Technical notes on implementation of CAPE-OPEN material objects

CAPE-OPEN
What is CAPE-OPEN?

The CAPE-OPEN standard is the de facto standard for interfacing process modelling software components for use in the design and operation of chemical processes. It is based on universally recognised software technologies, such as COM and CORBA. The CO standard is open, multi-platform, uniform and available free of charge.

(Note: practical implementations restricted to COM at Windows platforms)
What is CAPE-OPEN?

It is described in a formal documentation set covering areas such as unit operations, physical properties and numerical solvers, (...). In practice, it enables components supplied by third parties, such as niche physical property packages or unit operation models, to be used in “plug and play” mode in commercial process modelling software tools.

(Note: practical implementations limited to physical property packages and unit operations)
What is CAPE-OPEN?

In reality this currently means:

• physical property package implementations
• unit operation implementations
• support for both of these in major simulation engines
• restricted to COM on Windows
• 28 different documents describing only version 1.0
• about 10 of those relevant to v1.0 material objects
• all summarized in one IDL
What is CAPE-OPEN?

- **Simulation environment process space**
  - Overall process model
  - Built-in sub models
  - 3rd party sub models (CAPE-OPEN)
  - User sub models (non CAPE-OPEN)
What is CAPE-OPEN?

Simulation environment process space

- overall process model
  - built-in sub models
  - 3rd party sub models (CAPE-OPEN)
  - user sub models (non CAPE-OPEN)
    - Physical Property Packages
    - Unit Operations
Function of the Material Object

- representation of a material
- connected to a unit operation as material stream
- passed to a property package for calculation inputs and calculation results
Function of the Material Object

- representation of a material
typically store T, P, compositions, flows, phases, properties, state (equilibrium or not, phase existence), ...

- connected to a unit operation as material stream

- passed to a property package for calculation inputs and calculation results
Function of the Material Object

- representation of a material
- connected to a unit operation as material stream
  - inlet and outlet ports connect to ‘streams’
  - material ‘stream’ is Material Object
  - other ‘streams’: Energy / Information
  - unit cannot write data to inlet stream
  - unit can duplicate inlet stream for calculations
  - unit must flash outlet material streams

- passed to a property package for calculation inputs and calculation results
Function of the Material Object

- representation of a material
- connected to a unit operation as material stream
- passed to a property package for calculation inputs and calculation results
  - PME sets properties on MO
  - PME asks property package (PP) to perform calculation
  - PP obtains calculation inputs from MO
  - PP performs calculations
  - PP sets calculation results on MO
Material Object ownership

- Simulation environment (PME) owns MO
- Communication between PMC and MO: CAPE-OPEN
- Communication between PME and MO: native

mat.T = 298.15
Material Object implementation types

- wrapper around existing material representation: Most common; e.g. to interface an proprietary simulation application material stream with CAPE-OPEN.

- material object designed to allow for communication with property package: state is specified, passed to PP, calculation is done, results are retrieved

- Material object that can do both: passed to PP as well as to unit operations. Most versatile.
Material object tasks:

- Expose set of compounds and phases
- Storage of property values
- Delegation of calculation calls to Property Package
- Conversion of basis
- Keep track of state (e.g. whether in equilibrium, which phases are present, etc)
- Keep track of error status of last call
Material object tasks:

- Expose set of compounds and phases
  - compounds: ID, name, CAS, boil temp, MW
  - phases: name, state of aggregation, key compound...
  - all or subset of the compounds and phases exposed by PP
  - some calls may be forwarded to the PP, but getting the list of compounds and phases NOT: the PP will ask the MO
  - so: when loading PP; asks for supported lists. Make selection of sub-sets. Return stored lists when asked.
Material object tasks:

- Storage of property values
  - Nearly all property values are stored at the MO
  - Exceptions:
    > T- and P-dependent properties using v1.1 thermo
    > CalcAndGetLnPhi
    > compound constant values
  - Stored values serve as representation for in- and outlet streams for Unit Operations
  - Stored values serve as inputs and outputs for thermo-dynamic calculations
Material object tasks:

- Delegation of calculation calls to Property Package

  - If a Material Object uses a Property Package, calculation requests must be forwarded to the Property Package

  - If the Material Object does not use a Property Package, the calculations must be done by the PME
Material object tasks:

- **Conversion of basis (I)**
  - Properties have no basis, or basis=“Mole” or “Mass”
  - Temperature and pressure: no basis
  - Fraction, phaseFraction: mole or mass fraction
  - Flow, totalFlow: mol / s or kg / s
  - Other properties: dependent on property:
    - Enthalpy: J/mol or J/kg
    - Viscosity, molecularWeight: no basis

- Derivatives:
  - mole number derivatives should have mole or no basis
Material object tasks:

Conversion of basis (II) conversion of fractions:

mole to mass:

\[
X_{i,\text{mass}} = \frac{X_{i,\text{mole}} MW_i}{\sum_{j=1}^{N_{\text{comps}}} X_{j,\text{mole}} MW_j}
\]

mass to mole:

\[
X_{i,\text{mole}} = \frac{X_{i,\text{mass}} / MW_i}{\sum_{j=1}^{N_{\text{comps}}} X_{j,\text{mass}} / MW_j}
\]
Material object tasks:

- Conversion of basis (III) conversion of phase fractions:

  mole to mass: \[ \Theta_{i,\text{mass}} = \frac{\theta_{i,\text{mole}} PMW_i}{\sum_{j=1}^{N_{\text{phases}}} \theta_{j,\text{mole}} PMW_j} \]

  mass to mole: \[ \Theta_{i,\text{mole}} = \frac{\theta_{i,\text{mass}} / PMW_i}{\sum_{j=1}^{N_{\text{phases}}} \theta_{j,\text{mass}} / PMW_j} \]

With: \[ PMW_i = \sum_{j=1}^{N_{\text{comps}}} X_{j,\text{mole,phase}} MW_j \]
Material object tasks:

- Conversion of basis (IV) conversion of flow:

  **mole to mass:**

  \[ F_{i,\text{mass}} = 10^{-3} F_{i,\text{mole}} MW_i \]

  **mass to mole:**

  \[ F_{i,\text{mole}} = \frac{F_{i,\text{mass}}}{(10^{-3} MW_i)} \]
Material object tasks:

- Conversion of basis (V) conversion of total flow:

  **mole to mass:**

  \[ F_{\text{mass}} = 10^{-3} F_{\text{mole}} MW \]

  **mass to mole:**

  \[ F_{\text{mole}} = \frac{F_{\text{mass}}}{10^{-3} MW} \]

  \[ MW = \sum_{j=1}^{N_{\text{comps}}} X_{j,\text{mole}} MW_j \]
Material object tasks:

- Conversion of basis (VI) other properties
  - conversion depends on property
  - we must know
    - how property depends on mole / mass
    - which MW to use
  - enthalpy: J/mol or J/kg, conversionOrder = -1
  - density: mol/m^3 or kg/m^3, conversionOrder = 1
  - viscosity: conversionOrder = 0
- mole to mass: * MW \(_{\text{conversionOrder}}\)
- mass to mole: * MW \(_{-\text{conversionOrder}}\)
Material object tasks:

- Conversion of basis (VII) **which MW?**
  - version 1.1: always mixture MW
  - version 1.0: compound MWs for many ‘pure’ calculations
    (e.g. ‘pure’ enthalpy)
  - Mole number derivatives: undefined, do not perform conversion, see remarks in v1.1 spec
Material object tasks:

Conversion of basis (VIII)

Property conversions do not apply to properties that are not stored at the Material Object:

- compound constants

- version 1.1:
  - $T / P$ dependent properties
  - CalcAndGetLnPhi
  - special case: GetTPFraction / GetOverallTPFraction

These are always obtained in a fixed basis
Material object tasks:

- Conversion of basis (IX) Implementation
  - Store the property in the basis in which it is set
  - Store the basis in which it is set
  - Perform basis conversions only when getting properties
  - Do not allow invalid basis
Material object tasks:

- Expose set of compounds and phases
- Storage of property values
- Delegation of calculation calls to Property Package
- Conversion of basis
- Keep track of state (e.g. whether in equilibrium, which phases are present, etc)
- Keep track of error status of last call
Material object tasks:

- Keep track of state

  - Is the material in equilibrium?
    - yes, directly after an equilibrium calculation
    - no, in case any property is set

  - Which phases are present?
    - poorly defined in version 1.0
    - equilibrium phases after an equilibrium call
    - any phase for which a property is set
Material object tasks:

- Keep track of error status of last call
  - store the error
  - return an error code
  - caller will ask for error message
Property classes:

- Special properties
- Compound constants
- T- and P-dependent properties
- Single-phase and Overall properties
- Two-phase properties
Special properties:

Special properties must always be supported:

- **pressure and temperature:**
  - version 1.0: single value for material object
  - version 1.1: one value for each present phase + Overall

- **fraction** (composition)

- **phaseFraction**

- **flow** (per compound, not relevant in communication with PP)

- **totalFlow** (not relevant in communication with PP)
Compound constants:

Single value for each compound:

- value can be string or double precision
- value is universal per compound; no storage on MO
- some values are passed with compound list
- fixed basis (no basis conversion)
- example: molecularWeight, criticalTemperature
Pressure and temperature dependent properties:

- only depend on *pressure* or *temperature*
- one value for each compound
- fixed basis (no basis conversion)
- example: vaporPressure, glassTransitionTemperature
Single phase and Overall properties:

- value depends on pressure, temperature, fraction, phase
- value can be scalar, vector (one value for each compound), matrix ($n_{\text{compound}} \times n_{\text{compound}}$), depending on which property and which derivative
- possibly no basis
- possibly mole or mass basis: MO converts
- pure / mixture
- example: enthalpy, density, …
Two-phase properties:

- value depends on two sets of pressure, temperature, fraction, phase

- value can be scalar, vector (one value for each compound), matrix \(( n_{\text{compound}} \times n_{\text{compound}} )\), depending on which property and which derivative

- Composition derivatives: 2 matrices (derivative w.r.t. each phase)

- poorly defined in version 1.0 thermo

- example: kvalues, surfaceTension, …
Property derivatives:

- .Dtemperature
  (applies to T-dependent, single-phase and two-phase props)
- .Dpressure
  (applies to P-dependent, single-phase and two-phase props)
- .DmolFraction
  (applies to single-phase and two-phase props)
- .Dmoles
  (applies to single-phase and two-phase props)
Material object must know things about a property:

- property class
- basis dimension:
  - whether or not to accept a basis
  - how to convert from mole to mass basis
  (e.g. density vs volume)
- possibly: extensive vs. intensive
- property dimension:
  (scalar, vector, matrix)

Application must therefore have lists of property information
(lookup by case independent name: calls for hash table)
Interfaces to implement – I. Common Interfaces

ICapelIdentification

Error common interfaces
Notes on Common interfaces

- Identification: name should be unique. When copying a material, give it a new name

- Error common interfaces: all errors that occur must result in a CAPE-OPEN error code. The MO will implement the appropriate interface for the caller to get the error details
Interfaces to implement – II. Thermo v 1.0
Interfaces to implement – III. Thermo v 1.1

ICapeThermoMaterial
ICapeThermoCompounds
ICapeThermoEquilibriumRoutine
ICapeThermoPhases
ICapeThermoPropertyRoutine
ICapeThermoUniversalConstant
### Setting properties:

<table>
<thead>
<tr>
<th></th>
<th>Version 1.0</th>
<th>Version 1.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compound const.</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>P- and T-dependent</td>
<td>SetProp</td>
<td>N/A</td>
</tr>
<tr>
<td>Special, overall,</td>
<td>SetProp</td>
<td>SetOverallProp SetSinglePhaseProp</td>
</tr>
<tr>
<td>single-phase</td>
<td></td>
<td>SetTwoPhaseProp</td>
</tr>
<tr>
<td>Two-phase</td>
<td>SetProp</td>
<td>SetTwoPhaseProp</td>
</tr>
</tbody>
</table>
## Getting properties:

<table>
<thead>
<tr>
<th></th>
<th>Version 1.0</th>
<th>Version 1.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compound const.</td>
<td>GetCompoundConstant</td>
<td>GetCompoundConstant</td>
</tr>
<tr>
<td>P- and T-dependent</td>
<td>GetProp</td>
<td>GetPDependentProperty</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GetTDependentProperty</td>
</tr>
<tr>
<td>Special, overall,</td>
<td>GetProp</td>
<td>GetOverallProp</td>
</tr>
<tr>
<td>single-phase</td>
<td></td>
<td>GetSinglePhaseProp</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CalcAndGetLnPhi</td>
</tr>
<tr>
<td>Two-phase</td>
<td>GetProp</td>
<td>GetTwoPhaseProp</td>
</tr>
</tbody>
</table>
Calculating properties:

<table>
<thead>
<tr>
<th></th>
<th>Version 1.0</th>
<th>Version 1.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compound const.</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>P- and T-dependent</td>
<td>CalcProp</td>
<td>GetPDependentProperty GetTDependentProperty</td>
</tr>
<tr>
<td>Single-phase</td>
<td>CalcProp</td>
<td>CalcSinglePhaseProp CalcAndGetLnPhi</td>
</tr>
<tr>
<td>Two-phase</td>
<td>CalcProp</td>
<td>CalcTwoPhaseProp</td>
</tr>
</tbody>
</table>
Reference state correction:

- MO can optionally provide reference state correction for enthalpy and entropy

\[ H_{SIM} = H_{PP} - \sum_i x_i H_{ref,i} \]

- Reference values calculated only once
- Direction of conversion depends on who is asking (whether or not property package is asking)
- Store values as they are set, do conversion as they are asked
Compound and phase mapping

- PP compound IDs may differ from simulation compound IDs
- PP phase IDs may differ from simulation phase IDs
- This situation is common when mixing property packages
- MO should map IDs from simulation to PP before calling PP
- MO should map IDs from PP to simulation when called by PP
- MO thus needs to know who is calling a function
Who’s calling me?

- important for reference state correction
- important for compound / phase ID mapping
- may be useful for error reporting
- the one that is calling is the one that currently has control
- sufficient to know whether it is the PP or not, so keep track only of PP having control
- when unit operation is active, if MO is inlet stream, disallow setting properties, calculating properties, calculating equilibria
More advances topics

- choosing implementation platform (native vs .NET)
- efficient property and state storage
- mapping between version 1.0 and 1.1 thermo (e.g. version 1.0 UO calling MO with version 1.1 PP)
- mapping between version 1.1 and 1.0 thermo (e.g. version 1.1 UO calling MO with version 1.0 PP)
- caching of released MOs for re-use at Duplicate
- providing derived properties
- error trapping
Pay special attention to data ownership:

- Creator of data is not necessarily the owner
- The owner must delete the data
- IDL: [in], [in, out], [out, retval]
- Data that you create and own, you can pre-allocate