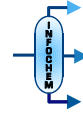# Who Knows Where the Time Goes?
Computational overheads of using the thermodynamics interfaces

Richard Szczepanski, Infochem
Jasper van Baten, AmsterCHEM
Tom Williams, PSE

6th CAPE-OPEN European
Conference
Munich
2-3 April 2009

Objectives

- **Determine computational overhead of using CO thermo interface compared with a native interface**
- **Identify how overhead is divided between different software components**
- **Recommendations for**
  - **PP developers**
  - **PME/application developers**
 - **Future CO specs**
- **Discussion**

It has been suggested that standardised software interfaces such as those defined by CAPE-OPEN will not support computationally efficient links between the components of a process simulation environment.

The most computationally intensive part of most simulations is concerned with evaluating physical properties and doing phase or chemical equilibrium calculations.

In this study we have tried to quantify the computational overheads of using the CAPE-OPEN version 1.0 and 1.1 thermodynamics and physical property interfaces and to investigate how the different software components such as the Material Object and Property Package contribute to the overheads. Finally we make some recommendations for factors that should be considered in the design of software components and for improvements that might usefully be incorporated in future versions of the interface specifications.

Software components

Native

Native PME

Native Physical
Property
System

INFOCHEM
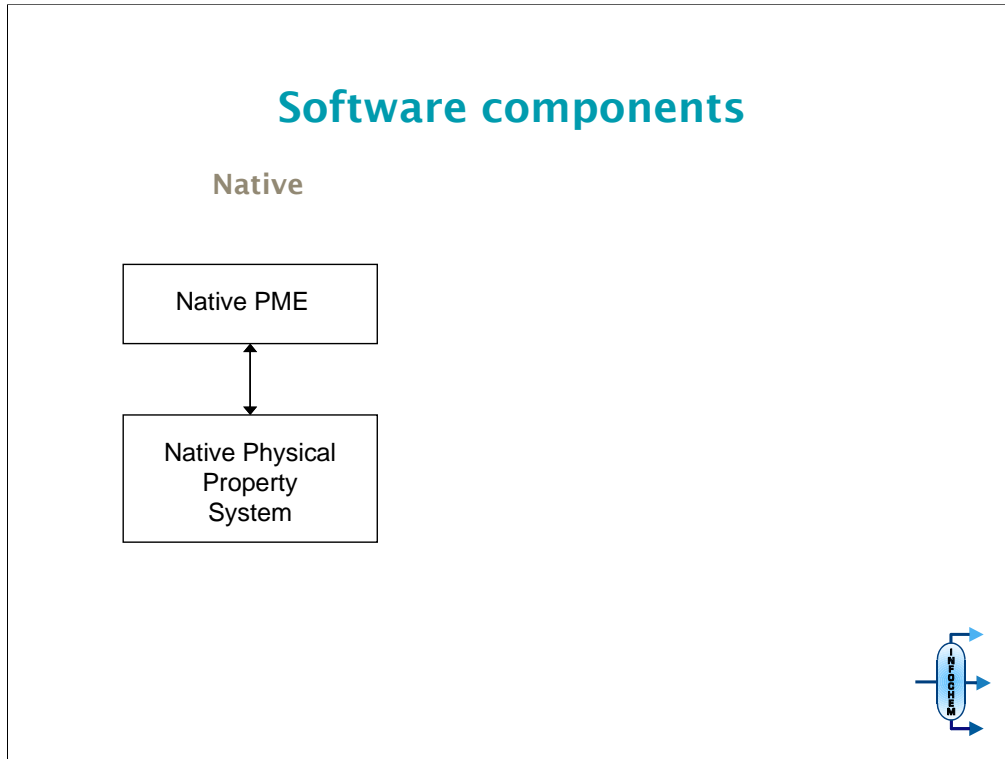
This diagram shows the typical configuration of a Process Modelling Environment (PME) with its own physical property system. Typically the interface between the two will be proprietary and designed to match the way that the PME works. It should be efficient because the PME and property system 'know' each other. Links can be direct and in some older systems the two components would be tightly integrated.

Software components

Native

CO
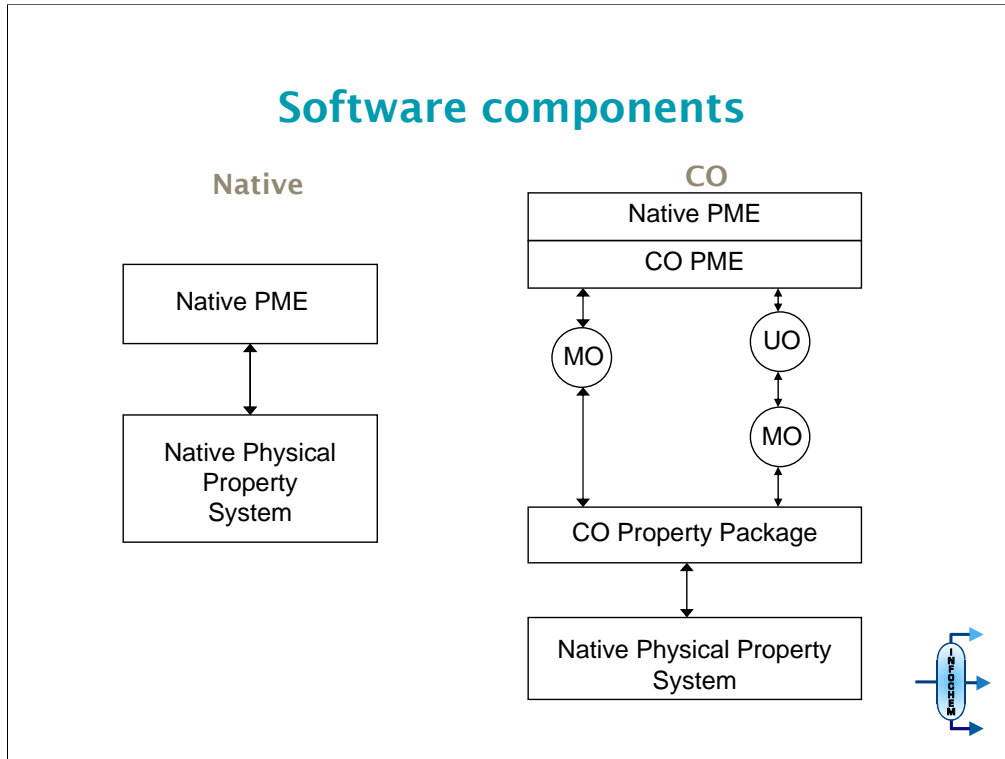
With CO interfaces the PME may have a software layer dedicated to interacting with CO components. The primary object used by CO interfaces is the Material Object (MO) which is a container for properties associated with a material (stream). The MO is passed to the CO Property Package (PP) and provides the input information for physical property calculations. The results are placed in the MO by the PP and are available for use by the PME. A CO unit Operation (UO) will also use the MO to get properties from the PP. Although it is possible to build a PP entirely around the CO interfaces the normal situation is than some native physical property system will provide a software layer that handles the requirements of CO. It is clear from this picture why there may be overheads when using the CO route rather than the native route. There is much more software to traverse and each component will inevitably add some overhead.

Tools Used

- Native physical property system
  - Multiflash 3.8 dll
  - RKS equation of state
  - Equimolar mixtures of 2 to 80 compounds (hydrocarbons)
  - Calculations over grid of P, T points with large number of repetitions
  - CP time reproducibility: 5 – 10%
- Multiflash CO Property Package
  - Implemented in C++
  - Supports CO thermo 1.0 and 1.1
- ThermoWrapper for CO 1.1
  - Library of Fortran-callable routines for using CO interfaces
  - Provides a Material Object implementation
- CO 1.0 Test Application
  - Custom application using CO 1.0 interface and MO
- Matlab CO Thermo Import (AmsterCHEM)
  - Allows a CO 1.1 PP to be imported into Matlab and used to perform physical property calculations

The 'native' property system used in this study was Infochem's Multiflash package (version 3.8) which is delivered as a Win32 dll. All comparisons were made with the Redlich-Kwong-Soave cubic equation of state for equimolar mixtures of hydrocarbons containing between 2 and 80 compounds. Calculations covered a grid of pressures and temperatures between 1 bar and 10 bar and 250K and 450K. Timings for a large number of calculations were accumulated to provide a reasonably large elapsed time and the process was repeated several times to get some indication of the reproducibility which was between 5% and 10%. It should be recognised that in a Windows environment it is difficult to ensure that no other processes are active so the total uncertainty in timing is greater than the scatter in reproducibility.

The Multiflash CO Property Package implements the CO thermo interfaces versions 1.0 and 1.1 and passes requests for calculations to the Multiflash dll. The original PP was implemented in VB but the current version has been rewritten in C++ to improve efficiency and support for CO interfaces. For a comment on the difference in performance between VB and C++ see the conclusions.

Four software combinations were used in this study:

1. Fortran application calling the Multiflash dll directly.

2. Fortran .application using the ThermoWrapper to call the Multiflash PP. The ThermoWrapper is a library of Fortran-callable routines that allows a Fortran application to use a CO version 1.1 PP. In particular it provides a MO implementation. The ThermoWrapper is available from CO-LaN.

3. C++ application calling the Multiflash PP using an existing Thermo 1.0 MO.

4. Matlab calling the Multiflash PP using the CO Thermo Import component from AmsterCHEM. This allows Matlab to use a CO 1.1 PP.

PT Flash & Overall Enthalpy (CO 1.1)

- **Application**
  - **Specify MO to be used:** PP_SetMaterial
  - **Specify list of phases to be considered:** MO_SetPresentPhases
  - **Set overall composition and 2 constraints to define calculation (P and T):** MO_SetOverallProp x 3
  - **Call Property Package:** PP_CalcEquilibrium
- **PP_CalcEquilibrium**
  - **Get calculation conditions:** MO_GetOverallTPFraction
  - **Get list of *possible* phases for calculation:** MO_GetPresentPhases
  - **Do (P,T) flash calculation: call Multiflash dll**
  - **Set list of phases *actually present* at equilibrium:** MO_SetPresentPhases
  - **Set phase compositions,phase fractions, T, P:** MO_SetSinglePhaseProp x 4NP
- **Application**
  - **Get list of phases at equilibrium:** MO_GetPresentPhases
  - **Get phase fraction and composition** MO_GetSinglePhaseProp x 2NP
  - **Calculate phase enthalpy** PP_CalcSinglePhaseProp x NP
  - **Get phase enthalpy** MO_GetSinglePhaseProp x NP
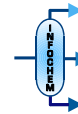
This slide shows the interactions between an application and various CO components when calculating the overall enthalpy of a stream. Essentially we must do a PT flash followed by a loop over the phases present calculating the enthalpy of each one.

The methods shown are the Thermo 1.1 variety but there are mostly similar calls for version 1.0.

The application must first tell the PP which MO it is going to use (PP_SetMaterial). The list of possible phases for the flash is set (MO_SetPresentPhases) and the calculation conditions (MO_SetOverallProp called 3 times). The PP_CalcEquilibrium call carries out the flash.

In the PP the calculation conditions must be recovered from the MO (MO_GetOverallTPFraction) together with the list of possible phases (MO_GetPresentPhases). The flash is done by calling the dll. The results must be stored back in the MO. First the phases present at equilibrium are set (MO_SetPresentPhases) and the temperature, pressure, phase fraction and composition are set for each phase (MO_SetSinglePhaseProp).

Control is returned to the application which must retrieve the calculation results. The enthalpy of each phase is then obtained by calling PP_CalcSinglePhaseProp and the enthalpy value is found by calling MO_GetSinglePhaseProp.

Note that the application has the choice of doing the equivalent operations without using CAPE-OPEN; more on this later.
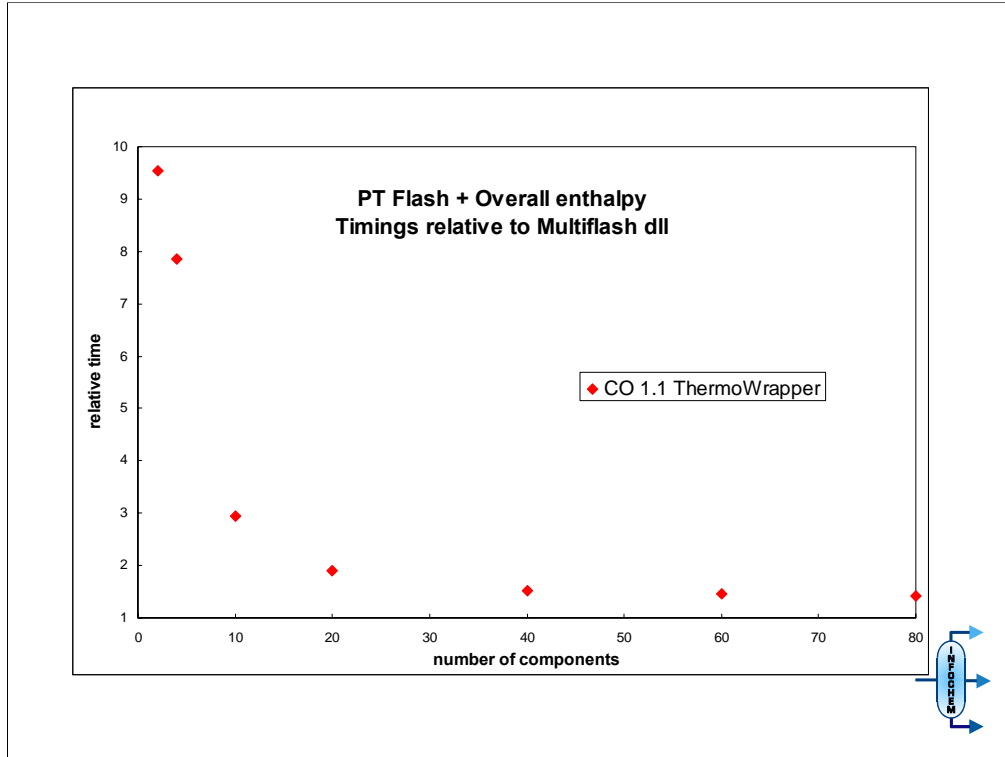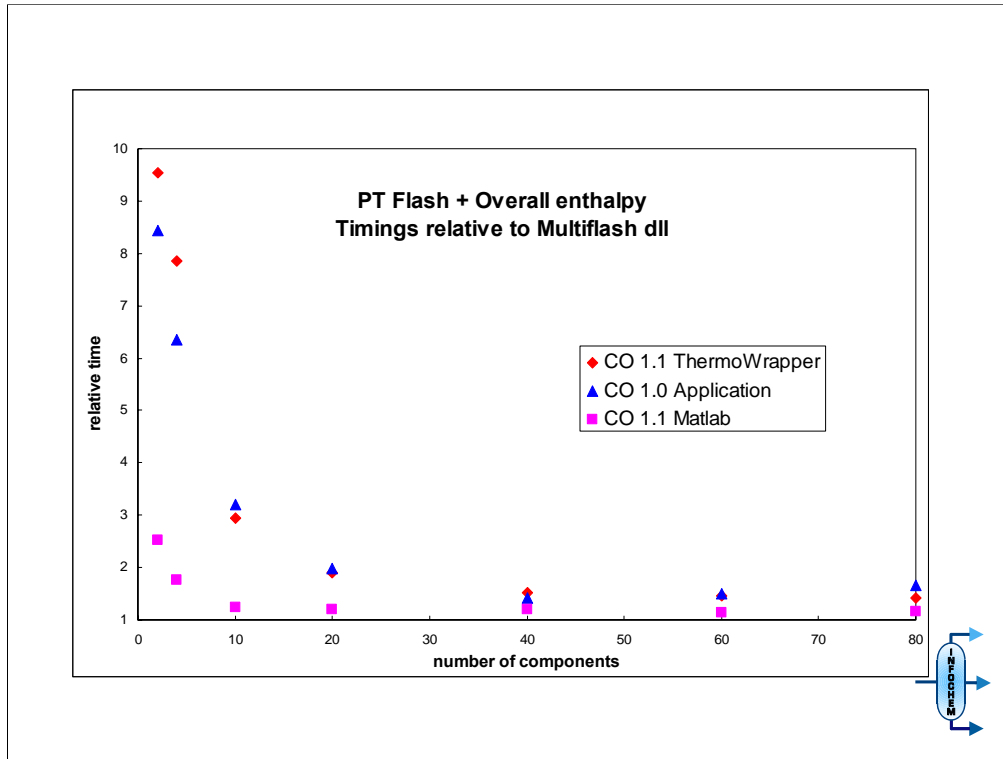
The calculation time for the PT flash and enthalpy calculation just described is shown as a function of the number of components in the mixture. The time is expressed relative to that for doing the equivalent calculation by making direct calls to the Multiflash dll.

These results are for the ThermoWrapper application. The large overhead for a small number of components falls rapidly as the flash and property calculations take up more time for larger mixtures. For 40 components and over the overhead is a factor of about 1.5

PT Flash + Overall enthalpy
Timings relative to Multiflash dll
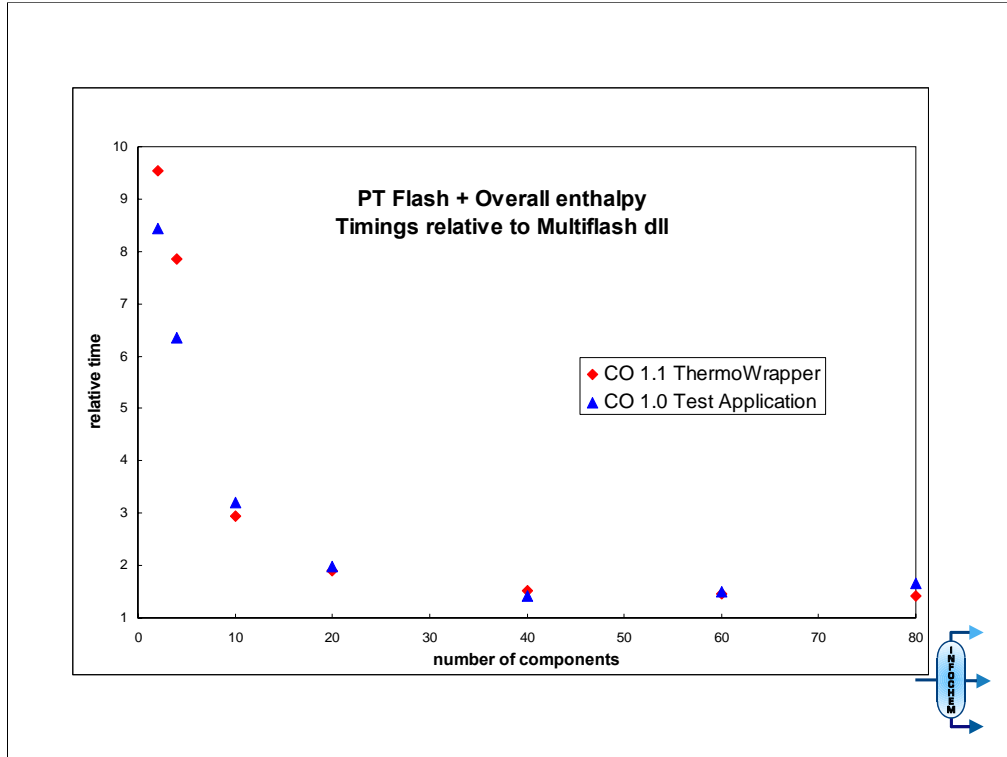
The picture is very similar for the C++ application with the 1.0 MO.

**PT Flash + Overall enthalpy
Timings relative to Multiflash dll**

For the Matlab application there is a much lower overhead. For mixtures of 10 components or more there is an overhead of between 20% and 15%.

Comments

- **Applications**
  - ThermoWrapper: CO 1.1, Fortran, versatile MO
  - Test application: CO 1.0, C++, versatile MO
  - Matlab: CO 1.1, C++, simple MO
- **Differences between 1.0 and 1.1**
  - Analysis of compounds in MO
    - For 1.1 is only done when SetMaterial called
    - For 1.0 must be done on every call for a calculation
  - Getting calculation conditions
    - 1.1 has GetTPFraction and GetOverallTPFraction
  - Fewer arguments in 1.1
    - Set/Get: no compound list
    - Calculate: no calcType or MO
- **Performance**
  - No significant penalty for large number of compounds (>40) whatever the implementation
  - For more complex models overhead will be smaller
  - By appropriate design of MO it is possible to have a reasonable overhead even for small number of compounds

The principal reason for the difference in performance is the type of MO used.

Both the ThermoWrapper application and the C++ application use a versatile MO that is capable of supporting CO unit operations as well as PMEs. Error checking and diagnostics must be provided as well as unit conversions. The MO in the Matlab case is relatively simple and is only used for this application. The operation can be simplified and, most importantly, the application 'owns' the MO and can bypass the Set and Get operations described previously. It uses direct and efficient methods to access the MO data structures.

There are also some significant differences between the Thermo 1.0 and 1.1 specifications that can have an influence on overheads. The SetMaterial in 1.1 method allows the PP to avoid reanalysing the compound list on every call to calculate. The short-cut methods GetTPFraction and GetOverallTPFraction reduce the number of calls to the MO.

It is clearly possible to reduce the overhead in the overall enthalpy calculation to a tolerable level by an appropriate design of the MO. Although the overhead is still significant for a 2-component mixture the calculation is very fast and the overhead is less important. The model used is quite simple and for more complex thermodynamic models the performance will be shifted towards that observed for larger mixtures where the overhead is smaller.

Property Calculation (CalcSinglePhaseProp)

- Application
  - **Set P, T and composition of a phase:** MO_SetSinglePhaseProp x 3
  - **Call to Property package:** PP_CalcSinglePhaseProp
- PP_CalcSinglePhaseProp
  - **Get P, T and composition of phase:** MO_GetTPFraction
  - **Calculate property: call Multiflash dll**
  - **Set property value(s):** MO_SetSinglePhaseProp
- Application
  - **Get property value(s):** MO_GetSinglePhaseProp

The sequence of calls for the calculation of a single-phase property for Thermo 1.1 is shown.

As before there are many Set/Get calls on the MO in addition to the property calculation itself. Note that again the application can choose to do the equivalent without using CAPE-OPEN.

Timings for log Fugacity Coefficient Calculation relative to Multiflash dll

◆ ThermoWrapper 1.1 CalcSinglePhaseProp
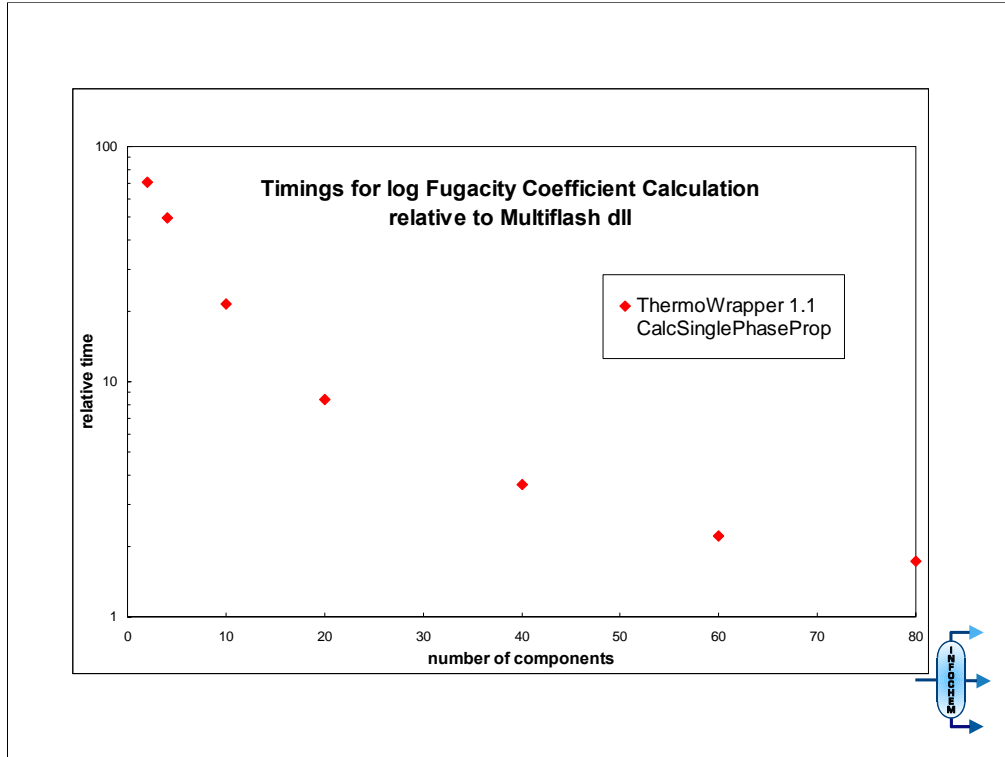
The time required to calculate the log fugacity coefficients was evaluated for the same mixtures as previously used. The calculation was done for both vapour and liquid phases at the bubble point evaluated at 300K. The results are presented on a log scale.

The overhead is much higher than for the overall enthalpy case because the property evaluation is much faster than the flash plus properties.

Timings for CalcSinglePhaseProp (VB PP)
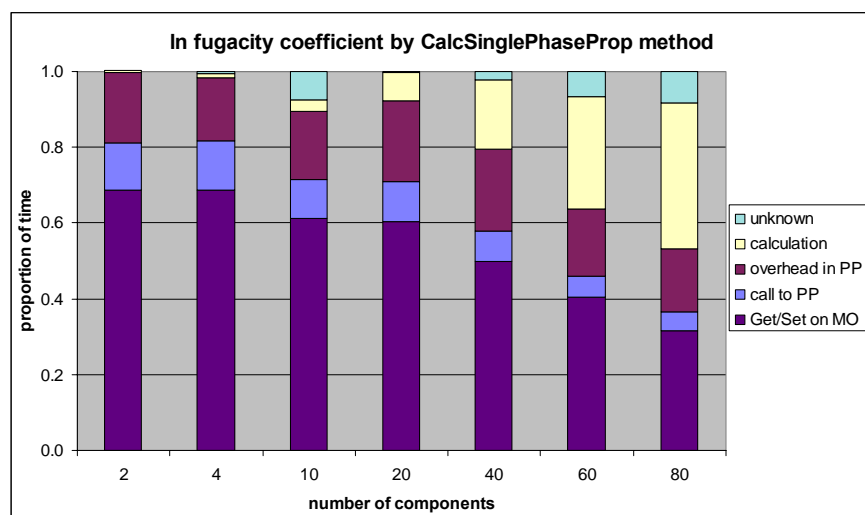
The total calculation time has been broken down into a number of categories. The timing was done on the old Multiflash PP implemented in VB which is not the one used in the rest of the tests. For the C++ PP the general trends are expected to be the same although the proportions will be different. There is considerable uncertainty in the results as illustrated by the proportion of time that could not be accounted for and is shown as 'unknown'.

As expected the Get/Set calls on the MO make up most of the time except for the largest mixture. 'call to PP' is the time required to go from the calling application to the PP and presumably is mostly type conversions. The overhead in the PP excludes any MO operations. The calculation time is what is required to evaluate the property in the Multiflash dll.

Property Calculation (CalcAndGetLnPhi)

- Method does not use Material Object for communication
- PME
  - **Call Property Package:**
    PP_CalcAndGetLnPhi(T,P,x,lnφ)
- PP_CalcAndGetLnPhi
  - **Type conversions COM to double**
  - **Calculate lnφ: call Multiflash dll**
 - **Type conversions double to COM**
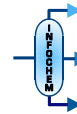
In the Thermo 1.1 specification there is a short-cut method called CalcAndGetLnPhi for evaluating the log fugacity coefficients (and derivatives). All information is passed by arguments and nothing needs to be set or got from the MO.

Timings for log Fugacity Coefficient Calculation relative to Multiflash dll
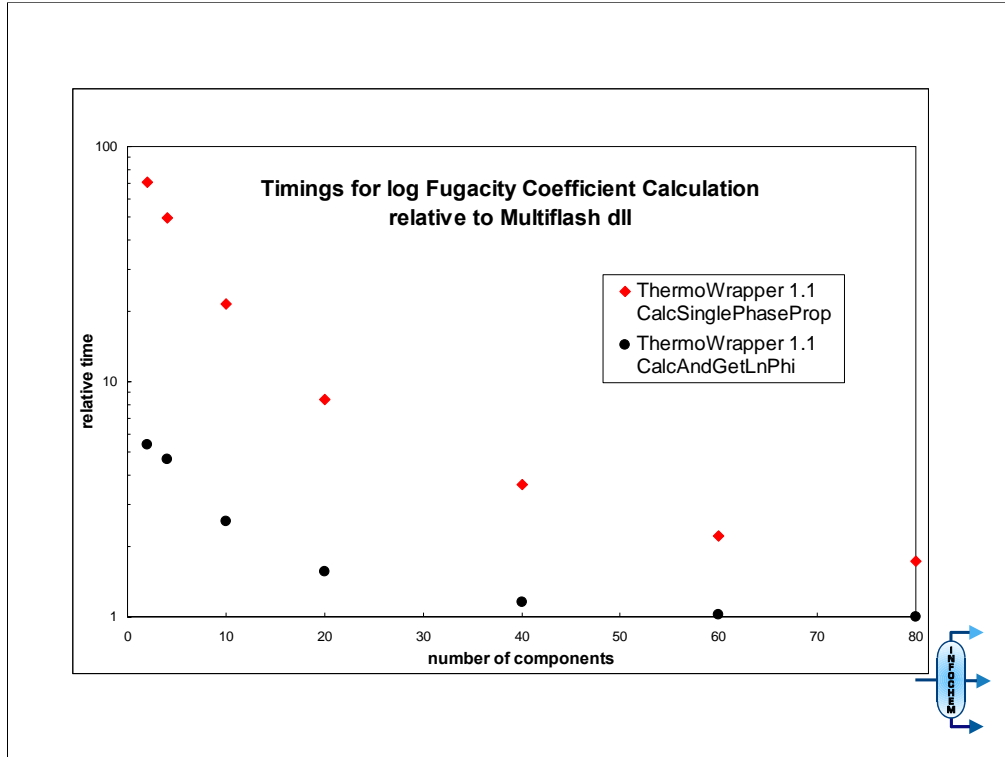
The timing exercise was repeated using CalcAndGetLnPhi and the results are shown in this slide.

The dramatic difference shows clearly the overhead of using the MO for communication.
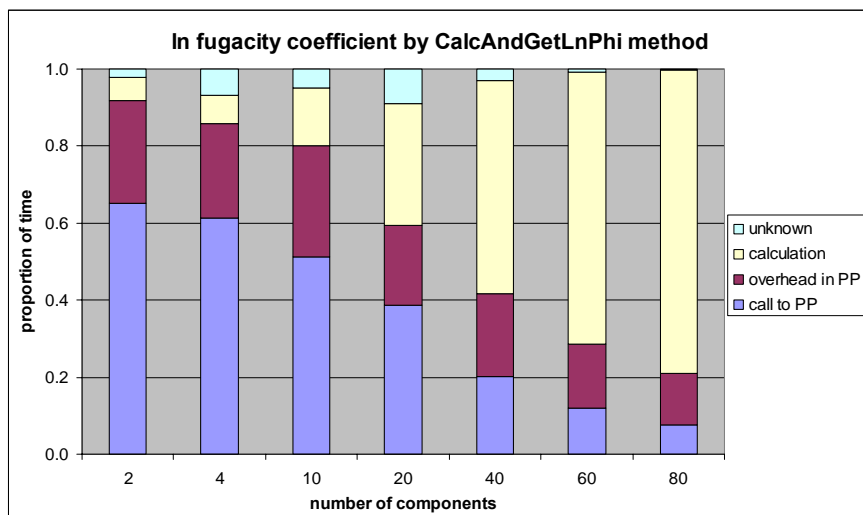
The breakdown of the total time shows how the overhead is reduced making the actual property calculation a significant part of the whole. Note that for 60+ components, the calculation takes up all time according to the previous slide. Some timing differences arise from the VB vs C++ Multiflash DLL.
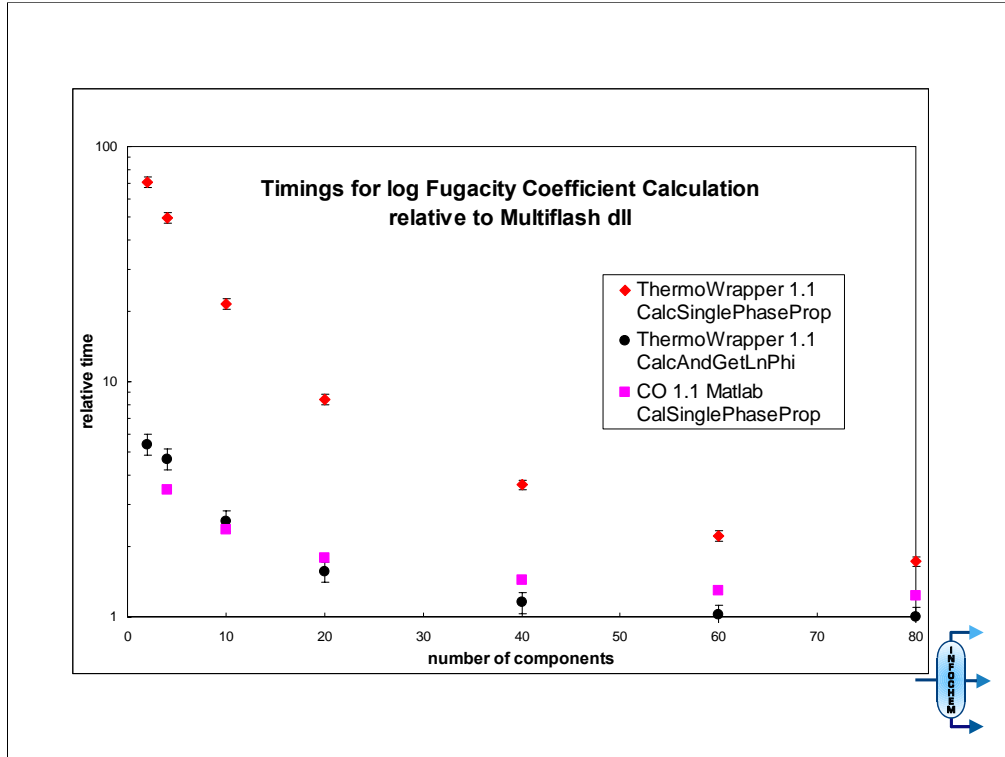
The timing was repeated again with the Matlab application which only implements the CalcSinglePhaseProp method. The scatter in the timings is also shown by error bars.

Because the Matlab application avoids many of the Set/Get calls on the MO the performance of the CalcSinglePhaseProp method is comparable with CalcAndGetLnPhi with the ThermoWrapper.

Conclusions and Recommendations 1

- **The overhead of using a CO property package can be made quite small: factor of between 1 and 2**
- **Much of the overhead seems to be associated with the design and operation of the Material Object**
  - **Competing objectives of efficiency and generality**
    - **error checking and diagnostics**
    - **type conversions**
    - **support of both thermo 1.0 and 1.1 in the same MO**
    - **PME interaction with MO**
  - **Attend the short course on implementing MOs**
- **Thermo 1.1 offers the possibility of more efficient operation**
  - **SetMaterial**
  - **GetTPFraction and GetOverallTPFraction**
  - **Fewer arguments**

The overheads of using a CO PP need not be very great. Good design of the MO is important. The requirements of a versatile, general-purpose MO can result in compromises on efficiency. Detailed information on MO requirements and design is provided in the short course on implementing MOs.

The Thermo 1.1 specification offers the possibility of more efficient operation for both the MO and PP.

Conclusions and Recommendations 2

- **PP Design is also important**
  - **Re-writing the Multiflash PP in C++ instead of VB reduces CalcSinglePhaseProp time by 25% for small no. of compounds, no difference for large no.**
  - **Essential to analyze the compound list efficiently and only when SetMaterial is called**
- **PME design**
  - **The PME should use SetMaterial only when the MO changes its compound list or compound order (or phase list for flashes)**
  - **PME owns the MO so can avoid all CO Set/Get calls**
- **Comparisons of CO and native applications for complete flowsheets would be more realistic for estimating overheads**
  - **However > 80% of simulation time is typically spent in phys props calculations**

Attention should also be given to the design of the PP. Changing from VB to C++ for the implementation of the Multiflash PP produced an efficiency gain of up to 25%. Analysing the compound list can be time-consuming because it involves string operations. It should be done efficiently, eg. by use of hash tables and should only be done when the MO changes, ie. on the call of SetMaterial.

PME design is crucial and again it is important that SetMaterial is only used when necessary and not prior to every call to the PP. The lists of compounds, phases and properties should only be obtained once from the PP and stored for reuse. When accessing the MO the PME should use efficient direct methods to set and retrieve properties.

Conclusions and Recommendations 3

- **Improvements to thermo interfaces**
  - **methods to identify compounds, phases and properties by integers (handles) rather than strings**
  - **Direct methods (similar to CalcAndGetLnPhi) for evaluating properties in order to eliminate use of MO as much as possible**
  - **SetTPFraction and SetOverallTPFraction methods to eliminate multiple references to MO**

It is now clear that the current (1.1) thermo interfaces could be improved in order to get better performance. Here are some suggestions for a future thermo version.

It is relatively expensive to process strings. String identifiers are used for compounds, phases and properties. This could be avoided by having methods that return the lists both as strings and integers. The integers could be used subsequently to identify the entities.

Reducing interaction with the MO reduces overheads and, probably, makes it easier to produce a MO. More direct calculation methods like CalcAndGetLnPhi would allow properties to be calculated and retrieved quickly and simply.

A minor but useful addition would be methods to set the temperature, pressure and composition in a single call.

Who Knows Where the Time Goes?
Computational overheads of using the thermodynamics interfaces

Richard Szczepanski, Infochem
Jasper van Baten, AmsterCHEM
Tom Williams, PSE

6th CAPE-OPEN European Conference
Munich
2-3 April 2009