Object-oriented modelling applied to hybrid unit operations

Paolo GREPPI, Barbara BOSIO, Elisabetta ARATO
DICAT – Università di Genova
Via Opera Pia 15 – 16145 Genova, ITALY

Modern programming languages such as C++ overcome the limitations of purely procedural programming, adding modularity, data abstraction, object-orientedness and generic programming. This increased expressivity can handle the diversity of objects, which arise while modelling continuous processes involving hybrid unit operations.

A reformulation of the basic concepts of chemical engineering suitable for translation into C++ classes is presented in this contribution. To increase code reuse across different models the basic unit operations are obtained by combining smaller components called mixins; hybrid or complex unit operations are obtained by composing the basic classes in multistage arrangements and/or by customising them. As practical applications for the proposed approach we present the model of a simple distillation column and the model of a planar fuel cell discretised in two-dimensions.

1. Introduction

Unit operations (Little 1915) were proposed to unify the treatment of different processes in chemical engineering. The spectrum of unit operations has expanded as hybrid unit operations have been proposed in the context of process intensification, and innovative unit operations recombine features found in different domains. Object-oriented modelling has been successfully employed to face the challenge of this diversity, using special-purpose formal languages such as Modelica, SPEEDUP and more recently SCHEMA (McGahey, S. L. et al. 2007) or object-broker-based interfaces between process simulators (CAPE-OPEN 1999). Our proposed approach (Greppi 2006) to chemical process modelling is based on a formal description of both the unit operation models and the system models expressed in the modern general-purpose programming language C++ (ISO 1998) rather than a special-purpose language. To maximise code reuse across different models, thereby reducing the cost and effort involved in developing and maintaining model-based solutions, we have employed the "*Template-based Mixins*" technique (Ducasse, S et al 2006). A mixin is an abstract partial class specification that may be applied to various parent classes to extend them with the same set of features; templated mixins are mixins parameterised by an additional type.

The aim of this paper is to present a reformulation of the basic concepts of chemical engineering in terms of two basic classes: the reactive equilibrium stage and the multi-stream heat exchanger. Furthermore we aim to illustrate how the two basic classes can be customised to represent most of the concentrated parameters hybrid unit operations encounter in practice, or assembled in multi-stage arrangements to represent unit operations with an intrinsic distributed nature.

2. The equilibrium stage

The reactive vapour-liquid stage with unlimited inlets and split phase outlets (for example genflashN_VL in the case of vapour-liquid equilibria) is obtained by assembling a mixin capable of connecting to streams, a core mixin providing the reactive phase equilibrium (genflash) and a phase_split mixin which takes any stream and splits its phases between several outlets. The core mixin genflash is perfectly mixed, with the state and composition of the exiting stream being the same as that inside the stage; therefore to avoid redundant data duplication it is best to have it reference a perfectly mixed, concentrated parameter material stream object providing the phase equilibrium capabilities, which will be described first.

2.1 Phases and streams

We call the intensive variables required to define the state of a single-phase material (i.e. pressure and temperature) the canonical state. A phase object contains extensive properties, such as mass- and mole-based composition, mass- and mole-based flows, both total and for individual components, and mass-specific energy and density properties. A material stream object contains the canonical state and a phase object ("HasA" arrangement) to represent the total phase, and provides methods to solve the phase equilibrium.

Since there are many combinations of stream types according to the number and type of phases, to maximise the code reuse, it is best to create a hierarchy of semi-abstract classes with the data and without the phase equilibrium algorithms:
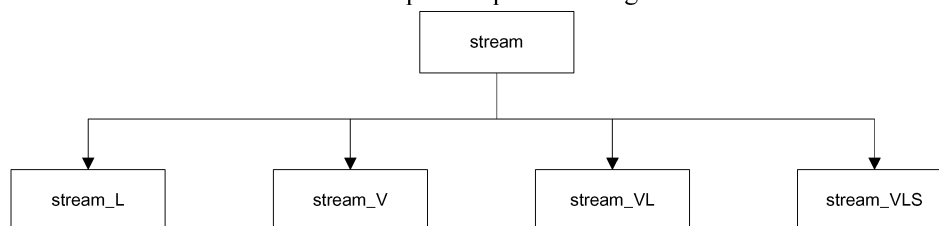

Figure 1 - Hierarchy of semi-abstract classes for stream types

Subsequently the phase equilibrium is added, creating concrete stream classes which can be actually used; for example, using stream_VL (a stream with vapour and one single liquid phase) it is possible to derive the stream_VL_eos<> class (temperature- and composition-dependent K_{VL} obtained from an equation of state for the fluid).

The stream_VL_eos<> class is a generic class, i.e. a template parameterised by the actual equation of state; for example if the equations of state classes rks and pcsaft are available, one can instantiate objects of types stream_VL_eos<rks> and stream_VL_eos<pcsaft>.

Also stream_V_eos<> the vapour-only stream with density and vapour departures computed from the equation of state) or stream_VLS_eos<> and many others need equation of state functionalities; the "*Template-based Mixins*" technique described above is then employed to reuse the glue code between the equation of state and the stream object.

2.2 Reactions
The options for providing the reaction functionality to the equilibrium stage are:
- Derive the equilibrium stage from a reaction class (or a mixin): "IsA" arrangement;
- Include the reaction objects inside the stage: "HasA" arrangement as used for the phase object inside the stream object above;
- Include pointers to dynamically allocated reaction objects inside the stage: "HasA-pointer" arrangement or flexibility via delegation design pattern.

Since the number of reactions is unlimited the first option is not viable; since the type of each reaction can vary and a separate stage type for each combination of reactions is undesirable the second option is not viable either; consequently the chosen approach is the "flexibility via delegation" pattern, and the reactions are the customising classes for genflash.

3. Rate-limited couplings between equilibrium stages

Two or more equilibrium stages can exchange heat or material in a rate-limited fashion; rate-limited phenomena have to be taken into account in the following unit operations:
- Multicomponent diffusion film theory-based non-equilibrium flash;
- HTU/NTU absorption column/stripper;
- Multi-stream reactive heat exchanger;
- Membrane units;
- Concentrated parameter fuel cell models.

Many of these unit operations can be modelled with the multi-stream heat exchanger abstraction illustrated in the next paragraph.

3.1 The multi-stream heat exchanger
The multi-stream heat exchanger is built by assembling ("HasA" arrangement) a number of genflash objects (thereby reusing their support for the reactions and the underlying support for the phase equilibrium coming from the respective exiting streams).
The "HasA-pointer" arrangement is used to support:
- The exchange of thermal energy in a variety of arrangements, defined by the heat exchange connectivity customising class;
- The possibility of mass-transfer or electrochemical reactions, defined by the multiReaction customising class.

The only code left to implement in the actual concrete heat exchanger model is the glue code between the embedded genflash subobjects, the customising heat exchange connectivity and the optional multiReactions.

4. Multistage arrangements

Certain unit operation models encountered in chemical engineering possess an intrinsic distributed nature, i.e. assuming that they are concentrated parameters does not just yield a (more or less acceptable) approximation, it is a plain modelling error.
For example a counter-current liquid-liquid extractor, or a very long pipe (for which the $\Delta P < 10\% \cdot P_{in}$ approximation does not hold) must be considered 1-D distributed objects. A planar fuel cell (see example in paragraph 6.2 below) has to be considered a

2-D distributed object or the maximum temperature in the plane and exit temperatures of the gases will never be matched (Dellepiane et al. 2003).

Multistage units are implemented as flowsheets, appearing in the system model as a flowsheet-in-flowsheet. In this way they carry over the capability of solving in sequential mode (preferred for robustness and useful for initialisation) or in simultaneous mode (preferred for speed).

The 1-D arrangement of units is defined as a multistage class together with:

- The type of the repeating unit operation;
- The number of repeats;
- The number of streams connecting the units (i.e. a traditional distillation column has two, a divided wall column has four);
- The orientation (top-to-bottom or bottom-to-top) for the second and successive streams, given a top-to-bottom orientation for the first stream.

The 2-D arrangement of units is defined as a multistage2D class together with:

- The type of the repeating unit operation;
- The number of repeats along each axis (x and y);
- The number of stream "layers" (i.e. a standard fuel cell has 2, a fuel cell with integrated reformer has three);
- The flow pattern for each stream layer.

5. Customising classes

As we have seen the basic and multistage models can be customised by zero or more customising objects, using the "HasA-pointer" arrangement. The customising classes are reactions, multireactions, heat exchange connectivity and flow patterns.

A **reaction** acts on a phase by altering its composition while keeping the atom and mass balance consistent; it is fully specified by listing the stoichiometric coefficients and the conversion.

A **multireaction** acts on two or more phases by altering their composition while keeping the overall atom and mass balance consistent; it is fully specified by listing the stoichiometric coefficients for all the components for each of the N phases, and the conversion.

The **heat exchange connectivity** customising class determines how the thermal energy is exchanged. in a multi-stream heat exchanger: either pair-wise between the stages without interposition of solid sects (ring-shaped arrangement), via a single central solid sect (star-shaped arrangement), or via a single central solid sect (star-shaped arrangement), or in a combination mode (ring-star i.e. the well-known star-triangle arrangement in the case of three stages).

The **flow pattern** customising objects are used to specify a regular connection pattern in a 2-D arrangement by a symbolic or graphical representation of the pattern rather than by explicitly connecting the discretised elements.

6. Applications

6.1 The simple distillation column

To test the feasibility of the proposed approach, we have run a very simple distillation column model, with eight equilibrium stages. The unit was decomposed into the following parts (see Figure 2 for the resulting process flowsheet):

- Two multistage<genflashN_VL> for the stripping and rectifying sections;
- Three genflashN_VL units for the feed, the reboiler and the condenser;
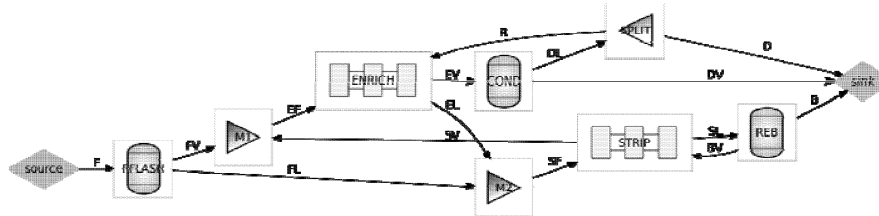- Two mixers;
- A splitter for the distillate draw-off.

Figure 2 -Process flowsheet for the simple distillation column model

The model can be solved in sequential mode with a prohibitive 265 iterations in about 1200 s. The simultaneous resolution takes two sequential iterations for initialisation followed by 7 simultaneous for an overall 12 s. This timing obtained with automatic differentiation and a generic dense Newton solver is about an order of magnitude slower than state-of-the art commercial process simulation tools based on manually coded sparse derivatives and the special purpose inside-out algorithm.

6.2 Planar Fuel Cell

The proposed approach has been applied to planar fuel cell macromodelling based on the past (Bosio et al. 2002) and present research activities of our group in the field of fuel cell modelling .

A concentrated parameters fuel cell model can be implemented in C++ as a multi-stream heat exchanger with electrochemical reactions. A distributed parameters fuel cell can be built using the multistage2D general-purpose bidimensional model. For example the (simplified) code to instantiate a 9x6 2-D discretised fuel cell model, assuming a spiralling pattern for the fuel and a straight pattern for the oxidant (see Figure 3 for the symbolic representation of the flow patterns), would be:

```
multistage2D<multihx> test(9,6);
test.flowpatternType[0] = "spiral96right";
test.flowpatternType[1] = "upFlowpattern<9,6>"
```
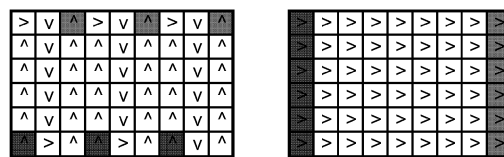

Figure 3 - Fuel (left) and oxidant (right) flow patterns for the fuel cell example

We have benchmarked the timing for a 6x12 discretised fuel cell in cross-flow solved with the approach we present here with the timing for the same discretisation using a special purpose 2-D finite differences code; the latter was 1.5 orders of magnitude faster while still providing more accurate results (accounting for a pressure-driven flow profile in the channel, including additional mid-cell discretisation points for the solid temperature)

On the other hand, the special purpose code does not currently support spiralling flow patterns, equations of state for the gas properties or internal reforming; all these functionalities could easily be added using the customising capabilities and the flexibility of our approach.

7. Conclusions

Although there is room for improvement, the approach we have illustrated in this paper is currently no match for commercial process simulators in the area of conventional unit operation modelling, or for finite difference/finite elements codes for specific distributed parameters unit operation modelling in terms of simulation run-time.

But simulation run-time is measured in CPU minutes, whereas modelling project time is measured in man-months; CPU time is steadily getting cheaper, while skilled manpower is not. We believe that our approach for the reusable modelling of hybrid and complex unit operations could reduce the overall cost and manpower effort to develop and maintain model-based solutions.

The header files with the complete class structures are available for download from the PERT Group's website (www.diam.unige.it/pert). A compiled version of the LIBPF library with headers and samples can be obtained with an academic license by mailing the author.

8. References

Bosio B., Costamagna P., Arato E., Costa P., 2002, Modelling approach to fuel cell development, in Recent Research Developments in Electrochemistry, 5, ISBN 81 7895-044-8, Transworld Research Network, Trivandrum, Kerala (India)

Dellepiane D., Bosio B., Arato E., 2003, Clean energy from sugarcane waste: feasibility study of an innovative application of bagasse and barbojo, Journal of Power Sources 122 (2003) 47–56

CAPE-OPEN 1999, CAPE-OPEN Open Interface Specifications - Unit Operations

Ducasse S., Nierstrasz O., Schärli N., Wuyts R., Black A.P. 2006, Traits: A mechanism for fine-grained reuse, ACM TOPLAS, Volume 28 , Issue 2 p. 331 - 388

Greppi P., 2006, LIBPF: A Library for Process Flowsheeting in C++, EMSS2006 Barcelona, Spain

ISO 1998, Programming Language C++ ISO/IEC 14882:1998

Little A.D., 1915, Report to the Corporation of M.I.T.

McGahey, S. L., Cameron, I. T. 2007, A multi-model repository with manipulation and analysis tools, Computers and Chemical Engineering, doi:10.1016/j.compchemeng.2006.10.019

Meyers S., 2005, Effective C++, 3rd edition