of an object are made. That implies that components specified in CORBA cannot inherit behavior, only the specification of the interface can be inherited.

Note that CORBA is not a binary standard as opposed to COM. Therefore the separation of an interface and an interface is necessarily designed much cleaner as in COM. On the other hand, CORBA does not allow features like component aggregation as a consequence.

Elements in IDL which are not composed of other elements are called `types`. Predefined types include integer and floating point numbers, characters, and strings. Figure 8 presents an exemplary interface definition of a unit operation. The interface is inherited from a `CheopsObject` which can provide general definitions for objects to fit in the overall architecture of the project. This can include a name, a unique identifier, or services regarding persistence, error handling etc.

The keyword `exception` defines a concept which indicates that an error during program execution has occurred. Exceptions can be routed via the network just as normal objects. When a method detects an abnormal state during execution, it throws an exception which will be propagated back through the calling stack until a method is found that can handle the exception. The unit operation interface defines an exception to indicate that a convergence problem arose during computation.

Finally, the unit operation interface has a set of methods: the method `compute` for example computes the values at the outlet ports given the values at the inlet ports. The method indicates that it may throw a `CannotConvergeException`. The keyword `void` specifies that no value will be returned from this method. This particular method takes no arguments. In general, arguments as well as return values can be elementary types, sets or object references. Argument values are further classified into `in`, `out` and `inout` values. Arguments declared as `in` do not change during the procedure call, `out` arguments are valid only after the procedure call and `inout` values are valid before and after the procedure call but need not necessarily be the same.

The object model of CORBA directly maps to the common object models, e.g. [4]. It provides (multiple) inheritance and aggregation of components. The COM model instead focuses on interfaces and uses the aggregation of interfaces to emulate inheritance. Current CORBA implementations have proven to be very stable in use.

4.1. Implementing CORBA Interfaces

The following section describes how a set of CORBA interfaces as described above can be implemented. The conceptual prerequisite to be able to implement CORBA interfaces is a language mapping from the CORBA IDL to the chosen implementation language. Currently, officially defined language mappings are available for Smalltalk, C++, C, and Java.

An example of the working process is shown in Figure 9: After the interfaces have been specified in IDL, the IDL compiler is invoked and generates code in the target language: stubs for use by clients and the skeleton code to implement the server.

The application specific code for the client (Java in the example) uses the generated code for the client stubs to communicate with the server (here C++) through the middleware layer. In object-oriented languages like Java or C++ the stub definition often includes some kind of object reference which is needed to contact the server on the network. Client stub objects can be used similarly to native objects, but they hide location, language, and operating system details of the server.

The C++ implementation of the server includes the server skeleton code to be called by the middleware runtime system on incoming requests. Orbix (a major CORBA implementation from IONA) for example wraps the skeleton code into a framework class where the actual implementation is derived from.

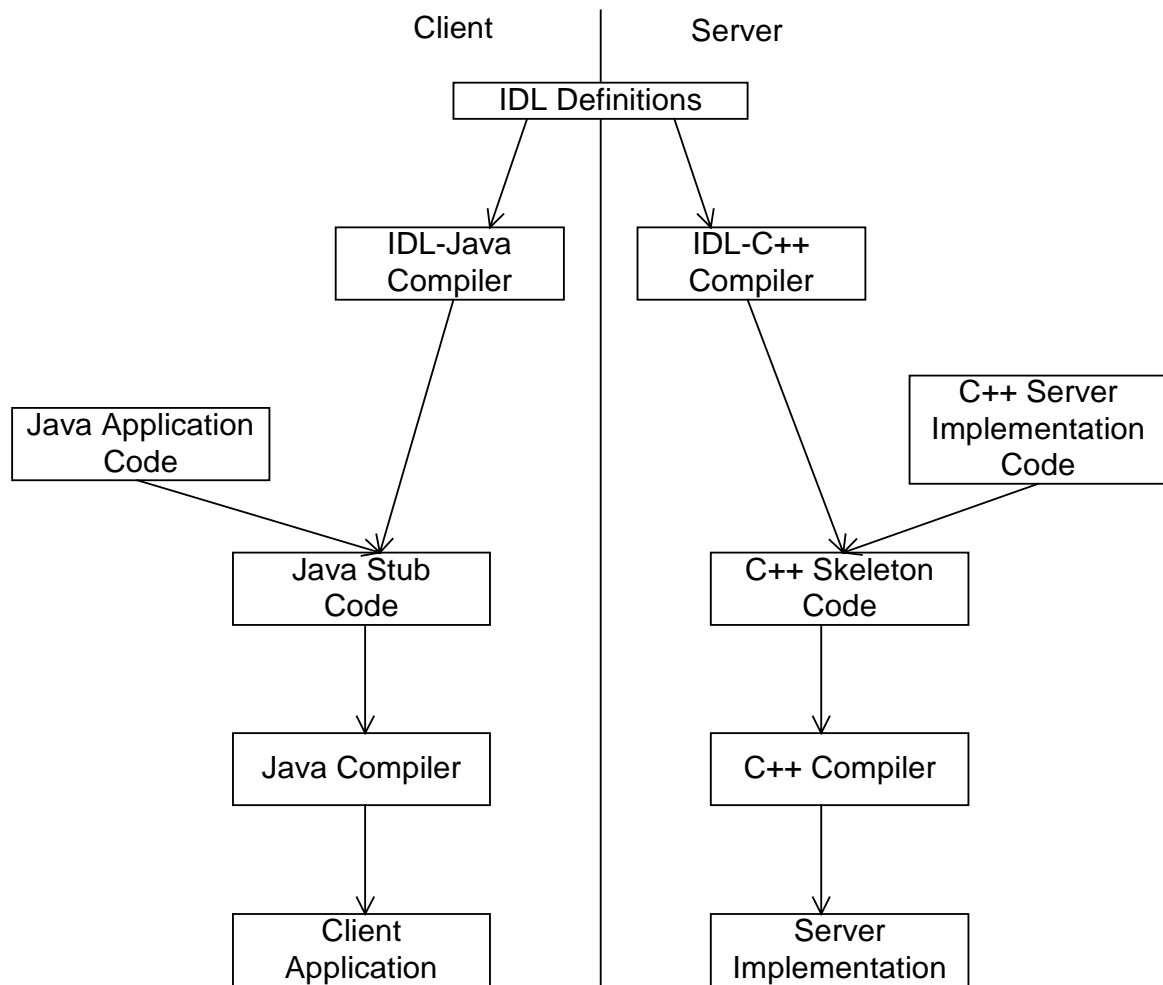


Figure 9: Development Process using CORBA

4.1.1. The Invocation Mechanism

After describing the development steps of clients and servers we will now trace a method invocation of a server requested by a client. CORBA offers two possibilities to issue a method call: static and dynamic. A static method invocation (cf. figure 10) is executed in the following steps (figure):

1. The client calls a method on the stub in its local address space, which routes the call to the object request broker (ORB). The stub translates the language specific data for the arguments and the method name into the protocol used for the communication with the ORB. This is called *marshalling*.
2. The ORB hands the request to the object adapter of the server. The *object adapter* (OA) provides the runtime environment for servers to activate them, pass requests to them, and assign and interpret object references. A server can contain several objects and the object adapter is responsible for relocating a distributed object inside the server.
3. The server implementation invokes the OA to notify that it is active. The required object may have been reactivated from a database for instance.
4. The object adapter passes the method request to the server implementation via the skeleton which calls the predefined method which was overwritten with the actual implementation of the server, this is called *unmarshalling*.

5. The implementation returns control back to the ORB to indicate that the method call is finished.
6. Return values (or an exception) are transmitted from the ORB to the client via the stub. This includes another marshalling/unmarshalling activity in the reverse direction.

In contrast to a static method call, the *dynamic interface invocation* (DII) allows clients to call methods that were not known at compile time. The service resembles the use of the IDispatch interface in COM but takes another approach, though. Whereas in COM ordinary COM system services are used to pass the information on method call and arguments from client to server, CORBA natively integrates this functionality in its definition. Rather, the code generated by the IDL compiler does the marshalling to free the programmer from this exercise. In COM the programmer has to use the slower general marshaling services provided by the system or he has to write his own, more efficient, marshalling code manually.

The counterpart to the dynamic invocation interface on the client side is the *dynamic skeleton invocation* (DSI). It allows the server to add interfaces during runtime.

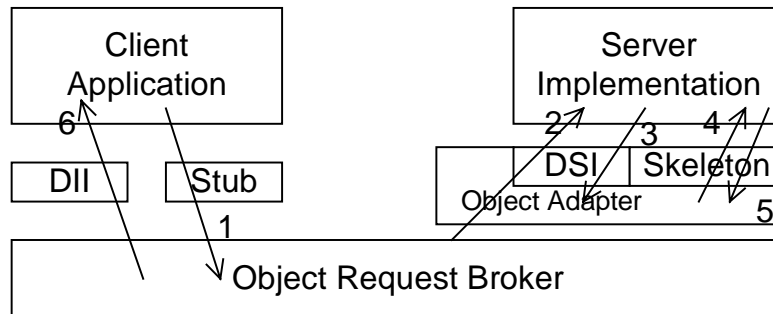


Figure 10: Control flow of a static method invocation in a CORBA environment

4.2. CORBA Services

Beyond the ORB itself a lot of useful services are described in the OMA two of which are clearly of relevance for the CAPE-OPEN project, namely the naming service and the trader service specification.

4.2.1. The Naming Service

The *naming service* is like a telephone book: it associates a name with a given object (name binding). Objects can be retrieved by looking up their names (name resolution). Names can be hierarchically structured by the concept of a *naming context* (figure 11). A naming context groups different names similar to an operating system's directory. Since it can in turn refer to other naming contexts the approach allows the construction of hierarchical name spaces. Clients can navigate through the hierarchy to search objects by names.

The CORBA standard defines no equivalent to the registry of Microsoft Windows. Taking into account that the functionality of the naming service is very similar to that of the registry, one may want to use the naming service to build some kind of replacement in a CORBA-based environment.

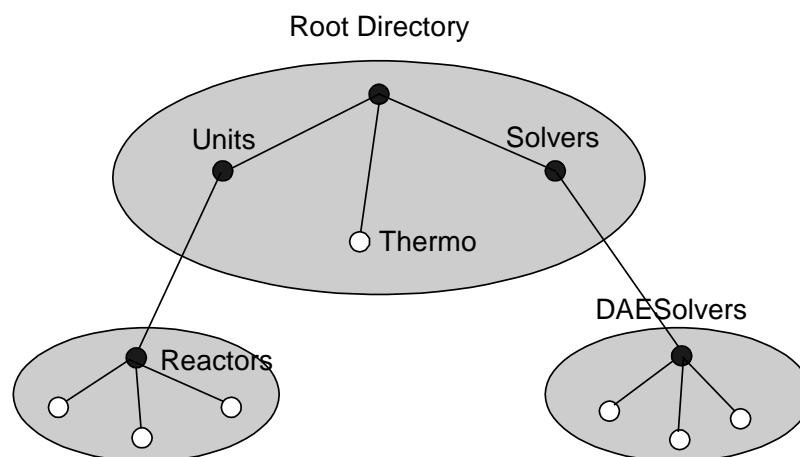


Figure 11: A hierarchy of naming contexts

4.2.2. The Trader Service

The trader service is some sort of directory for advertising services offered by components and properties associated with those objects. The trader keeps a database of registered objects and allows clients to query for components which offer a certain service and fulfill a set of constraints concerning their properties. The trader provides more information on the components registered whereas the naming service uses names which have no semantic value beyond identifying an object.

As an example for trader usage take a set of thermodynamics packages that are able to compute various thermodynamic properties by a service `calculateProperty`. They can be registered with the trader, each announcing a set of computed less common properties and a cost (e.g. a license fee).

Component 1 : Properties={Density, Viscosity, Odor, Color}, Cost=5

Component 2 : Properties={Density, Odor}, Cost = 3

Component 3 : Properties={Odor, Color}, Cost=4

Clients can now query the trader for a thermodynamic component which can compute a set of required properties (for example odor and color) at a minimal cost,

Color in Properties, Odor in Properties, min Cost

and the trader would clearly select component 3.

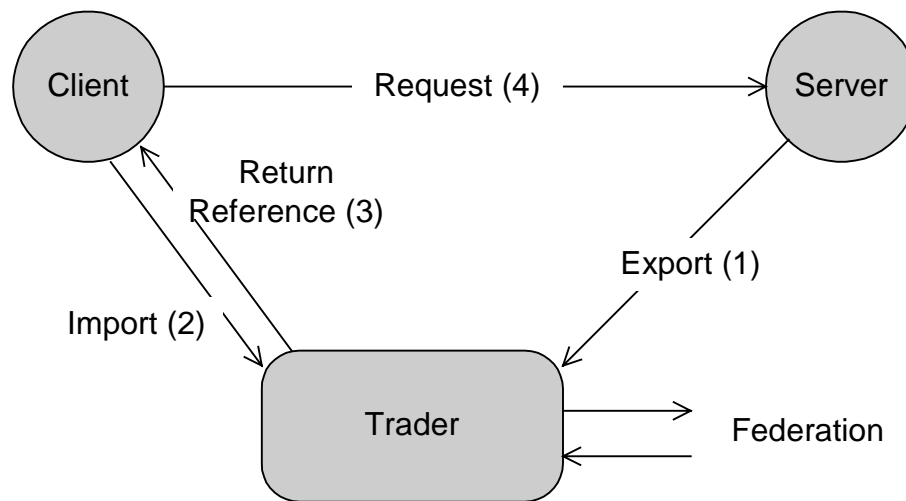


Figure 12: A client using the trader service

Using the trader usually follows the process shown in figure 12: first the server is registered with the trader (export, 1), together with a description of the services provided and a set of characteristic properties. A client will then query (import, 2) for a certain service with given properties. The trader will return a list of object references matching the specified criteria (3). The client will then use the object returned in a normal fashion (4).

The trading service might be very useful to build a tool which can help in component selection and set-up of a component-based simulator. Similar to the example mentioned above, e.g. unit operation modules can be associated with properties describing their phase systems, the spatial distribution, occurring phenomena etc.

Bibliography

Biegler, L. T. (1999). Multi-Solver Modeling for Process Simulation and Optimization., *Proc. FOCAPD '99*, Breckenridge, CO, USA.

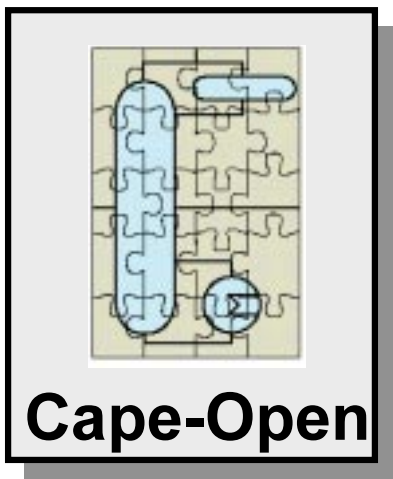
Nicholls, D. (1999). *Erweiterung der Simulationsumgebung CHEOPS um simultane Lösungsstrategien*. Diploma thesis, in progress, RWTH Aachen, 1999.

von Wedel, L., W. Marquardt (1999a). Cheops: A case study in component-based process simulation., *Proc. FOCAPD '99*, Breckenridge, CO, USA.

von Wedel, L., W. Marquardt (1999b). Cheops: Components in process simulation., *Proc. Chemputers*, Duesseldorf, Oct 18-23, Germany.

Jouliia, X.; Jourda, L.: *Material Modeling*. 1997, CAPE-OPEN project document.

Comments on Thermo



An object-oriented view

Lars von Wedel

CO-DSGN-RW-04 Version 1 Januar 2000

Archival Information

Reference	CO-DSGN-RW-04
Authors	Lars von Wedel
Date	09 July, 1998
Number of Pages	??
Version	Version 1
Filename, short	.DOC
Filename, long	
Location: Electronic Hardcopy 1 Hardcopy 2	BSCW:Cape-Open/
Reviewed by	
Approved by	
Distribution	CO
Originating Organisations Reference Number	

Summary

The presented paper designs a thermo package with a deeper object-oriented structure according to the Material Model [CO-DSGN-IN-01] and implements it using a FORTRAN legacy thermodynamic package (IK-CAPE) to show that the approach is feasible. From the design some comments on the current interface specification in Cape-Open is derived: the concept of a process quantity is supported and a better design of a port is proposed to remove the dependency of unit on thermo and to make the integration of equation-oriented and modular simulators easier.

Contents

Appendix A	18
5	Introduction 4
5.1	The Analysis Model revisited - Going to Design 4
5.2	Thermodynamic Quantities 6
6	Phase Systems 7
6.1	Using Dynamic Attributes 8
6.2	The Phase System Interface 9
7	Chemical Components 10
7.1	Mixture Components 10
7.1.1.	Accessing Mixture Properties 10
7.1.2.	Registering External Routines 11
7.2	Pure Components 12
8	The Properties Package 12
9	Further Analysis of Phase Systems 13
10	Using Process Quantities in Thermo 14
11	The Integration of Ports and Thermo 15
12	Concluding Remarks 17
Bibliography	18

5. Introduction

Though there exists a well-defined set of interfaces [CO-THRM-AT-??] to fulfil the requirements which have been stated for an open interface standard for thermodynamic packages [CDD] there are ongoing discussions about a final interface standard [CO-THRM-MIN, CO-MT-MIN]. Furthermore, there have been proposals for structuring the interfaces of a properties package which have emerged from a concise object-oriented analysis [CO-DSGN-IN-01, CO-DSGN-RW-03]. They have been discussed but were never taken into account for whatever reason.

In this paper, the author will show, that

- the proposed interface definitions from the analysis phase (aka Material Model in [CO-DSGN-IN-01, CO-DSGN-RW-03]) can be mapped to modern componentware with a legacy chunk of FORTRAN as an implementation
- the conflicts mentioned in RWTH.LPT's comments on CDD2 and [CO-UNIT-HY-02] can be resolved with a deeper object-oriented structuring of the interfaces,
- the proposed architecture is a natural way forward from the current draft interface specifications [CO-THRM-AT-??], and
- in combination with a correct interpretation of a port a flexible interface useful in modular as well as in an equation-based context can be achieved.

The proposed approach is in general a kind of model-based development. The goal of this approach is to construct software models that directly represent resources and processes of their application domain. In [Klay, 98], the advantages for this approach in the area of legacy systems integration are described as follows:

One of the goals of the model-driven integration is to leverage these legacy systems while providing an easy migration path to model-based systems. Well-designed business models, followed by applying a life cycle development process, can do just that by accessing legacy systems transparently, allowing the functionality of these systems to be incorporated into the model over time, without disrupting corporate solutions. [...] A model-driven integration should focus on the systems engineering of distributed object-oriented business solutions, where the majority of applications are already written. It requires that you analyse the business independently from the applications that will be or are being used to automate certain portions of it. You should model the business logically

It seems natural that an object-oriented model (like the material model is one) of something is a good basis of a migration since the legacy system is based on the same concepts and principles: all thermodynamic packages have been written with the thermodynamic theory as a basis. It seems to a suitable point to reintegrate these systems.

5.1. The Analysis Model revisited - Going to Design

The above discussion shows that a more concise interface definition can be achieved if a more detailed analysis model of thermodynamics is used as a basis for the interface definitions, underlining the comments in [CO-UNITS-HY-02]. In addition,

the interfaces would become more readable and easier to understand if the impedance mismatch between analysis and design would be reduced; one of the advantages of the object-oriented approach.

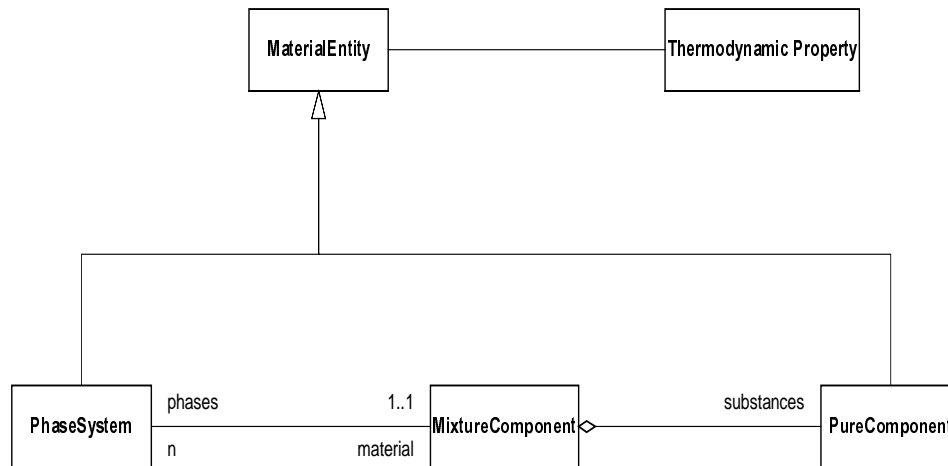


Figure 1: The analysis model from [CO-DSGN-IN-01].

The current specifications of course take into account the results from the analysis phase, otherwise they wouldn't work, i.e. fulfil the given requirements [CDD]. What is being investigated here is whether the design model represented by the interfaces can be made more similar to the analysis model and what are the benefits (and cost) of this approach.

First, let us step back to the analysis model and draw some conclusions about the proposed concepts with respect to design and implementation. The definitions from the Material Model in [CO-DSGN-IN-01] are simplified as follows:

- The *substance* is not taken into account at this time. It is a useful concept when connecting the thermodynamic package to a database which is not the objective of this work. This is however another interesting area of standardization.
- The *entity* is a generalization of objects which can calculate properties. We will not pick it up since for reasons of complexity we currently do not assume a single solution to access properties of different concepts. Later, when these mechanisms are realized homogeneously we will generalize appropriately using the entity concept.
- The *pure component* will be picked up as an interface providing properties of a substance of a single species.
- *Mixture, cluster, and component* are interpreted a bit differently from the original proposal: the information contained in component and cluster is combined into a mixture, however regardless of a thermodynamic context. The cluster will occur implicitly when querying e.g. interaction parameters.
- A *phase system* finally is a mixture located in a thermodynamic context (i.e. associated with a thermodynamic state). According to the material model we will further distinguish single and multi phase system (chapter ?) in a later stage.

An important concept has been mentioned explicitly in [CO-DSGN-IN-01], but has not been discussed any further: the thermodynamic property which is a means to

describe the qualitative behavior of a material entity. This concept is the subject of a longer discussion in [CO-UNIT-HY-02], which we want to pick up in the next section.

5.2. Thermodynamic Quantities

Some more comment on thermodynamic properties: [CO-UNIT-HY-02] distinguishes different natures of properties with respect to their dependencies. The problem itself is important for the interface definitions of a thermodynamic package. Let's elaborate here a bit:

- *Extensive/intensive quantities:* In the author's view, **extensive quantities should not be associated with a thermo package**. Extensive quantities are those which are dependent on the amount of material and can therefore be added. Flows and holdup are not part of the thermo calculation, they are rather properties (in a general sense) of a port or a unit, respectively. Ports will provide flow quantities, whereas units (especially in the dynamic case) will use holdup quantities.

For example: molar enthalpy (J/mol) should be a thermodynamic quantity whereas the enthalpy flow (J/s) should be regarded as a port's property. Hence we propose to use intensive quantities throughout in the thermo package and to let ports and unit do the association with flows or holdups. (See chapter 7 for a proposal of how to use ports.) Putting extensive quantities into the thermodynamic package leads to an inconsistency, since either a flow has to be interpreted as a holdup or vice versa; in each case at least one property has a wrong dimension.

- *State-dependent properties, composition-dependent properties:* The thermodynamic state of a system is described by a set of process quantities where the needed number of quantities to fully describe the state is fixed by Gibb's law. These quantities are *independent*, they can be given arbitrary values, whereas all other values are now fixed in terms of the independent state variables. Taking the latter into account removes the difference between thermodynamic state dependent and composition dependent variables.

The identity of independent properties can change from calculation to calculation depending on the service requested: for an enthalpy calculation pressure, temperature, and composition as well as the aggregate state may be useful, on the other hand flash calculations like a p-,H-flash are likely to occur.

- *Phase dependent quantities:* As proposed in [CO-DSGN-RW-03] single phase systems and multi phase systems can be distinguished. A model such detailed is capable of incorporating the differences between the two types of quantities mentioned in [CO-UNITS-HY-02] in an elegant manner. They are always dependent on a full thermodynamic state (e.g. p, T, and x).
- *Pure Component Properties:* Those properties either depend only on the structure of the molecule itself (e.g. molar weight or group contribution parameters) or also on molecular interactions and hence on pressure and temperature (e.g. specific heat capacity) but never on composition. All composition-dependent properties are only defined in the context of a mixture. Thus we distinguish two kinds of pure component properties: constant and dependent.

- *Mixture properties*: Mixture properties are always dependent on the contained components, as well as on the composition of the mixture. Therefore we can distinguish the same quantity types as for pure components, however they are additionally dependent on composition.
- *Mass-/molar based properties*: A clean way to deal with this duality of properties is needed. At this point we can state that this distinction can probably be made on a very generic level, because we do not expect a single unit to use both bases in an arbitrary fashion. Probably a single thermo client will use one base for specific quantities throughout. This is the assumption for dealing with this issue. If it does not hold, we have to find another approach, e.g. making the distinction on the level of a single process quantity.
- *Derivatives* are not properties per se. Rather they should be seen as a further attribute of a variable (or process quantity). Such concepts have been proposed in [vonWedel, 97] and were also used in [CO-GRP1-BP-03]. The application of this concept in the field of thermodynamics is subject of chapter 6.

The current specification uses a flat but simple structure of a material package. Requested thermodynamic and physical quantities are identified by tags such as *enthalpy*. But enthalpy is not just that: it can refer to different aggregate states, to single components, to the contribution of single component enthalpy to the overall mixture enthalpy, or one is interested in the derivatives of enthalpy. Those different flavors of course relates to the analysis model. The current design of the interfaces partially translates the different concepts into a set of modifiers used to build a character string for identification of the properties:

[Mass/Molar]{Partial}Propertyname{.d[T/p/x]}

The proposed way in this paper avoids those lengthy identifiers and the needed computations to translate them by using a deeper object-oriented representation of the involved concepts. During the recent method and tools meeting this approach was discussed as a way towards a thermodynamic interface that better reflects chemical engineering knowledge [CO-MIN-MT].

After we have now recapitulated the analysis model and some of the simplifications being made, we will now step into the design versions of the concepts which have been identified as relevant in the above discussion.

6. Phase Systems

Let us start with a client view of a thermodynamic package. Models of chemical (and petroleum, and ...) processes (unit operations with ports) are usually described in terms of *phases*. These offer an interesting perspective of a material since a client *usually* does not need to know anything about how the properties of a phase system are determined. It does not need to know, which thermodynamic approaches are being used. This is defined within the thermo package. The concept of a phase system provides a level of *encapsulation and abstraction* which is very important and convenient in open systems.

The concept of a phase system is quite comparable with the `MaterialObject` in [CO-THRM-AT-??], the current proposal. (By the way, I strongly recommend not to name software constructs `XXXComponent` or `XXXObject`, just for a homogeneous naming convention: we do not say `UnitOperationObject` or `PortComponent` either).

It is important to note that no assumption is made on how a phase system is implemented. It may be implemented as a part of an existing simulator executive or as a standalone model. Using a standalone model not only reduces the work that is needed for a client to use a thermodynamic package and is thus preferred.

From a software engineering perspective it is an absolute must to make the unit operation and thermo module as independent as possible. With the current specifications it is not possible to exchange data between unit and executive without a material object, *the thermo package is on the critical path of the unit package*. If this material object is implemented by the simulator executive there is no way for a unit to implement its own thermodynamic calculation routines. *The origin of this problem is an incorrect understanding of the port concept which has not been used as the required encapsulation layer between unit and executive*. In the proposed model in chapter 7 the thermo is transparently hidden behind the port.

6.1. Using Dynamic Attributes

The interface of a `MaterialObject` in [CO-THRM-AT-] has been defined in an interesting manner: instead of implementing a fixed number of operations, a *dynamic attribute* [Mowbray, 97] has been used to accomplish the following requirements:

- *Extensibility of a properties package with foreign routines (external routines):* a foreign property routine must be registered dynamic, i.e. during run-time. It must be identifiable by something - a name or whatever.
- *Exploitation of the capabilities of existing software:* note that different properties packages may implement different properties. It would be ugly to implement a set of methods just returning an error while other features of the package cannot be used due to a restrictive interface definition.
- *Less cluttered interface definition:* An interface with probably hundreds of methods is hard to maintain and implement. Dynamic attributed may be used to simplify the interface definition.
- *Increased robustness of the architecture* due to more stable interfaces.

Facing those advantages, we expect run-time errors more likely to occur since they can no longer be discovered during the compilation phase. *Hence, extended capabilities for component selection and error handling are required*. A first step is the incorporation of a method returning all implemented tags to let a client know which quantities it can expect to be calculated.

Another disadvantage may be a lack of speed. Open standards probably have certain disadvantages in terms of speed and flexibility. Dynamic attributes however are not to the bottleneck compared e.g. to componentware's marshaling/unmarshaling process [von Wedel, 98; Orfali, Harkey, 97].

A third problem, maybe the biggest one, is the needed standardization of property names. Not only need property names be standardized during the project, an institution supervising property naming is necessary to avoid ambiguities like duplicate names for a single property or a single name for certain properties. Existing institutions like pdXi or SVPPM could take care of this point in the future.

Though there is currently no solution for the latter problem the author strongly supports the dynamic attribute pattern for phase system properties.

A remark on the technical details: in effect the dynamic attribute mirrors modern middleware facilities for late binding, but using a simpler way than e.g. using DSI/DII in CORBA. When using COM we could also use the QueryInterface mechanism to support the functionality discussed above. The individual interfaces queried for would then be derived from the ProcessQuantity interface described in section 6. These capabilities should be a topic of further investigations. A structuring of the interfaces as shown here is a necessary step for such work.

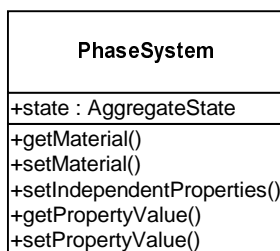


Figure 2: Interface definition of a phase system

6.2. The Phase System Interface

Some rationales about the concepts: The link to a (yet undefined) chemical component (`material`) is the outcome of the discussion in [CO-UNITS-HY-02]. The material of a phase system actually implements the thermodynamic calculations, whereas the phase system is just the client's handle to it. Through this material link, the client can access mixture and pure component properties separately. The calculation method `getPropertyValue` uses a dynamic attribute to identify the set of properties that shall be calculated. The method `setPropertyValue` can be used to set independent quantities or to supply estimates for dependent properties. The method signature has not been elaborated because the more structured approach using process quantities is finally used in the implementation of these ideas.

The approach chosen in CHEOPS, the CORBA workbench, goes further and implements some of the ideas which are also briefly mentioned in [CO-UNITS-HY-02]: CHEOPS uses the concept of a process quantity as a model of a thermodynamic property (see section 6 for more). Such a concept has also been used in [Jourda, 96]. Using process quantities to query thermodynamic properties avoids the use of dynamic attributes by turning the access mechanism into an ordinary collection of objects.

At this point we will not make a distinction between properties comprising several phases (macroscopic properties such as overall composition,) describing single phases, and describing interphase phenomena (eg. equilibrium ratios or surface tension). This will be the topic of a more detailed analysis in section 5.

For the remainder of this paper, the `PhaseSystem` shall be the client's (e.g. a unit's) view of the thermodynamic package. In an ideal case, no further information than intensive properties of a phase system as described above should be required; in this case a phase system is a sufficient source of thermodynamic services. More sophisticated implementations of a unit operations can of course have a need for

more information describing the material associated; this is defined within the thermo package and the topic of the following chapter.

7. Chemical Components

We have now elaborated the interaction between client and thermo server for the ideal case in which the actual configuration of the material does not matter.

Now let's assume that the client is interested in some of the underlying properties which further characterize the material present in a phase system. Those properties can be described by a mixture containing several pure components. Note that the former statements exactly mirrors the UML diagram of figure 1.

7.1. Mixture Components

7.1.1. Accessing Mixture Properties

Taking into account the above discussion on thermodynamic properties in mixtures we could distinguish operations to query properties dependent on composition only (e.g. molar weight) and those dependent on pressure, temperature, pressure and composition. This would lead to two methods which would be hard to distinguish from a client programmer's viewpoint (which method supplies which quantity?). The distinction would be even ambiguous if a usually non-constant property (eg. specific heat capacity) is provided as a constant with respect to e.g. pressure and temperature.

Hence we propose to use a single method `calculateProperty` for all mixture properties, again using a dynamic attribute in conjunction with the independent variables as parameters. The method almost resembles the functionality of the current specifications and can be extended to calculate several properties at once to allow for internal optimization. Again, the approach is not further investigated here as we will introduce a different mechanism to query properties using process quantities in chapter 6.

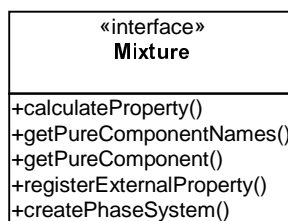


Figure 3: Interface definitions for a `MixtureComponent`

The rationale for most definitions follows directly from the discussion above. There is one additional method to discuss, `createPhaseSystem`. Instead of using an isolated factory (a `MaterialTemplate`) we use the `MixtureComponent` to create a `PhaseSystem`. The mixture is advantageous over the material template as it establishes a link between the thermodynamic configuration (the mixture) and a client's view, the phase system directly on the interface level. Another important fact is that a unit

may wish to derive several phase systems, e.g. a distillation column for each of its trays. Overall, the mixture is very similar to the material template in that

- it can be associated with units to set a certain configuration for those concepts,
- implements the factory methods [GOF94] to create `PhaseSystems`,
- it can be used to support naming mapping
- it can support reindexing, lumping/delumping for pseudo-components on the fly

with the additional advantage of explicitly making an association between a thermodynamic configuration (mixture) and a client's handle for thermodynamic calculations (a phase system).

7.1.2. Registering External Routines

Another point to discuss is the proposed process to register external thermodynamic properties. Since we do not only want to add new routines but also override existing routines with custom implementations we have to route all requests through a central component, a dispatcher, where new routines have to be registered.

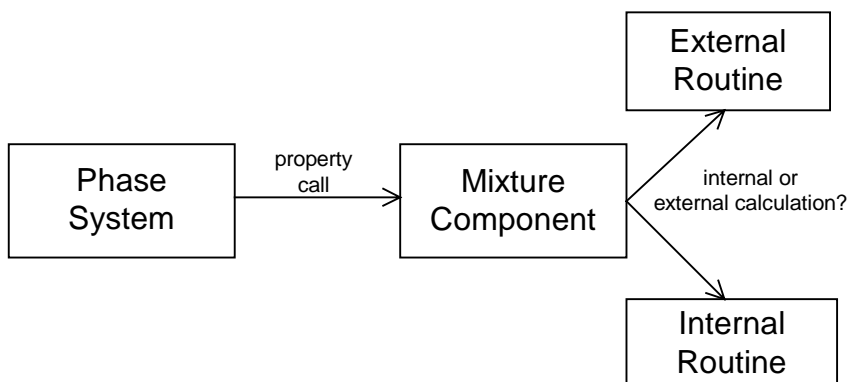


Figure 4: Pattern for registering external properties

Since external routines are in general supplied for single thermodynamic configurations the mixture seems to be an appropriate place to put the registration functionality in.

Similarly to the material object/template pair, all properties calculations requested for a phase are internally routed through the mixture where we can register an external property with the mixture. The mixture will decide whether a requested property can be calculated internally or if it is provided by an external component.

To calculate a property we probably need information about the involved substances, e.g. molar weight, specific heat capacity etc. Hence we pass the mixture to the external property during initialisation to make those information available. This process of registering external properties is currently not covered by the interface specification around in thermo. Refer to figure 5 for a proposal how to make this information accessible. Note that some method calls to query the mixture properties have been grouped as `getPropertyValue`.

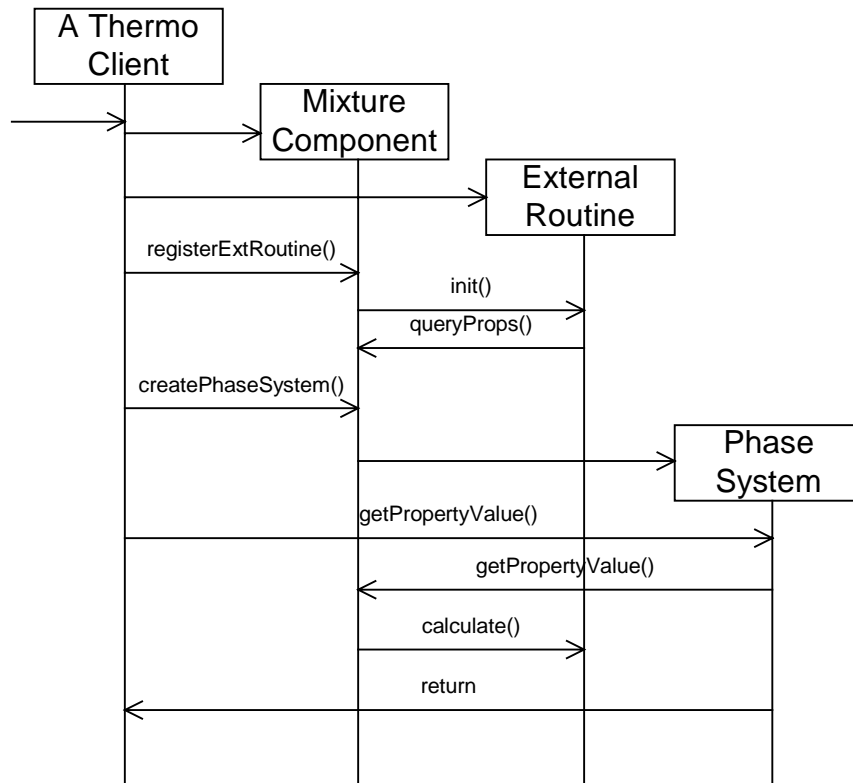


Figure 5: A sequence diagram for registration of external routines

7.2. Pure Components

At this point we can follow the same argumentation as above and use a single method `calculateProperty` to calculate pure component properties. The composition can be neglected in this case.

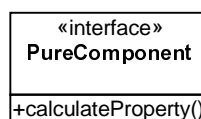


Figure 6: Interface definition for a pure component

After we have elaborated a picture of the interfaces involved in properties calculations for phase systems and material properties we now have to develop a model for setting up and initialising the complete environment.

8. The Properties Package

The interface of a `PropertyPackage` provides, just as in the current draft specification, central services for thermodynamic calculations. After a property package has been launched and configured, it offers a set of mixtures. The mixture is in this sense similar to the option set of former proposals of the thermodynamic interface specifications. Mixtures can differ in their components or only in the thermodynamic options (e.g. high-pressure/low-pressure).

Existing properties packages can use mixtures (aka option sets) in different ways:

- A property package could employ a single set of pure components but use different option sets: those would be exposed as distinct mixtures, yet each with the same set of components (e.g. PropertiesPlus).
- A property package using only a single option set for a fixed set of pure components would provide only a single mixture (e.g. the Hyprotech properties package).
- Properties packages like IK-CAPE can use different pure component sets each with its own set of thermodynamic options.

The configuration of a property package is outside of the scope of this work. It is even neither necessary nor desirable to include the configuration here. With respect to existing legacy systems it is better to define which do not cover the configuration since this may vary from system to system.

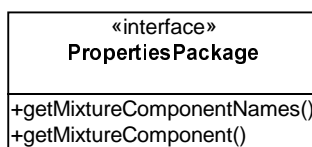


Figure 7: Interface specification of a `PropertiesPackage`

Now we can draw a complete picture of the interaction between a thermo client and the different components in the thermodynamic module. What has not been covered is the configuration of a properties package.

9. Further Analysis of Phase Systems

Looking in more detail at what a phase system really is we can distinguish *single phase systems* and *multiple phase systems* [CO-UNIT-HY-02, CO-DSGN-IN-01, CO-DSGN-RW-03].

The two concepts can be used to further discriminate phase system properties:

- All properties applying to a combination of phases are *only* accessible through a multiphase system. *Equilibrium ratios, phase fractions, or surface tension* are examples among others.
- All properties applying to a single phase (e.g. *enthalpy*) can also be applied to a multiphase system. In this case they are interpreted as a global (macroscopic) property of the system as a whole.

Since we are describing throughout phase systems in equilibrium state we assume that temperature, pressure, and fugacities are equal in all phases. A single phase is additionally described by its aggregate state (vapor, liquid, solid). Multiple phase systems consist of several single phases and can be queried for them (figure 8).

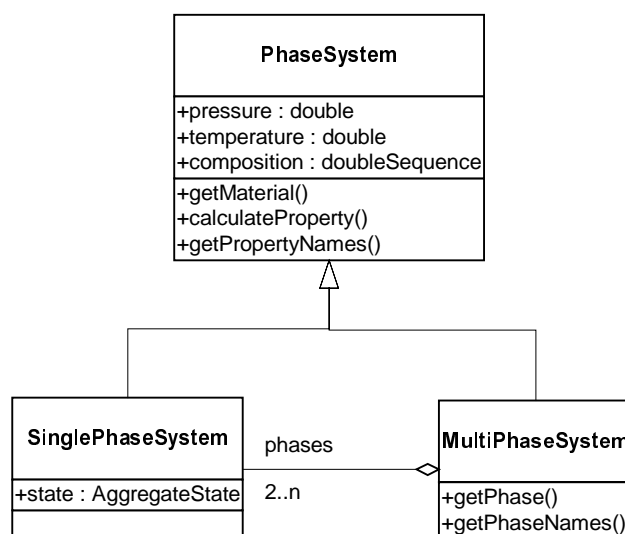


Figure 8: Interface specification in UML with `SinglePhaseSystems` and `MultiPhaseSystem`

Just as the former phase systems were created using the mixture as a factory, we create single phase and multi phase systems using two similar methods on the mixture. The method `createSinglePhaseSystem` takes the specified aggregate state which cannot be changed during the lifecycle of the object, since no flash calculation or phase tests are integrated into the single phase system. A method `createMultiPhaseSystem`, given a list of aggregate states for the contained single phases, will create a set of single phase systems plus a containing multi phase system. The interface proposal is very open in the sense that it allows an arbitrary number of phases, which may be more than we need at this point. Being open is important though.

10. Using Process Quantities in Thermo

Within [von Wedel, 1997], the concept of a `ProcessQuantity` has been used as a standard way data exchange between components. The concept is based on [Marquardt, 1997; Bogusch/Marquardt 1997], showing that it is reasonable to consider a quantity not just as a value but as a concept with additional properties like dimensionality, derivatives, or a status (computed, estimated, specified, etc.). The current proposal for a unit operation interface also suggests such a concept.

Such a concept has proven quite useful during the work in CHEOPS as it enables an open interface to different types of packages: an optimizer e.g. could use a set of quantities as the optimized variables and another set as manipulated variables without explicitly knowing where they belong to. The design of the system is also influenced in a positive manner: exchanging values in a standard way in *all* cases makes the standard homogeneous and transparent and hence easier to understand and to use.

Nevertheless, the concept hasn't made its way into other packages of which especially thermo is of interest here. On our way towards evaluating a more

elaborate interface of a thermo package, we will employ process quantities as a means to communicate values and relevant attributes. Instead of using doubles and sequences to pass around data we can employ process quantities for pure components, mixtures and phase systems (figure 9). Using process quantities as a base concept in all modules can lead to a simpler and more transparent standard definition.

Having made an association between those concepts in the same manner we can extract this feature (together with the access mechanisms) into a common superclass which is called `MaterialEntity`.

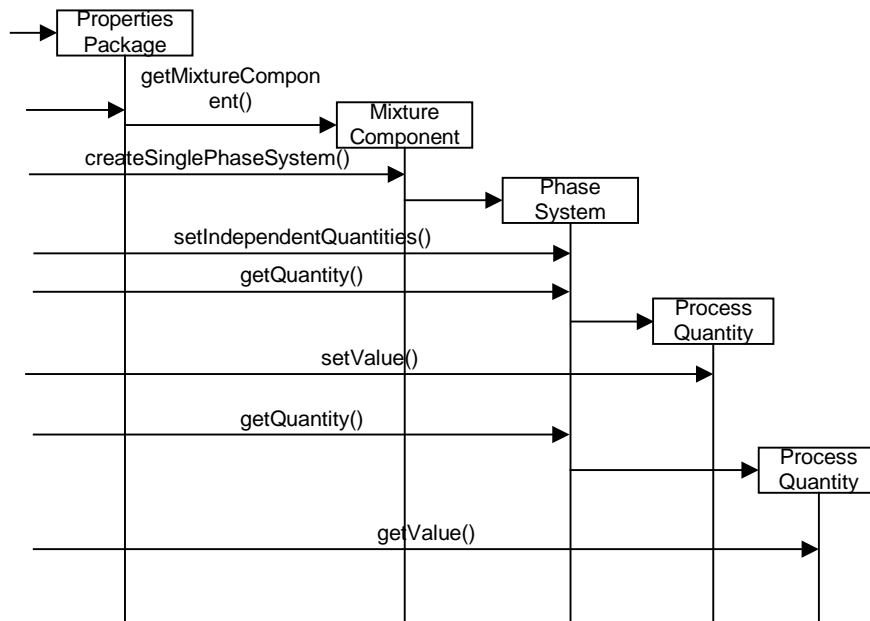


Figure 9: Using process quantities to access thermodynamic properties.

Process quantities allow performance improvements similar to the result set worked out in the thermo group, like value caching or updating several quantities at once. When caching is employed, querying a value from a process quantity associated with a phase system can launch its calculation if the process quantity is not up to date. Otherwise the value is returned, because the thermodynamic state has not been changed since the last calculation.

To perform more effective calculations, we can pass a list of properties to update in a single calculation. This calculation enables the underlying property package to perform the calculations in an efficient order. However, this is an implementation detail and not considered any further.

11. The Integration of Ports and Thermo

The current Cape-Open proposal uses the `MaterialObject` as a backdoor to the legacy simulator internals. *That implies that each client has to supply and implement its own `MaterialObject`.* The argument does not just apply to simulator executives but even to the tiniest C++-program which wants to use a Cape-Open

unit operation module. From a software perspective the two modules, unit and thermo, are unavoidable dependent – there is no chance to use thermo standalone because the ports do not provide any access mechanism for the transferred variables but via the material template.

In the author's opinion this problem arises due to an incorrect interpretation of a port. If ports are used as a backdoor to the simulator internals, we end up with an easier approach. *Therefore we propose to use the port as a backdoor to the simulator's internals.* This was always the intention of a port, a well understood means to exchange data between unit and unit client (the simulator executive in this case): the unit client puts the values into the port (using standard access mechanisms or a backdoor) and the unit retrieves the values using standard access mechanisms. The unit is aware of the correspondence between its port variables and some of its internal variables. This concept is an absolutely mandatory encapsulation layer between a unit and its foreign environment. The thermo package is needed

- to compute additional quantities a unit needs for its calculation, and
- to enable the executive to convert between different representations of a unit's port variables.
- Another important point is that the current implementation of port and material object seem to map rather bad to an equation-oriented environment.

Note that the possibility to convert unit representation also applies to the mass-/molar-based dualism of the thermodynamic properties. Another solution is to put this conversion mechanism into the port object itself. A unit would then have to take care that it uses a consistent *implementation* of a port. The difference need not necessarily show on the interface level. If a port is implemented by the simulator executive both proposals coincide (figure 10).

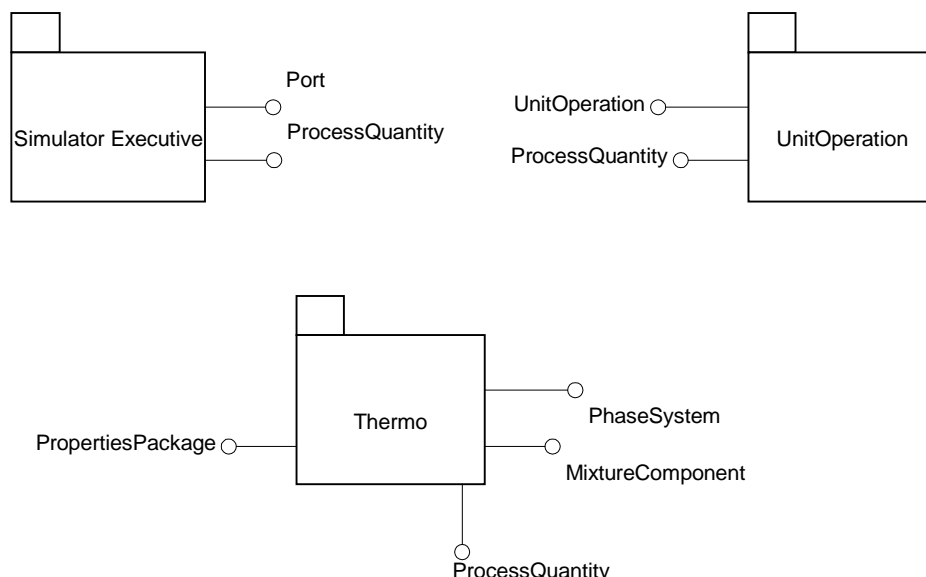


Figure 10: Deployment diagram for the interaction between port, unit, and thermo.

The port of a unit is an intermediate object that no one has requested to be interchangeable and is therefore a good place to put a backdoor in. But still we have to consider the port interface as open if we do not want to fix a stream format. The

proposed solution is similar to the open approach in the thermo package itself: using a dynamic attribute, we can keep the port interface open to cover different specifications of a unit operations in- and output.

If a port as a part of a unit is aware of the transmitted variables (again by process quantities), equation-oriented and modular contexts can be realized easily [Marquardt, Pantelides, 98]:

- In the case of an EO unit, the internal equations additionally have to identify corresponding unit and port variables.
- For SM units, the internal stream format can be calculated using the port's thermo handle.

A simulator executive needs some knowledge about the port variables, either explicitly on the interface level or through an internal backdoor. In a fully component-based environment there must be a way for the executive to query the independent variables which have to be passed from one unit to another (SM context) or which have to be identified (EO context). Hence we can imagine two flavors of ports:

- Those which have an *explicitly fixed set of variables*, e.g. temperature T, total flowrate N, and partial pressures $p_{x[i]}$. They are chosen since they are sufficiently describing a thermodynamic state (with $p = \sum(p[i])$) and do not introduce further constraints as is the case with concentrations [Johns, 98]. This would yield for partial flowrates as well, but they have the disadvantage of being zero for a vanishing phase. This approach seems useful if ports are implemented as standalone components in a fully component-based environment like CHEOPS [vonWedel, 97]. Note that the independent properties of the thermo package can be derived easily from the proposed representation.
- The other sort of ports does not have fixed quantities. Some additional information is necessary for the executive to know which variables have to be transferred and converged. This could be achieved by implementing the port interface on behalf of the simulator executive.

Both types of ports are very similar. Indeed the first type is a specialized type of the second which has emerged through fixing the set of transferred quantities.

If we finally compare the proposed interpretation of a port with the current material object we see that with few changes in the current proposals we can achieve a cleaner picture of the overall standard: *The current MaterialObject should be moved into the unit package and work as a port*. The implementation is essentially the same but using this approach allows to build a standalone unit prototype that is not dependent on the thermo specification. The thermo package should instead focus on a standalone package with a deeper object oriented structure. The current port definition in the unit package would then have some sort of behavior as opposed to a simple reference that it is now.

12. Concluding Remarks

An implementation of the proposed interfaces is currently in progress, based on the FORTRAN library IK-CAPE which has been wrapped as a whole. No changes have

been made to its implementation to show that the approach can be easily realized on legacy systems. Figure 9 shows the deployment of the interfaces on the thermo component: the components are all implemented in a single component. They do not add any piece of functionality, they just provide a structured and system transparent access to existing functionality.

What are the next steps? Of course, a list of properties has to be set up and associated with the correct material entity. Former discussions showed that this is not always easy. If the definition of the thermodynamic properties is carefully evaluated, these problems should not occur. There are properties which can be retrieved from a pure component as a scalar (e.g. ideal enthalpy), but which could also be retrieved from the mixture as a vector (list of ideal enthalpies of all components). Quantities of this type could be accessible through both interfaces. This is also true for the MolarWeight, but in this case there is another meaningful concept, that of an AverageMolarWeight which is additionally composition dependent and a mixture property. The PartialMolarEnthalpy is not defined for pure components per se, it is only defined in the context of a mixture and is therefore a vector mixture property.

However, we should remember the problems we encountered with dynamic attributes. They have to be picked up and solved to make such systems work reliably and efficiently:

- Improved capabilities for component set-up and selection are required.
- Property names have to be standardized. There needs to be something which supervises the process of naming properties to avoid ambiguities especially for external routines.

A disadvantage with the proposed interfaces is that they require more efforts to implement them. The advantage of the open-ness reached and the object model closer to the analysis concepts however have a distinct advantage when the results and interfaces of this work shall be used in other contexts since they build on a clean model of the reality which is valid in all applications.

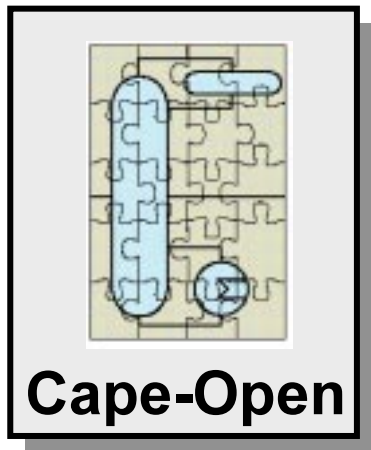
Bibliography

- [CO-THRM-AT-??] Thermo Workpackage Team: *Physical Properties Interface Model & Specification Draft*. Cape-Open Internal Working Document, 1998.
- [CDD] Cape-Open: *Conceptual Design Document*. 1998.
- [CO-THRM-MIN] Thermo Workpackage Team: *Minutes of the Thermo Meeting in Henley*. May, 1998.
- [CO-DSGN-IN-01] Joulia, X.; Jourda, L.: *Material Modeling*. 1997.
- [CO-DSGN-RW-03] Bogusch, R.; Marquardt, W.; Schneider, R.; von Wedel, L.: *Comments on Streams, Thermo, and Units*. 1997.
- [CO-UNIT-HY-02] Sama, S.: *Requirements for the design of the interface for Material Objects from a CAPE-OPEN Unit Operation's perspective*. 1998.

- [Klay, 98] Klay, H.: *He's got the Whole World in His Hands*. Application Development Advisor, Vol. 1, No.6. 1998.
- [CO-MT-MIN] Method & Tools Team: *Minutes of the Barcelona meeting*. June 1998.
- [Jourda, 96] Jourda, L.: *Composants logiciels orientes-objet pour la modelisation et la simulation des procedes chimiques*. Doctoral thesis. Toulouse, 1996.
- [von Wedel, 98] von Wedel, L.: *PATH Presentation*. Lyon, March 1998.
- [von Wedel, 97] von Wedel, L.: *A Component-Based Approach to Open Process Simulation*. Diploma Thesis, Aachen/Rueil-Malmaison, 1997.
- [CO-GRP1-BP-03] Banks, P., et al.: *CAPE-OPEN Interface System. Unit Operations: Draft Interface Specification*. CAPE-OPEN working paper. 1998.
- [GOF] Gamma, et al.: *Design Patterns – Elements of Reusable Software*. Addison Wesley, 1994.
- [Mowbray, 97] Mowbray, T.: *CORBA Design Patterns*. John Wiley, 1997.
- [Marquardt, 97] Marquardt, W.: *Comments on Equation Set Objects*. Personal Communications, 1997.
- [Bogusch, Marquardt, 1997] *A formal representation of Process Model Equations*. Computers chem. Engng 21 (1997), 1105-1115.
- [Orfali, Harkey, 97] Orfali, Robert; Harkey, Dan: *Client/Server Programming with Java and CORBA*. John Wiley, 1997.
- [Marquardt, Pantelides, 98] Marquardt, W., Pantelides, C.: *Proposal for CAPE-OPEN Material Objects*. 1998.

Appendix B

Comments on Streams, Thermo, and Units



CO-DSGN-RW-03 Version 1 (Draft) January, 2000

**Ralf Bogusch, Wolfgang Marquardt,
Ralph Schneider, Lars von Wedel**

Archival Information

Reference	CO-DSGN-RW-03
Authors	Ralf Bogusch, Wolfgang Marquardt, Ralph Schneider, Lars von Wedel
Date	20 June, 1997
Number of Pages	22
Version	Version 1 (Draft)
Filename, short	11rw031.doc
Filename, long	11rw031.doc CCPT Design
Location: Electronic Hardcopy 1 Hardcopy 2	CAPE-Open/Subprojects intermediate results/1- CCPT/DSGN/ARCHITECTURE http://sunsite.informatik.rwth-aachen.de/bscw/bscw.cgi/0/9464
Reviewed by 1 Date 2 Date Reviewed by 1 Date 2 Date Reviewed by 1 Date 2 Date	
Approved by	Not yet approved
Distribution	CAPE-OPEN

Contents

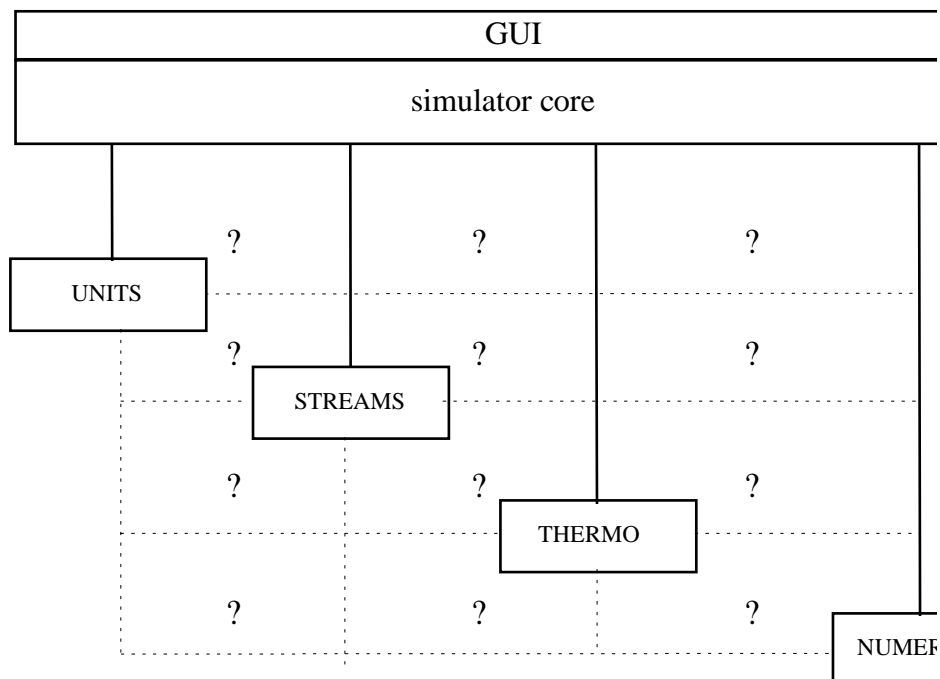
Appendix B	19
13	Introduction
.....	1
14	Streams
.....	2
14.1	Overview on Existing Documents
.....	2
14.2	Ports
.....	3
14.3	Connections
.....	4
15	Thermo
.....	8
15.1	Overview on Existing Documents
.....	8
15.2	Material Concept
.....	8
15.2.1.	Example: Computation of phase enthalpy
.....	10
15.3	Association of Thermodynamic Concepts with Units and Streams
.....	12
16	Units
.....	13
16.1	Elementary Units
.....	14
16.2	Composite Units
.....	16
17	Summary
.....	18
18	References
.....	20

13. Introduction

The current situation in the CAPE-OPEN project is the stage of specifying the work packages, in particular streams, thermo, units, and numerics regardless of the possible interactions of these work packages (see Figure 13-1). There is a risk to create inconsistencies in the design considerations because of missing links between the work packages at the conceptual level. In this paper we comment on the existing proposals for streams, thermo, and units. Furthermore, we want to propagate a conceptual structure in an object-oriented way. Such a structuring approach has to ensure the following items:

- interchangeability of components
- possibility to extend the simulator
- flexibility of the simulator
- communication between different components without failure

This paper spotlights only some of the basic ideas. It is a draft note far away from being a complete presentation of the topic. We are interested in your opinion on this structuring approach, especially if you think it is useful for CAPE-OPEN to develop these ideas further.



----- possible (at the moment missing) interconnections

Figure 13-1 Status quo in CAPE-OPEN

14. Streams

14.1. Overview on Existing Documents

The documents dealing with streams in CAPE-OPEN have quite different views on what a stream is. Benaouda et al. [1997] propose to define stream standards. They distinguish four different types of streams (material, energy, information, numeric/dynamic) transmitting information from one unit to another (see Figure 14-1). For each stream type they define attributes on three different levels (from compulsory to optional). If the attributes of units on either side are not identical, the co-operation of both units fails. This has to be checked whenever connections between units are established.

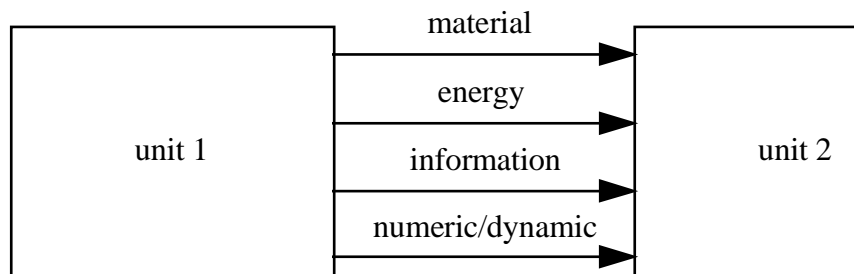


Figure 14-1 Standardised streams in the sense of Benaouda et al. [1997]

Defining stream standards imposes tight constraints on unit operation developers since only a minimal set of stream attributes should be included in a stream to achieve a precise and lean interface definition. If all parameters of existing unit operation's code shall be covered by a standard interface, it will have to include a broad range of parameters most of which are used very seldom. We think that this can lead to a proliferation of process quantities on the optional level. The challenging problem is how a new process quantity can be added to an existing stream definition. In our opinion this should be possible as easily as adding a new physical property. The latter may even force an existing stream definition to be enhanced, consider ill-defined process quantities similar to those mentioned in Johns [1997d]. Those should not be included in every unit operation stream. Inheritance is of no great use here: If an existing stream interface is specialised by several developers all derived interface definitions are incompatible with each other (This is exactly the same reason why OLE interfaces cannot be extended by inheritance.) The same applies for the numerical part: exchanging a numerical routine in a unit operation must not lead to a redefinition of a stream. By the encapsulation of numerical data into a static stream definition the exchangeability of both parts is strongly limited.

Our conclusions on this point are quite contrary to Benaouda et al. [1997] and Johns [1997a]. Also in contrast to the latter two references, Vacher et al. [1997] regard a stream as a set of data shared between two unit references. The unit operations must define their own semantics, i.e. how this data is to be interpreted. Besides being computationally very efficient, this leads to a very flexible approach: If it can be ensured, that unit operations rely on the same interpretation, no fix stream types will

have to be defined. Unit developers can agree on a useful set of data to be transmitted, but still the set of data exchanged between unit operations can be enhanced further. However, some intelligence in the executive is required to check if both units can cooperate and perform necessary translations.

The major difference between Benaouda et al. [1997] and Vacher et al. [1997] is that the former uses an explicit definition of streams properties, whereas the latter lets unit operations implicitly define data to exchange.

Johns [1997a] supports the approach of Vacher et al. [1997] in the context of designing a new simulator. He points out that all existing simulators already have defined stream formats. Therefore CAPE-OPEN cannot redefine something that „lies at the heart of an existing simulator“. In his opinion streams do not simply carry information, they have methods associated with them. Johns stated four requirements:

- Stream values should be set and accessed by methods.
- Physical property methods should be associated with phases in material streams.
- There is a need for methods checking which values and properties are available for setting or recovering for each stream.
- There have to be methods to handle inconsistencies between streams without failures.

Our view on streams is similar to that of Vacher et al. [1997]. In the following sections we will try to combine advantages of both approaches and fulfill Johns' requirements by introducing the concepts of ports and connections. Furthermore, the thoughtfulness of these concepts is demonstrated by different examples. The concepts we are suggesting are the result of our research on computer-aided modelling of recent years. Background information can be found in Marquardt et al. [1993], Bogusch and Marquardt [1995], Marquardt [1996], Bogusch et al. [1996], and Marquardt et al. [1997].

14.2. Ports

All material entities (things) in a process are represented by what we call a device (e.g. unit operations, flowsheets, subflowsheets etc.). Devices usually have several inlets and outlets where material flows enter or leave the device. Additionally, energy or information may cross the system to be balanced. We introduce the concept of a port as an abstraction of any location where material, energy or real information (referring to sensors and actors) leaves or enters the device. We conclude that for a proper flowsheet representation, streams should obviously connect ports rather than unit operations. Fluxes and state quantities associated with a port define **what** is being transferred through this port (see Figure 14-1). Ports are always related to „inlets“ and „outlets“ on the real device which is in contrast to Benaouda et al. [1997] where additional abstract information (numeric/dynamic stream) is exchanged via streams.

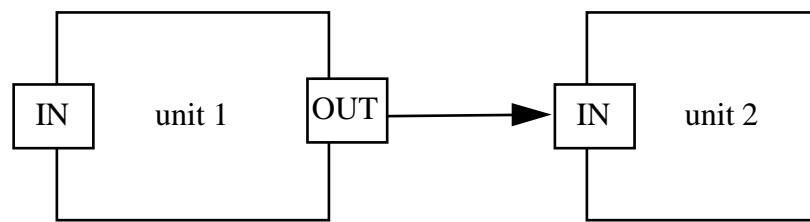


Figure 14-2 Stream connecting two ports

There are different types of ports that can be organised in a specialisation hierarchy in an object-oriented sense (Figure 14-3).

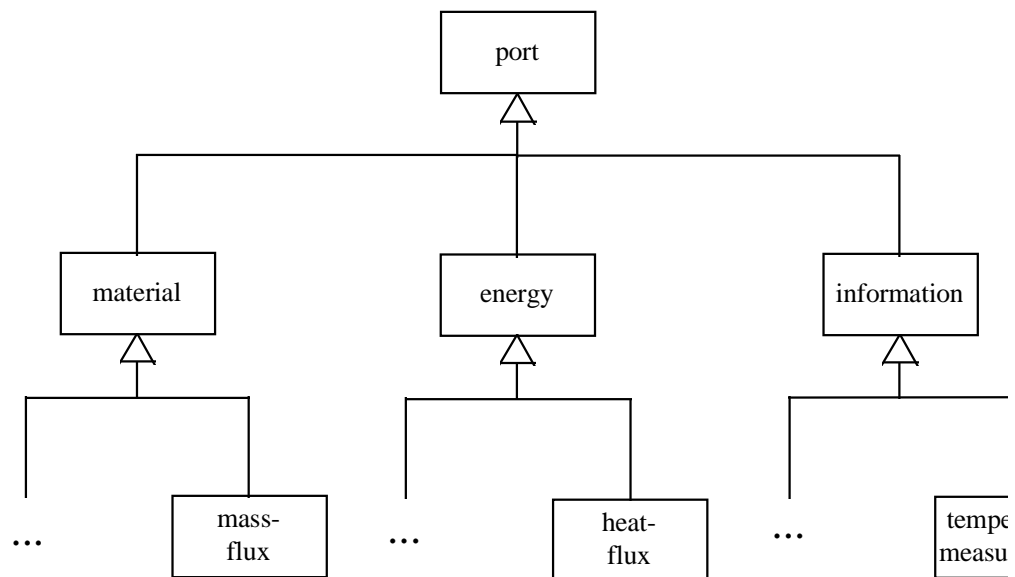


Figure 14-3 Specialisation of the object port

The advantages of such a concept are the achieved flexibility in the outlet and inlet ports of the units and the extensibility of the ports.

Since ports only define a set of information to be transferred to and from the unit operation they can be defined independently of the unit. Hence, we can use ports as a unit operations exterior, its interface, being independent of its interior, the implementation (see chapter 16). This concept is frequently used in data modelling (e.g. Kim [1990])

14.3. Connections

We want to introduce a so-called connection as an object which is responsible for connecting two ports (Figure 14-4). Such a connection refers to the real entity connecting two units, i.e. pipes, signal lines etc. This object shall overcome the limitations of the previously discussed CAPE-OPEN approaches for streams. The connection concept shall be able to comprise these proposals, so existing simulator technologies can be used and new ones can be developed based on a unique concept making an integration of both possible.

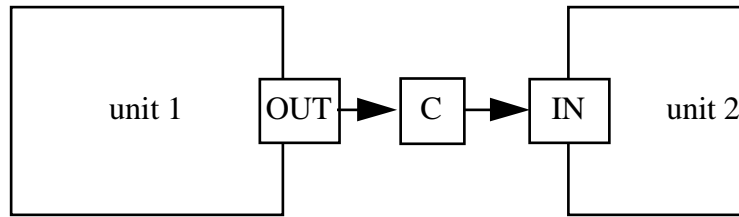


Figure 14-4 Concept of a connection coupling two ports

The motivation for the introduction of a connection concept becomes clear when considering the following example: The flow of saturated vapour through a pipe is accompanied with energy loss. The vapour will partially condensate. No useful abstraction for such phenomena is provided neither by current simulation software nor by what CAPE-OPEN calls streams. Usually, the modeler has to insert an extra unit operation (a flash) which calculates the flash equations. This unit operation is not an abstraction of a real device. This mismatch is even more obvious if the same process model is simulated dynamically. The flash has a volume and therefore shows a hold-up in the dynamic case. Pipes are usually modelled without hold-up. With the introduction of a „pipe“ as a special connection, this behaviour can be modelled easily in the stationary as well as the dynamic case without the aid of a flash. Other problems which may be tackled by a connection are mentioned at the end of this section.

Of course, connections with simpler or even no behaviour exist: The most simple „connection“ transfers the values at the output interface (in a directed information flow, SEM) to the input interface. This connection has no behaviour, it is a neutral element and can thus be called an „identity“ (Figure 14-5). Since no conversion is provided, this connection can connect two ports with identical sets of process quantities and identical units (e.g. SI) only. In this case there is no difference to the approach of Benaouda et al. [1997].

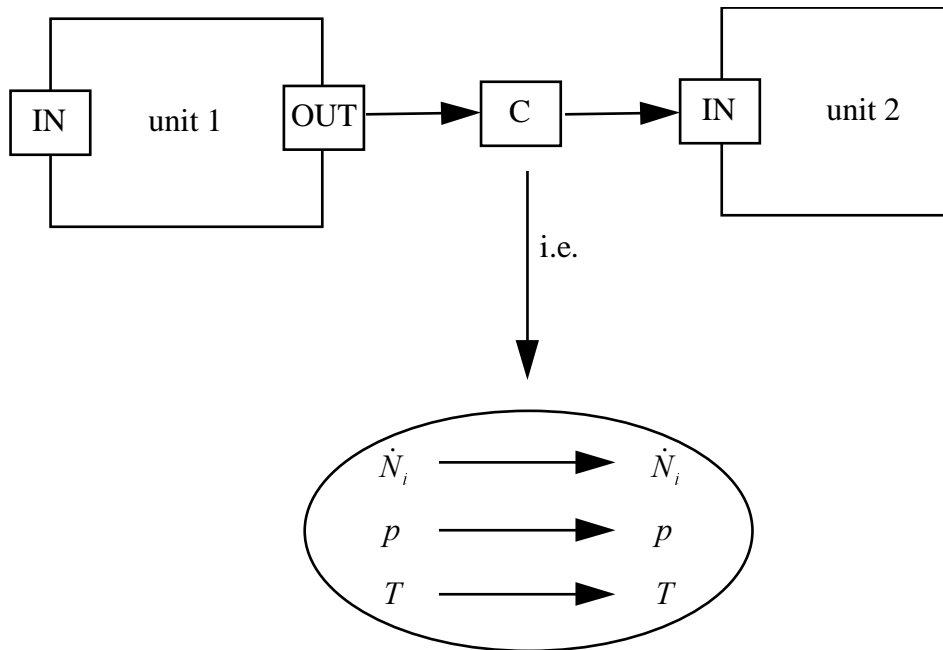


Figure 14-5 The identity as a special case of a connection (C)

If the connection can utilise a certain intelligence it can dynamically determine which process quantities are transferred by ports and establish links between them. This facility of dynamic querying is a major contribution of middleware platforms. Similar intelligence has been demanded by Johns [1997a], when saying that „We cannot assume that any particular pieces of information are available in any stream [...]. CAPE-OPEN access methods must be able to handle such inconsistencies without failures.“ When references to transmitted quantities can be cached, the query must be performed once when establishing the connection between unit operations, not each time, variables have to be propagated from one interface to another. This dynamic approach is in contrast to the static definition of a stream in Benaouda et. al. [1997] and extends the latter with the facility to handle several stream definitions. We now have an object representing the connectivity between devices but we are still able to deal with various stream definitions. However, the process quantities still have to be the same on both sides for using an identity connection.

If the connection can recognise, that two ports are using the same set of process quantities (e.g. easily achieved by a unique identifier for each set of quantities) it can unify the process quantities (i.e. the references to its data storage) to obtain the approach of Vacher et al. [1997] (see Figure 14-6).

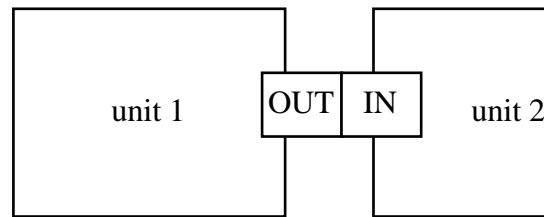


Figure 14-6 Plugged units

The connection is obsolete then, but we propose to model it in this case, too, to have a homogeneous, transparent representation of the flowsheet's topology.

Further problems which can easily be overcome by a connection are:

- The conversion of units (SI, ...) between two unit operations (unit converter).
- The conversion between (\dot{N}_i, p, T) and (\dot{N}_i, p, H) . For several unit operations it is quite simple to output the enthalpy flow instead of the temperature, e.g. a mixer or a splitter. With some means of intelligence in a connection, unit operations can calculate preferred process quantities and rely on the executive to ensure the compatibility on the flowsheet level (process quantities converter).

Like the object port the object connection can be organised hierarchically (Figure 14-7).

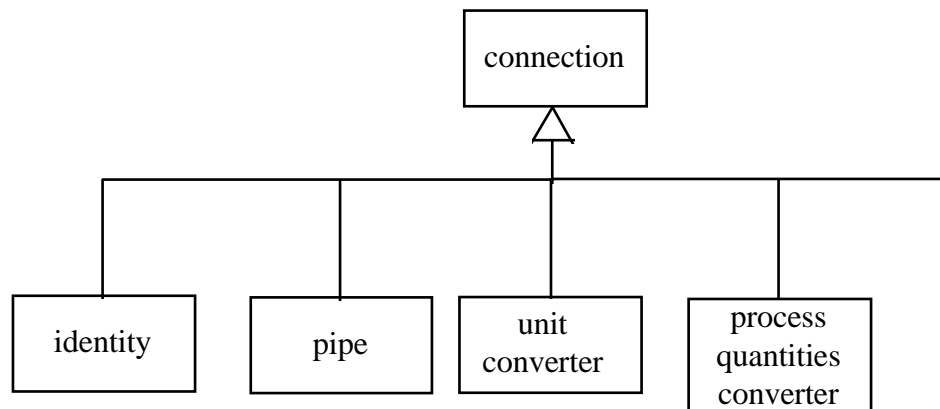


Figure 14-7 Specialisation of the object connection

With these concepts of „ports“ and „connections“ the stream definition of Johns [1997a] can be regarded as just one specialisation in our proposed hierarchy. Furthermore, his requirement list can be handled by this structure:

- In such an object-oriented approach values are set and accessed by methods.
- Physical property methods are associated with the phases in the port (see next chapter).
- The connection is responsible for checking the available values and properties and handling inconsistencies.

Summarising it can be stated that this approach comprises existing simulator technology but leaves the door open for further simulator developments. The

vendors should clearly state whether and at what expense these concepts could be implemented in their systems, if the consortium should agree on a conceptual level.

15. Thermo

15.1. Overview on Existing Documents

The association of thermodynamic properties with a stream has been mentioned in several CAPE-OPEN documents. Our examples emphasise the need for such an association and show how it can be easily achieved if material concepts are modelled in an object-oriented way (like ports and connections). This leads to a set of well-encapsulated and thus interchangeable concepts, that may, according to the idea of object-oriented modelling, be specialised to fit further needs of process simulation.

In general, we adopt the object-oriented approach of Joulia and Jourda [1997], but we propose some modifications. A similar model of an object-oriented representation of material concepts developed during previous work of the authors can be found in Marquardt et al. [1997]. Drewitz [1997] gives a comprehensive overview on thermodynamic properties, grouped in pure component properties (constant and model dependent) and mixture properties. They will be associated with object-oriented concepts in this paper. This was already proposed in Joulia and Jourda [1997], where „entities“ represent a material element which has a set of associated physical and/or thermodynamic properties and can therefore „be used as a property server“. The approach introduces some physically meaningful layers of abstraction into the set of properties functions from Drewitz [1997]. Such a model is useful because it provides several concepts that can be accessed in a general manner from units and streams to serve properties calculations. We want to elaborate these ideas with the example of the calculation of the enthalpy of a phase. Finally, we show how the association of thermodynamic properties with streams as well as with unit operations can be modelled in an elegant manner with a physical understanding of units and ports. We will frequently refer to the comments by Johns [1997b,d] on the above papers.

15.2. Material Concept

We support the object-oriented view on material concepts (Joulia and Jourda [1997]). However, we would like to make some modifications and further enhancements:

- We do not agree with phase and system **being** a mixture. One should rather say that any material entity (multi-phase-system) may be **composed** of several phases (single-phase-systems) which may in turn be composed of a mixture or a pure component (see Figure 15-1). We distinguish discrete mixtures (composed of several pure components) and continuous mixtures (consisting of a large number of ill-defined components), whose properties are defined by means of continuous functions. For continuous mixtures we need to refer to the basis in which to expand the continuous distribution in addition to the weights of the basis functions which in the most simple cases correspond to the concentrations of pseudo-components (von Watzdorf and Marquardt [1997]).

With this distinction, one can overcome the liquid-liquid extraction problem given in Johns [1997d] by an elegant generalisation.

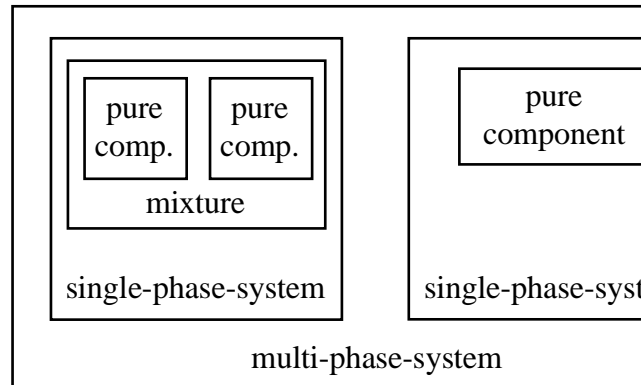


Figure 15-1 Structure of a multi-phase-system

- Do not model the concepts „substance“ and „cluster“. They are strongly influenced by implementation details (database connection for substance and indexing for cluster). These problems should be hidden in component and mixture when dealing with the „what“ aspect, as was intended in Joulia and Jourda [1997]. If the access mechanisms are not exposed one can freely exchange them or use heterogeneous mechanisms. This way, e.g. several databases can operate together transparently to units and streams. The approach allows even an existing physical property package to be wrapped. Similarly, component and pure component need not be distinguished on a class level since the thermodynamic context, specified by the actual aggregation, makes clear, how the concept is to be used. A mixture will use different properties of a pure component (e.g. UNIQUAC parameters and specific heat capacity to employ mixing rules) than a single phase system, which can use the pure components enthalpy calculation directly. We can take further advantage of this approach without introducing new classes when considering a single-phase, which can be either handled as continuous or as a particulate phase by the multiphase-system it occurs in.
- According to Joulia and Jourda [1997] we want to model properties as object attributes and methods. Pure component's constant properties can be seen as attributes since they need not be computed. Model dependent and mixture properties shall be modeled as a method where necessary parameters (e.g. the thermodynamic state) can be passed when the method is called.
- Do not associate a thermodynamic state (pressure, temperature and composition) with the material concepts. When material concepts are used as property servers, the thermodynamic state shall be supplied with the request for calculation (i.e. as an argument). If material concepts encapsulate a thermodynamic state as a set of attributes, they can not be referenced from different locations within the flowsheet. Each occurrence of a thermodynamic state would have to reference a material concept. We propose to use as few instances of material as possible and to reference them from several units and streams to achieve a greater consistency of property calculations among the flowsheet because the thermodynamic model, extrapolation methods etc. used will be the same at least for a given thermodynamic state.
- It is proposed to associate material concepts with a unit operation's implementation (see chapter 16) for the implementation of a unit) and ports each (Figure 15-2). The implementation will reference a material concept abstracting the device's holdup (at least in a physical sense as there is no

holdup represented in a mathematical model in the case of steady-state simulation). For the flash example this will be a multiphase system consisting of a liquid and a vapour phase. Each port will reference a phase system abstracting the material flowing through it. A port abstracting energy flow needs a reference to a material to access property values for the calculation of heat transfer coefficients via Prandtl's and Reynold's numbers etc. The inlet port of the flash references the same two phase systems present in the flash. Liquid and vapour outlet reference the LIQUID and VAPOUR phase, respectively. There need not be such an association for a port transferring signals. As the connection defined above references connected ports this is an elegant way to associate thermodynamic properties with what is called „stream“ in CAPE-OPEN. Recall that this reference is not established directly to a simulator executive or a connection. This makes connections and the simulator executive independent of a specific choice of a properties package. This point is discussed in more detail below.

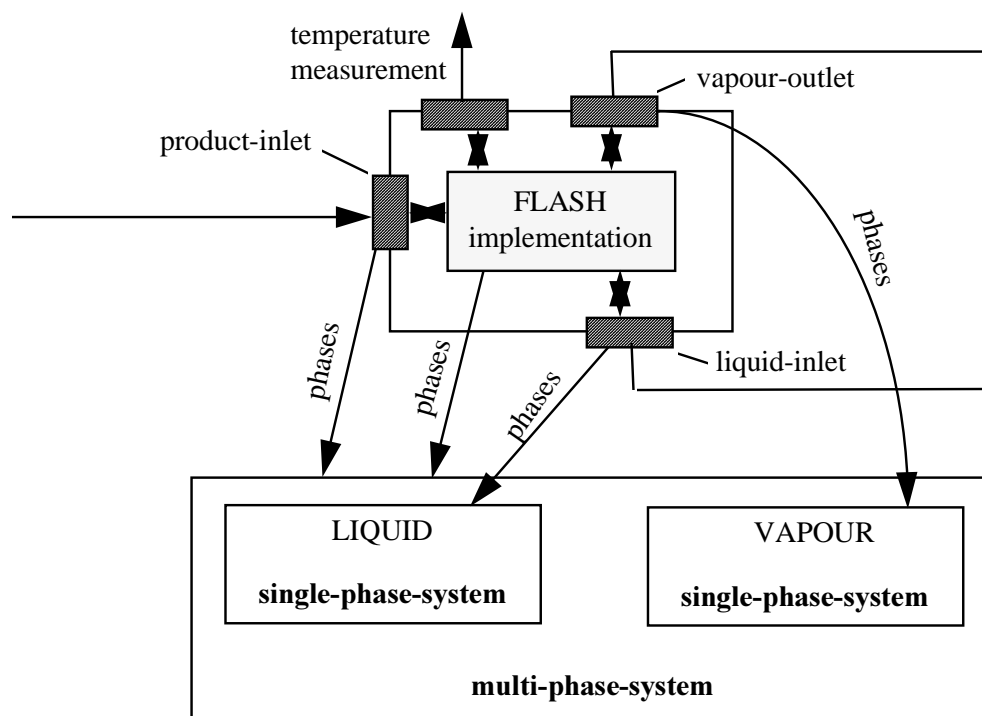


Figure 15-2 Association of the material concept with a unit (flash)

- The object-oriented modelling of material concepts provides a set of well-encapsulated and thus interchangeable components making it easy to associate property calculations with units and streams. Furthermore, the proposed model of a thermo package is used as a wrapper for a (legacy) properties packages or properties database (which is still possible). One can include information for the correct calculation methods and data items in the material concepts themselves as object attributes, which can be configured during startup or through a graphical user interface

15.2.1. Example: Computation of phase enthalpy

Phase enthalpy can be associated with a single-phase-system, called „phase“ in Joulia and Jourda [1997]. A single-phase-system can be seen as a property server for the enthalpy calculation. The phase „knows“ its aggregate state and can select the

proper method to request enthalpy for liquid, vapour or solid state (EnthalpyLiquid, EnthalpyVapor, EnthalpySolid for example) from its constituent, a mixture (Figure 15-3) or a pure component (Figure 15-4).

A mixture would have to calculate partial molar enthalpies based on activity coefficients. For the calculation of the activity coefficients a model for the free Gibb's enthalpy can be employed (e.g. a UNIQUAC model) and request properties of the constituents of the mixture, its pure components. Requesting pure component properties may for example result in a database lookup as already mentioned in Joulia and Jourda [1997]. The chaining of requests through the aggregation hierarchy of the material concepts is shown in Figure 15-3 and Figure 15-4.

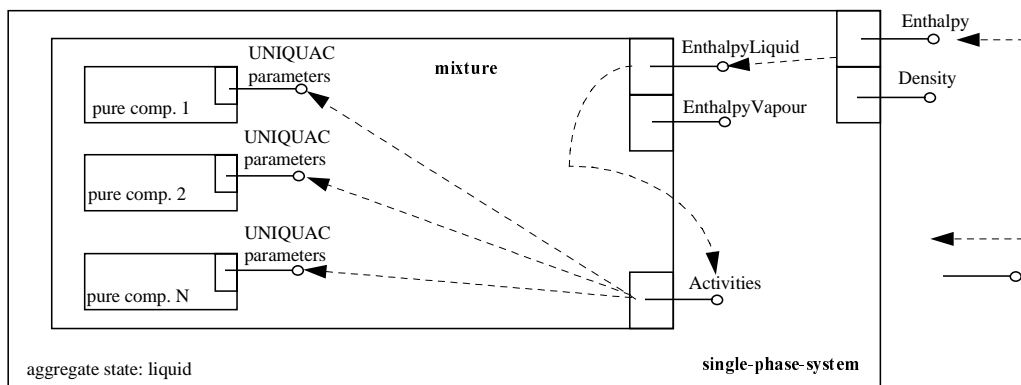


Figure 15-3 Chaining of requests in a mixture

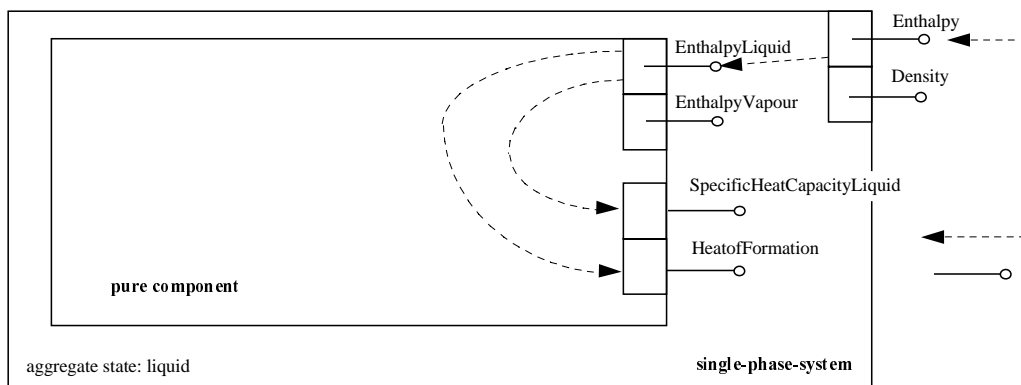


Figure 15-4 Chaining of requests in a pure component

A pure component can directly integrate its specific heat capacity and request its heat of formation to respond to the request for its enthalpy.

Note that the mechanism of enthalpy computation in mixture and pure component is transparent to the phase and can therefore be exchanged. The approach can also handle heterogeneous data storage mechanisms for pure component properties, because their access is hidden in the innermost concept, the pure substance.

15.3. Association of Thermodynamic Concepts with Units and Streams

We propose to associate each unit operation and each port with a suitable material concept introduced above. A flash (as shown in Figure 15-2) will reference a multi-phase-system that is composed of a liquid and a vapour phase. The outlet ports of the flash, LIQUID-OUT and VAPOUR-OUT will reference a single-phase-system each, LIQUID and VAPOUR, respectively.

Note that

- The unit operation can access phase equilibrium calculations from the multi-phase-system as well as methods associated with the single-phases LIQUID and VAPOR for the calculation of phase dependent properties (enthalpy, density, ...).
- Connections do no longer need an explicit association with a thermo package since they can now refer to physical properties via the ports they connect. That makes it possible to exchange (without reconfiguring the executive or connections) the unit operations implementation, the unit operation itself or the entire thermo package (which was required in Johns [1997b]) since every unit refers to ports and an implementation which refer to a set of thermodynamic concepts. These thermodynamic concepts are not referred to by connections or the executive. This is clearly in contrast to Johns [1997a], where it is „suggested that cape-open standards are based on the convention that all streams have methods associated with them that allow physical properties to be computed for any phase of any stream“. The latter approach however leads to a serious violation of encapsulation which makes it hard to exchange units or the thermo package and to extend the latter, since each stream definition would have to be extended with added property methods.
- Further comments on Johns [1997b]: Through the encapsulation of concepts on each aggregation level it is guaranteed that unit operations and streams cannot call methods to calculate properties (such as Antoine or Polynomial) on the conceptual level. The unit operation calls only methods for meaningful physical properties on the abstraction layer of single-phase-system and multi-phase-system. Any calculation method for the phase-systems or any concept aggregated within them can be exchanged in these constituting concepts (mixture, component) without changing the unit's code. If the extrapolation method is increased in accuracy (which can usually be done through inheritance in an object-oriented approach), the unit operation will use the new method without a need to be reconfigured or even recompiled.
- The object-oriented approach is not solely a step towards database and database modelling activities as claimed in Johns [1997b]. It is a common approach for both, modelling on the conceptual and the data layer. It will be easier to integrate physical properties and simulation databases if agreement on similar concepts for all these activities can be achieved. The same yields for a neutral file format (PDXI).
- The enhancement of a properties package with a new method is clearly vital for an open simulation system, but it is easy in an object-oriented approach. Adding new methods for existing properties is essentially the same and will not be discussed separately. In the following, we describe the process for adding a new property. Exchanging the computation method for an existing property is performed by giving the subroutine the same name, this technique is known as

„overwriting“ a method. In our approach for modelling a thermodynamic package a new method can be added (or an existing overwritten) by the following steps:

- Subclassing a suitable material concept, usually by means of an interface definition language.
 - Properties data will be read from the material concept itself, which should hide its connections to material databases etc.
The new routine can access existing property methods of the enhanced material concept (similar to the EnthalpyLiquid using the HeatofFormation in Figure 15-4) or request data items from an associated database.
 - An association with streams is not needed, as stated above.
 - Test routines to check the availability are usually not necessary, since most or even all middleware platforms provide dynamic querying of interfaces, which should be employed to check compatibility of unit and thermo package before starting to simulate.
 - The unit operation's implementation and the port's directly reference the new property, it is not searched for at every call.
- Further comments on Johns [1997d]: The object-oriented approach is not „of less application in defining standard interfaces between exchangeable components“. As shown above it allows making components interchangeable by providing a standard interface on the level of generally accessing material concepts from units and connections but also from a database/data modelling perspective.
 - Our approach can be extended to model the examples in Johns [1997d] as follows: The liquid-liquid extraction example can be modeled with the continuous mixture introduced above, which can provide a means to represent statistically distributed properties. For the hydrocyclone it will be necessary to extend the properties vector to include settling velocities as a continuous (or discrete) function of particle sizes. For the froth flotation cell, one could introduce surface specific rather than bulk specific quantities. The waste combustion models can be modeled by not introducing molar fractions as usual but fractions in terms of individual atomic species.

16. Units

In this section we present our view of a unit and how this unit operation can be related to other concepts presented in this document and to existing CAPE-OPEN documents (BP [1997], Pantelides [1997])

A unit is a model of a chemical device. As already said above, ports model inlets and outlets of the real device and define a set of process quantities transmitted through the port. A port is part of the exterior of a unit, so these quantities may be called „public“. A connection for example can pick them up and transfer them to another port. After having defined the exterior with the help of set of ports, we now want to define a unit operation's interior, which we call its implementation. The implementation references a set of ports being present at its unit operation. The reverse is not true: There are no links from a port to a certain implementation. Hence the ports are independent of a special implementation and we can exchange

the implementation inside a unit operation without notifying the executive or adjacent connections.

One can make use of this fact by specifying alternative implementations for a certain unit operation. These implementations can vary in level of detail, they can differ in the simulation strategy used (SEM, EO, ...) or even enclose a whole flowsheet. The hierarchical decomposition of a system into its parts is a basic approach of systems theory to reduce a system's complexity (Bunge [1977], [1979]).

Flowsheets that are employed as an implementation of a unit can be called subflowsheets and the corresponding unit is called composite - it is composed of the unit operations and connections on its subflowsheet. The separation section of a plant can be refined on a subflowsheet with several columns. Each column may in turn be refined into a condenser, a reboiler, its trays etc. Unit operations which are not hierarchically decomposed on a subflowsheet are called elementary (see Figure 16-1). Their implementations are described with a set of process quantities and a mathematical model describing the units behaviour.

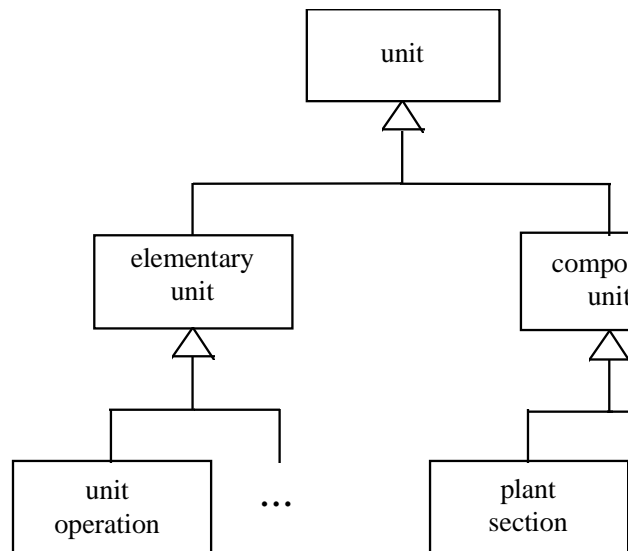


Figure 16-1 Specialisation of the object unit

On all hierarchical levels, we use special types of unit operations to denote feed and product streams. Feeds have output ports whereas products use input ports only. This way only two types of elements have to be distinguished in a flowsheet: units and connections.

16.1. Elementary Units

An elementary unit needs to know about its equation system, a set of unknowns, a set of design parameters, related ports and an associated properties package (see chapter 15 for an elegant solution on this issue). Design parameters refer to quantities such as geometrical size, number of trays e.g. which is usually supplied as input by the (in contrast to design or optimisation calculations). The set of design parameters is not fixed, but can vary from simulation to simulation, depending on the task the model is created for.

In a modular approach, the equation system should be able to compute (with the help of a numerics package inside or outside the unit) the unknowns with the help of the design quantities and the values from the input ports. Values of the output ports will coincide with several of the unknowns. We now want to relate our view on elementary units with the proposals of BP [1997] and Pantelides [1997]. First of all our comments on BP [1997]:

- BP does not take into account the association of the thermodynamic and numerical packages within the unit operations. An integrated view on these clearly strongly dependent issues can help to build. Chapter 15 shows, how a possible relation between a unit and a thermo package can be achieved easily by a holistic view on the underlying concepts.
- The postulation of a unit's independence of the flowsheet topology is supported. No extra work has to be done to achieve this, it is ensured by the encapsulation of a unit operation with its ports. A reference from a port is not necessary (only the reverse is true) and so a unit cannot „see“ beyond its ports.

The comments on Pantelides [1997] in relation to our view on units are:

- Identifying input and output connections is easy by exposing references to the port upon request (Pantelides [1997]: 3.2.2, 3.2.3)
- Variables transferred by a port (these refer to connection variables in Pantelides [1997]) should be better handled by the port itself instead of dealing with them on the unit operation layer (Pantelides [1997]: 3.2.6, 3.2.9). This will lead to a better encapsulation of the port.
- For performance issues, unit operation's implementations should directly access variables, those within the implementation as well as those of ports. The „ComputeSObject“ (Pantelides [1997]: 3.2.8) can access all variables in natural way.

Our structural unit approach has the following advantages:

- Modelling variables as process quantity objects can simplify their handling and lead to an elegant architectural model, especially in conjunction with the ports and connection objects.
- Obtaining information about process quantities can be modeled in an easy manner using object attribute and methods. Upper/lower bounds and current value can be modeled as object attributes. These can be extended in the future to include typical values for initialization or dimensions and units (SI, ...) for consistency checking and conversion.
- If two ports use identical process quantities, it is possible to reference only one variable for both ports. This way, unit operations can transparently share process quantities as proposed in Vacher et al. [1997] without violating the independence of the flowsheet.
- Some of the operations from Pantelides [1997] should be associated with the implementation rather than the unit object itself, e.g. a composite implementation cannot refer to internal process quantities, but only to the entailed unit operations and connections instead.

16.2. Composite Units

A composite unit operation can be refined into a subflowsheet (see Figure 16-2). If special types of unit operations, sinks and sources, are used (see above) there is an elegant way to propagate the input quantities of the composite unit's input ports to the source on the subflowsheet by unifying the corresponding process quantities. The output ports and sinks can be handled analogue.

This is similar to the approach of Vacher et al. [1997]. The propagation of process quantities on one hierarchic flowsheet level may be called „vertical“, the propagation of the variables from input/output interfaces of a composite units to the corresponding sinks and sources may be called „horizontal“.

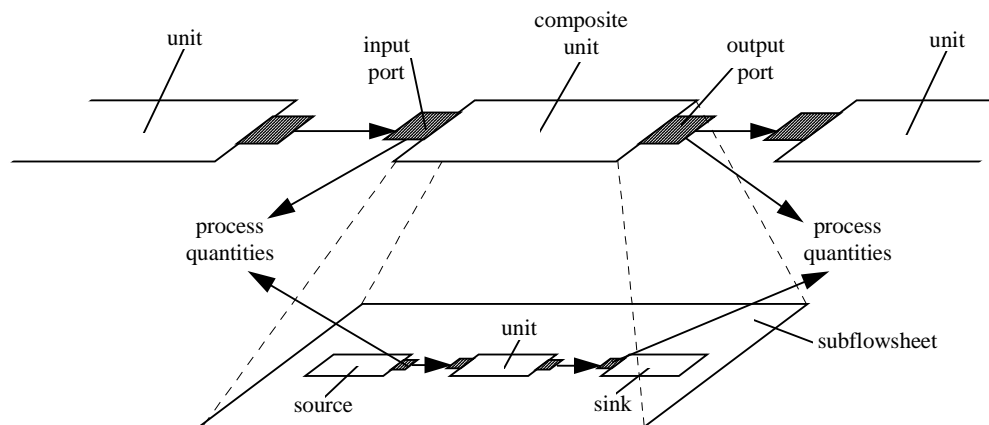


Figure 16-2 Composite unit with source and sink

The composite implementation can employ a solution strategy of its own choice to compute its output values from the input values by simulating its subflowsheet.

With these conceptual concepts it may even be possible to combine EO- and SM-representations in one single unit object (see Figure 16-3, presented at the Paris meeting). Like mentioned before alternative implementations can be realised without changing the ports. Furthermore, such a unit should provide methods to convert between the different representations.

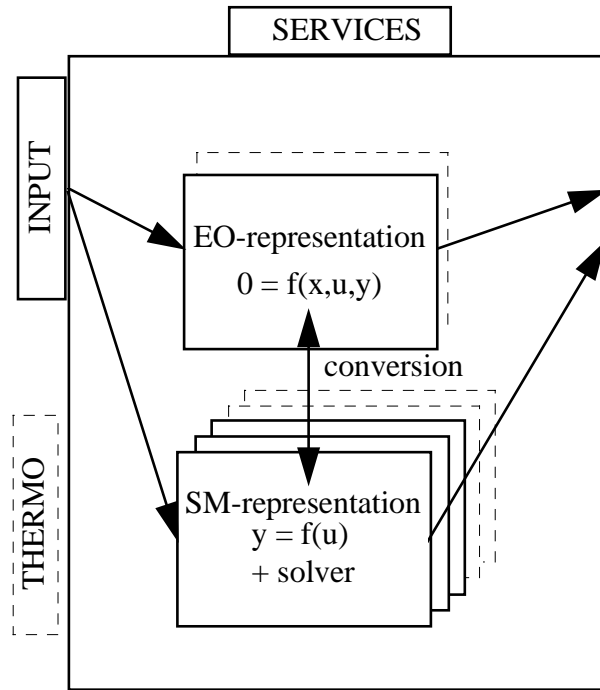


Figure 16-3 Unit combining SM- and EO-representations

17. Summary

In this paper we discussed the interconnections between streams, thermo, and units. In an object-oriented approach we distinguished four different type of objects:

- port
- connection
- substance
- unit

With the concept of ports the exterior of a unit is described by these ports; the interior of a unit by its implementation. A unit references their ports.

Instead of defining a single stream standard we introduced the object „connection“ which can have different levels of complexity and intelligence. The connection references the ports, which it is coupling.

Property packages can easily be associated if a unit or a port references our substance concept having its own calculation methods.

In Figure 17-1 these concepts are illustrated by showing the specifications and interconnections of the proposed objects (cf. Figure 13-1). With such a concept on a conceptual level we minimise the risk of creating inconsistencies. Furthermore we ensure that CAPE-OPEN stays open for further developments.

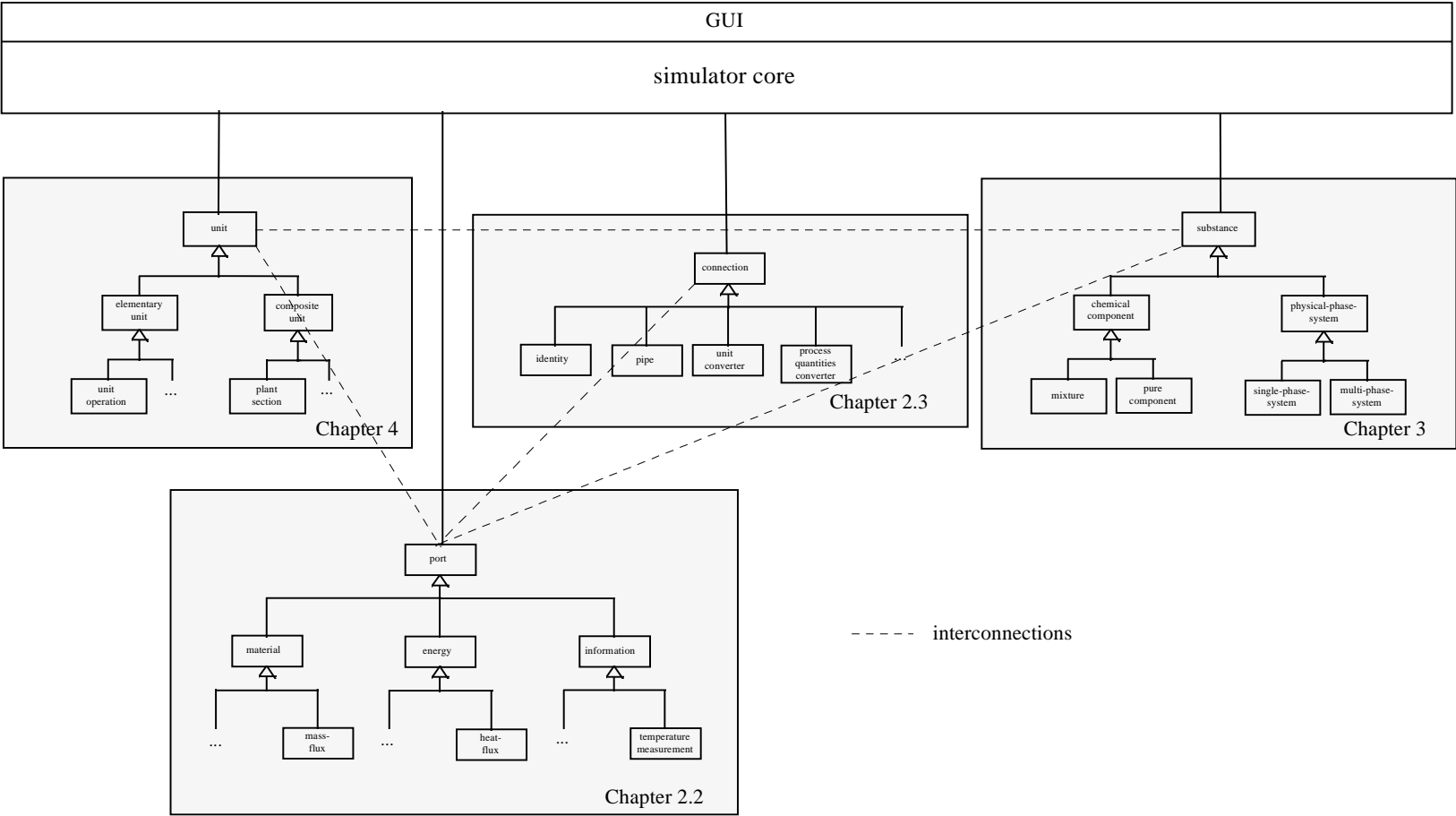


Figure 17-1 Structural concept

18. References

- Benaouda K, Esposito R, Belaud J-P and Joulia X [1997]. A Proposal for Streams. CAPE-OPEN Paper. CO-NUMR-EL-02.
- Bogusch R and Marquardt, W [1995]^{*}. A Formal Representation of Process Model Equations. *Comput. Chem. Engg.* 19 Suppl., S211-S216.
- Bogusch R, Lohmann, B and Marquardt, W [1996]^{*}. Computer-Aided Process Modeling with MODKIT. *Proceedings Chempusers Europe III*, October 29/30, 1996.
- BP [1997]. Draft Conceptual Requirements for a CAPE-OPEN Unit Operation Interface. CAPE-OPEN Paper.
- Bunge M [1977]. *Treatise on Basic Philosophy. Vol. 3: Ontology I: The Furniture of the World*. D. Riedel. Dordrecht.
- Bunge M [1977]. *Treatise on Basic Philosophy. Vol. 4: Ontology II: A World of Systems*. D. Riedel. Dordrecht.
- Drewitz W [1997]. Proposal for Thermo. CAPE-OPEN Paper. CO-THRM-BA-01.
- Johns W R [1997a]. Comment on „The use of stream“. CAPE-OPEN Paper. CO-CCPT-QS-08.
- Johns W R [1997b]. Comments on „Straw Proposal for WP 2.1“ and „Conceptual Design for Thermo Work Package“. CAPE-OPEN Paper. CO-CCPT-QS-09.
- Johns W R [1997c]. Comments on „Proposal for Thermo“. CAPE-OPEN Paper. CO-CCPT-QS-10.
- Johns W R [1997d]. Comments on „Modelling of Matter“. CAPE-OPEN Paper. CO-CCPT-QS-11.
- Joulia X and Jourda L [1997]. Modelling of Matter. CAPE-OPEN Paper. CO-DSGN-IN-01.
- Kim, W [1990]. *Introduction to Object-Oriented Databases*, MIT Press, Cambridge.
- Marquardt W, Gerstlauer A and Gilles, E D [1993]^{*}. Modeling and Representation of Complex Objects: a Chemical Engineering Perspective. *Proceedings of 6th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, Edinburgh, Scotland, June 1/4, 1993, 219-228.
- Marquardt, W [1996]^{*}. Trends in Computer-Aided Process Modeling. *Chem. Engg.* 20, 591-609.
- Marquardt W, Baumeister M, Bogusch R, Geffers W, Lohmann B, Sattler U and Souza D [1997]. VEDA: Objekt-orientierte Datenmodellierung für die Repräsentation von verfahrenstechnischen Prozeßmodellen. Internal report (in german).

Pantelides, C C [1997]. CAPE-OPEN: Standard Objects for Unit Operations (Draft 2). CAPE-OPEN Paper.

Vacher P, Emond F and De Hemptinne J-C [1997]. The Use of Stream. CAPE-OPEN Paper. CO-DSGN-IP-06.

von Watzdorf R and Marquardt W [1997]^{*}. Fully Adaptive Model Size Reduction for Multicomponent Separation Problems. Contribution to PSE '97/ESCAPE-7, Trondheim, Norway, May 26-29, 1997.

^{*} Abstracts of these papers can be downloaded from
<http://www.lfpt.rwth-aachen.de/Publication/publist.html>