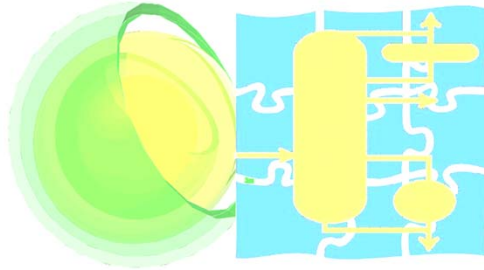


Consuming CAPE-OPEN Thermodynamics from Multi-threaded Applications

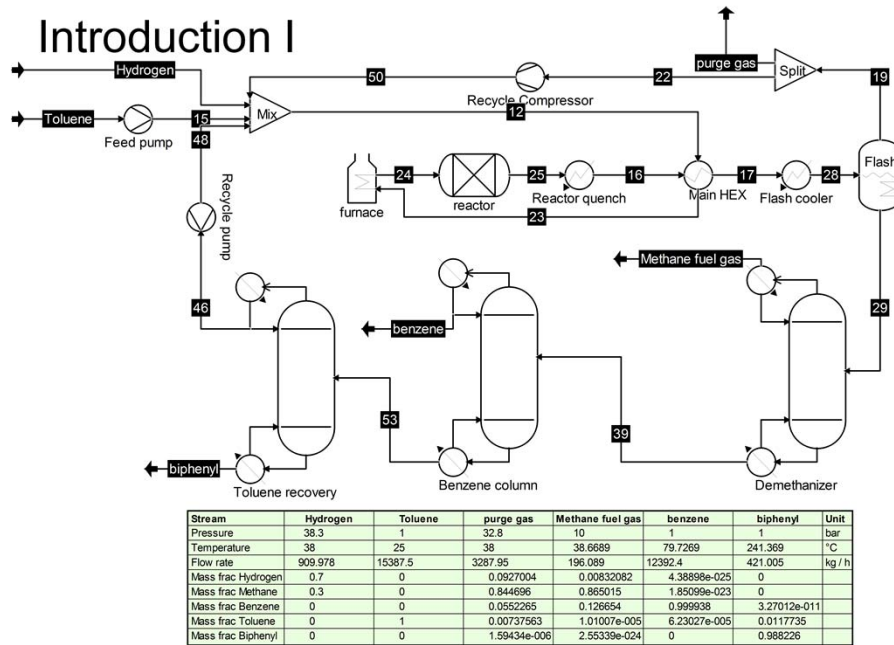


Jasper van Baten

AmsterCHEM

Good morning; my name is Jasper van Baten. I am an independent software consultant from AmsterCHEM, and the principle author of the COCO steady state flowsheet simulation software. This is a chemical process flowsheet environment that is entirely based on CAPE-OPEN. As with any chemical process calculations, COCO's flowsheet software COFE heavily depends on thermodynamic property and phase equilibrium calculations. COCO comes with the TEA thermodynamic server, but can be used in conjunction with other thermodynamic servers. In this presentation I want to share my experience with thermodynamics, CAPE-OPEN and multi-threading.

Introduction I

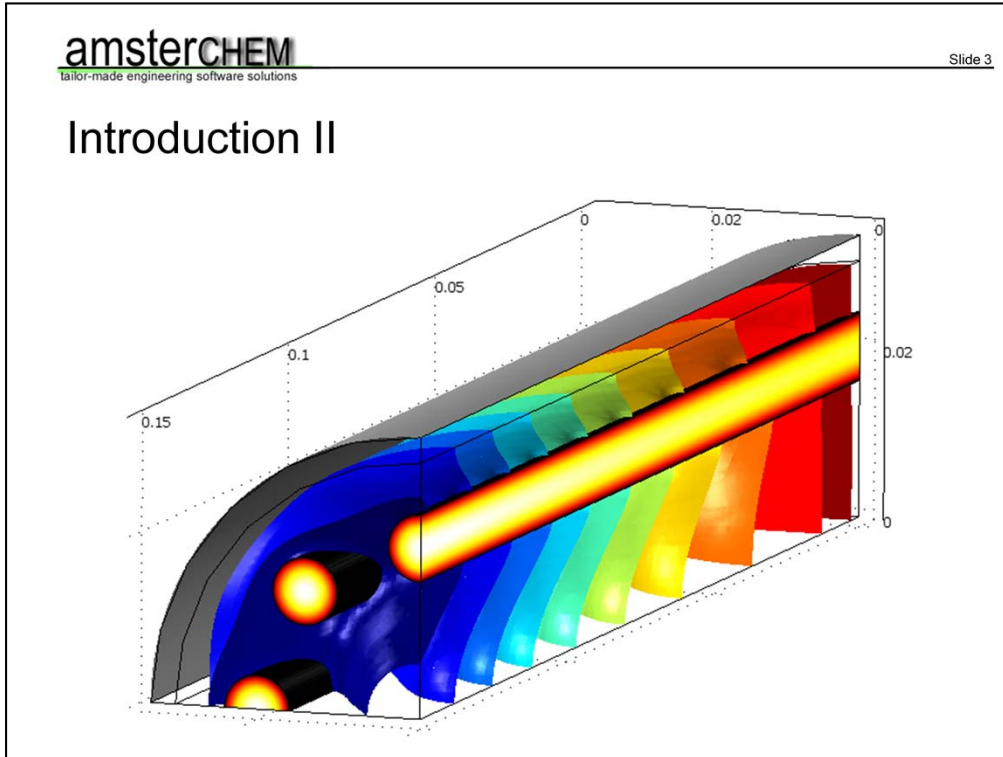


Let us start with setting the context. This slide shows a flowsheet for the simulation of the hydro-de-alkylation HDA process, as simulated in the COCO flowsheeting environment. Such flowsheet simulations are quite common in process engineering. They are used for design of new processes and analysis of existing processes. This particular flowsheet application is a steady state flowsheet environment. We see the feed streams of the process on the top left. Streams connect to models for process equipment, such as pumps, reactors, distillation columns, etc. These are called Unit Operations. Each of the unit operation models typically depends on thermodynamic property and phase equilibrium calculations to come to a solution. The flowsheeting application then has the task to come to an overall solution in which all the unit operation models are solved.

The thermodynamics are typically configured once, at the flowsheeting level. The unit operation models therefore typically all use thermodynamics that are served by the simulation application, instead of each unit operation defining its own thermodynamics.

The flowsheet is solved iteratively, as multiple recycles are present. This particular flowsheet is solved within minutes. If we look at optimization, the flowsheet has to be solved multiple times to meet the optimization criteria. In this case, processing time drastically increases. If we look at operator training, or process control, we need to move to a dynamic flowsheeting environment, instead of a steady state one; here too processing times drastically increase. Typically, about 90% of the time spent in flowsheet calculations is due to thermodynamic calculations.

Introduction II



Another area in which thermodynamic calculations may be used extensively is equipment design. The image shown here is part of a steam reformer simulated in COMSOL Multiphysics. In both the cooling tubes and in the reactor, computation fluid dynamics, or CFD, type of calculations are used to close the impulse, mass and energy balances. Pressure, temperature and compositions are time and space dependent. For accurate modeling, transport properties such as densities, viscosities and thermal conductivities used in these calculations can be based on rigorous thermodynamic calculations performed by an external thermodynamic server. Also here, the time spent in thermodynamic calculations is considerable.

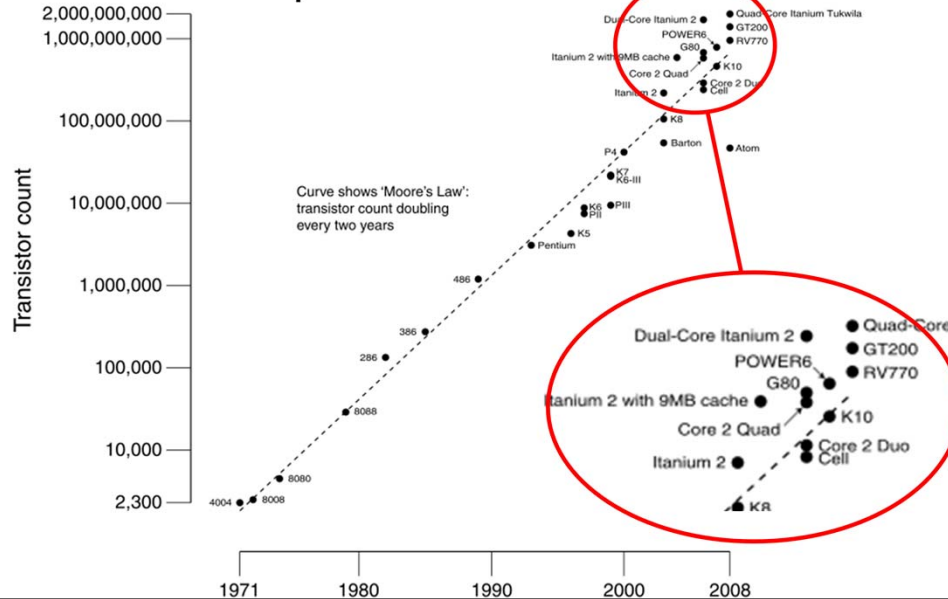
We see that for both process calculations, and equipment calculation, we need to think about efficient strategies to make get the best performance we can get, and part of that problem lies in efficient thermodynamic calculations. This presentation will focus on efficient computational strategies – this is not something that end users should be concerned with, but something software developers should think about.

Presentation outline

- Introduction
- Trends in computer hardware
- Distribution of computational load
- CAPE-OPEN
- Threading models in COM and marshalling
- Issues
- Conclusions

Now that we have set the scene, the remainder of the presentation will go into the details somewhat. First we will have a look at the current trend in computer hardware. We will see that we are best of distributing the computational load over multiple processes, or multiple threads in the same process. Next, I will tell something about CAPE-OPEN; this will be the source of all thermodynamic property and equilibrium calculations. Most CAPE-OPEN implementations run on Windows, and communicate via the Windows Common Object Model, or COM. Based on this object model, we will discuss threading models, and see which best to use. We will also discuss marshalling of COM calls, how performance is affected, and how to avoid it. Finally we will have a look at some issues that may result from implementations of thermodynamic servers that are currently commercially available.

Trends in computer hardware



Since the start of the computer era, the amount of transistors available on affordable computers, typically doubled every 2 years; this is known as Moore's law; the image is from Wikipedia commons. In an ideal world, this means that without further adjustments, our programs would run twice as fast every two years. This is not necessarily the case. If we take a closer look at the last portion of the line, as shown in the insert, we see that in the last couple of years, multi-core nodes appear. Whereas the increase in computational power was dominated by faster cores up to about 2004, currently computers power increases due to an increase in the number of computational cores. This means, if we want to benefit from faster computers, we need to divide our computations over multiple cores. Desktop computers equipped with up to 16 cores are not an exception anymore.

Distribution of computational load I

- Distributed calculations:
 - multiple processes
 - communication via TCP/IP (or Infiniband, ...)
 - flexible in scaling
- Multi-threaded calculation
 - single process, multiple threads
 - same process memory, fast communication
 - limited to single node (e.g. Desktop computer)
 - HyperThreading technology

To distribute the computational nodes over multiple cores, we have two choices. We can set up distributed calculations, meaning that we have multiple processes, each running their own set of calculations. If the calculations are completely independent and perfectly divided, this means that if we double the amount of cores, we halve the amount of computation time required. The calculations are however dependent on each-other, and as a result each process has to wait for the others to finish. After that, some communication is required between the processes to be able for all processes to proceed again. This communication takes a finite amount of time. Often this is done via TCP/IP, but faster specialized technologies such as Infiniband are available nowadays.

If we use a lot of cores, this communication is the limiting factor, and using even more cores, our calculations will actually slow down. Hence, for each process there is an optimal amount of cores. If we have enough cores available to use this optimum amount, distributed computing is very flexible in scaling. Often, the amount of cores used depends on the amount of cores available.

A typical situation is an engineer that performs the computer simulations at his own desktop machine. The amount of cores available is then limited to the amount of cores in his own computer. In this scenario, we can exclude a lot of the communication time by running in a single process. To do so, we need to run with a multi-threaded application, where each thread performs its own calculations. These threads all run in the same process, and therefore all have access to the same process memory. The amount of communication required is drastically reduced, and the

remaining communication is very efficient. As the engineer-working-at-his-own-desktop scenario is a very common one, this is the one we will focus on in the current presentation. With technologies like HyperThreading, the threads automatically distribute themselves over the available cores, so that the developer does not need to worry about that.

Distribution of computational load II

- Equal load division
- Minimum dependence
- Division depends on calculation at hand:
 - iterative processes: iteration per thread
 - grid based calculations: domain division
- Each thread will use thermo...

The question remains, how do we distribute the computational load over multiple threads? The optimal computational performance is obtained if each of the cores performs an equal amount of work, thereby occupying nearly 100% of the available computer power, presuming the number of threads is an integer multiple of the number of available cores.

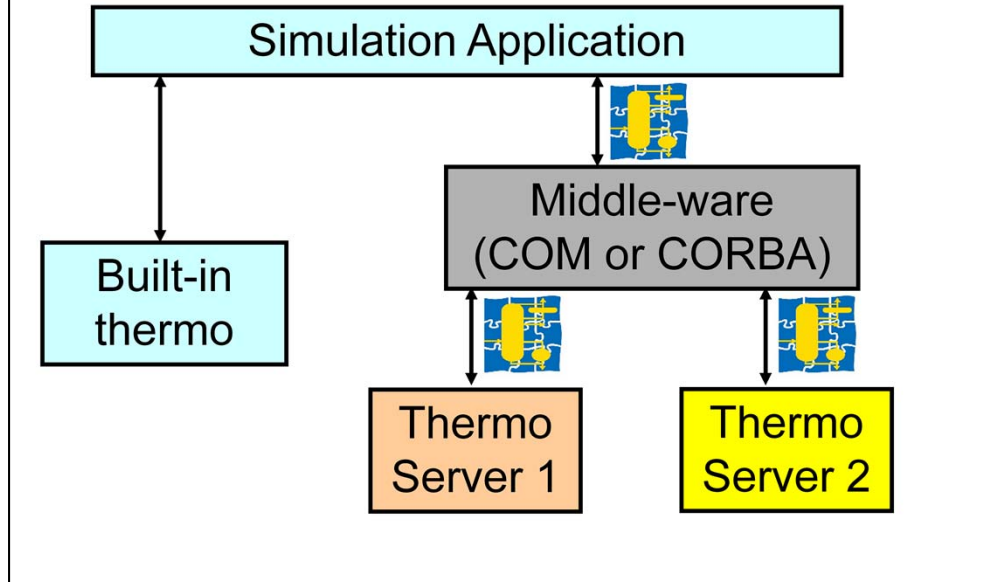
Also, we want to minimize the communication and synchronization between the different threads. We want therefore to distribute the calculations over the threads in such a manner that each of the calculations is as independent as possible.

The way to obtain this depends on the actual calculations being performed; no general rules can be applied. Some processes are iterative for example, and the iterations are independent in nature. This would for example be the case if a particular simulation is perturbed with respect to all of its inputs to obtain the sensitivities. For such processes, each of the iterations is independent, and a natural way to divide the load is by performing each iteration in its own thread.

For grid based calculations, such as CFD, it is common to cut the computational domain up into smaller pieces such that the connecting area between the domains is as small as possible. Each sub-domain is then calculated in its own thread.

From these examples, it shows that its very likely that the thermo calculations will get distributed over the threads as well. This means that each thread will use its own thermo and we should be prepared to access the thermodynamics server in a multi-threaded fashion.

CAPE-OPEN



The thermodynamic server: what is that? Which software is responsible for providing our calculations with thermodynamics?

Typically, flowsheet simulation environments have built-in thermodynamics. However, often it is desirable to have thermodynamics from a third party vendor that specialized in a particular field, or in-house thermodynamics such as used in the big petrochemical or chemical companies. Instead of depending only on built-in thermodynamics, most simulation packages these days allow for external thermodynamic servers. Support for external thermodynamic servers of many vendors is possible via a single open interface, called CAPE-OPEN; this is a set of interface definitions particularly designed for integration of process modeling tools. Specifically, the interface definitions for unit operation models and thermodynamic servers are widely used.

Using CAPE-OPEN, a simulation application only needs to implement its own CAPE-OPEN interface support to be able to communicate with multiple thermo servers, each of which provides its own thermo support. The communication takes place via middle-ware to provide the communication mechanisms. In Windows, this is the Common Object Model (COM); CAPE-OPEN also supports the CORBA middle-ware, which is in principle platform independent, but requires additional software installation to work. The COM implementation of CAPE-OPEN is particularly popular at this moment, and the focus of this presentation.

COM: threading models I

➤ *Apartment threaded:*

- only access COM objects from one thread
- multiple single-threaded apartments can exist

➤ *Free-threaded:*

- access COM object from any thread


COM provides various threading models. Two of them are common, and interesting to discuss in this context.

In the Apartment Threaded model, COM objects can only be accessed from the thread that created them. It is not allowed to access a COM object from another thread. However, multiple apartments can exist, each of which must be single-threaded.

Then there is the Free Threaded model, in which each COM object can be accessed at random from any thread.

Both the COM server, which is the thermodynamic server, and the COM client, which is the simulation application that consumes the thermodynamic calculations, can choose their own threading model. It would therefore be very intuitive for a multi-threaded simulation application to choose for the Free Threaded COM model.

COM: threading models II

	Client Apartment threaded	Client Free Threaded
Server Apartment Threaded		
Server Free Threaded		

This is however in practical scenarios most likely not the best choice. If the requirements by the client cannot be imposed on the server without violating the server's advertised threading model, the calls made to the server will not be made direct. Windows COM machinery will sit in between to make sure that the server's wishes are being honored. The process in which calls are being translated is called marshalling. Marshalling means extra overhead, and on thermodynamic calls that can be very efficient and fast, any small overhead may have a big impact. For the particular case of marshalling invoked by threading model differences, Windows will synchronize all calls to the thermo server onto the thread that created the server. In the best case scenario, this is a separate thread that will now be responsible for doing ALL the thermo work. In the worst case scenario, this is one of the calculation threads, and this thread now has to do its own calculations AND all thermo calculations of all other threads. Such marshalling should be avoided.

If the server is Free Threaded, it can deal with both Apartment Threaded and Free Threaded access. In this case marshalling is never required. For nearly all thermodynamic servers that are available however, the threading model is Apartment Threaded. In this case, if the client chooses the Free Threaded model, marshalling is invoked; unless we can guarantee that the server is Free Threaded, the client should not choose for the Free Threaded model, and use the Apartment Threaded model instead.

Of course this has the implication that each of the calculation threads needs to create its own copy of the thermodynamic server.

COM: Marshalling

- Threading model choice

- Different process-space:
in-process vs. out-of-process
64-bit vs. 32-bit

- Data marshalling due to data format:
.NET vs native

Now that we are aware that marshalling should be avoided if we want to access thermodynamic calculations in an efficient manner, let us look at some other reasons for marshalling; all of these should be avoided.

We have seen in the previous slide that the client should choose its threading model well. It should create an Apartment Threaded COM object in each calculation thread.

Marshalling is also invoked if the data of the client and the data of the server do not live in the same process space. This is the case if the server is implemented as an out-of-process executable rather than as an in-process DLL. For out-of-process information, all data passed between client and server is copied via RPC mechanisms. The overhead can be immense.

The client has no control over this, but the server should make sure it is in-process. This however brings about an issue with 32-bit implementations vs 64-bit implementations. One cannot load a 32-bit DLL into a 64-bit application, or vice versa. So a server best implement both a 32-bit version and a 64-bit version, instead of going for the single out-of-process implementation.

A third reason for marshalling is if the data of the client is not compatible in format with the data of the server. This happens for example if the server is native, and the client is managed (i.e. running in the .NET Common Language Runtime). Although

.NET applications have built-in COM support, the COM data types are not directly accessible from .NET and will be translated. As nearly all servers and nearly all clients are native, think twice before implementing either a server or a client in .NET.

Issues:

- Consistent objects in all threads: persistence
- 64-bit software support
- Not all thermo servers currently thread safe
- COM fading out??

As with development of all software, some issues will be encountered. This slides shows some of these issues that require some consideration.

First, as all calculation threads need to create their own thermo server object, all of these objects must be consistent. What if the user is editing the thermo package in the application? The user may very well want to modify the thermo, add or remove compounds, select different models, change parameters, etc. These changes need to be reflected in the calculation threads. This issue is easily resolved though. CAPE-OPEN has a mechanism for persistence; the storing and loading of the state of a COM object, in this case the thermodynamic server configuration. The persistence mechanism is intended to be used for saving the state of the COM server to file, so that upon loading the file, the COM server can be used in the state that it was saved in. This mechanism can also be used to create copies of COM servers in general: save the original, create the copy, restore the copy from the saved version.

Therefore, this can also be used to make a copy in a different thread. So at the start of the calculation, the thermo servers in each thread can be copied from the original that the user may have modified.

Next issue: we have seen that to prevent marshalling, we need in-process thermodynamic servers. Pretty much all process simulation software around these days is still 32-bits. So are the thermodynamic server implementations. As the

memory requirements of simulation applications change, these applications will start to migrate to 64-bit implementations, and can no longer use the 32-bit server implementations. The thermodynamic software vendor therefore will have to provide 64-bit support. Currently, nearly none of the thermodynamic software vendors do. It is expected that as the 64-bit software becomes more popular, the thermo software vendors will have to keep up with the developments and provide 64-bit support for their thermodynamic servers.

During the testing and evaluation of the multi-threading support in COCO, it was found that not all thermodynamics servers that are currently available, are thread safe. The servers are generally Apartment Threaded, and it is apparently assumed by the thermo vendors that this means that their software can only be accessed from a single thread. This is not the case, as multiple single-threaded apartments may exist, and according to the scenario sketched here, will exist. As a result of the server not being thread safe, crashes may occur, or worse, thermodynamic calculations may result in incorrect results. Again it is expected that these issues will be resolved by the thermodynamic software vendors in the near future, as multi-threaded thermodynamic consuming applications become more popular.

Finally, the current CAPE-OPEN implementations are mostly based on COM. Why? Because it works. COM is one of the few middle-ware implementations in which marshalling-free interaction of the server and the client can be accomplished; CORBA cannot do this; all CORBA servers are out-of-process. It is for this reason that even Microsoft uses COM as the basis of many of its current products, like office, visual studio, internet explorer, ... If COM will be fading out, it is in my view because it will be replaced by other middleware that can accomplish marshalling-free interoperability. In this case, CAPE-OPEN itself has to adapt to this new standard. Another disadvantage of COM is of course that it is bound to Windows platforms. It would be nicer to have generic middleware that is not. Perhaps CAPE-OPEN should provide its own middle-ware in the future, currently is it not. As long as we are willing to develop only for Windows, we can at this point safely use COM. Even if COM fades out, all we need to do is replace the COM middleware layer and our picture will be complete again.

Conclusions:

- Multi-threading is good for engineering Desktop applications
- CAPE-OPEN a good way to access multiple thermodynamic servers via one common interface
- COM is the middle-ware
- Threading model: Apartment Threaded
- Avoid marshalling
- Problems will be resolved as time passes
- New middleware?

To summarize:

In the scope of engineers having access only to their own Desktop computer hardware, multi-threading is a good way to distribute the computational load over multiple cores; all threads share the same memory, making for cheap communication.

CAPE-OPEN is a good way to access thermodynamics of multiple vendors and only have to implement a single interface set; the CAPE-OPEN standards are available from the CAPE-OPEN Laboratories Network, CO-LaN, at www.colan.org

COM is currently the middle-ware to pick. The reason for this choice: marshalling-free and thus very efficient implementations.

To keep marshalling free, some more choices need to be made: the client should choose the Apartment Threaded model and create a thermodynamic server COM object in each calculation thread.

The server should allow avoiding marshalling, by providing an in-process thread safe implementation. A 64-bit implementation should be provided to support 64-bit

applications.

Problems with existing applications are expected to be resolved by their respective software vendors as time passes and multi-threaded and 64-bit thermo access becomes more common.

Perhaps an alternative middle-ware would come in handy; one that allows platform independent marshalling-free operation. A generic solution does not appear to be available. Perhaps CAPE-OPEN should come up with one.

Status:

- COFE implements multithreaded calculations since 2.0:
 - concepts work
 - particular problems encountered as outlined
 - many of these already solved by the vendors
 - not diving load well (future work)

- COMSOL Multiphysics implements thermo import
 - all calculations are multi-threaded
 - 32-bit and 64-bit implementation

COFE is the flowsheet simulation environment of the COCO package; COCO is available free of charge, and includes a flowsheet, thermodynamics, unit operations, reaction packages and more, all available as separate modules that communicate via the CAPE-OPEN standards. Since version 2.0, COFE performs its main flowsheet calculations in a background thread, so that multiple flowsheets can be solved at the same time, and the user can continue to use the main application while solving flowsheets. Although this is not geared specifically to solving the flowsheet quicker by employing multiple cores, all the concepts that have been discussed are applied in this scenario, and they are work. During this implementation, many thermodynamic servers have been tested. I had already indicated at previous slides that some implementation issues were found. These were reported back to the vendors, and most vendors have already provided either fixes or workarounds for now, and fixes later.

COFE allows for parametric studies, in which the influence of particular inputs on the solution of the flowsheet is tested. This requires solving several individual flowsheets. These are solved at different cores in different threads; here the full potential of multiple cores is realized.

Another application that uses multi-threaded calculations with CAPE-OPEN thermodynamics is COMSOL Multiphysics. Here too, the concepts have been

proven to work. This application also provides a 64-bit implementation. 32-bit thermodynamic servers are available, but via a detour and out-of-process. The impact of out-of-process marshalling is found to be severe.

Both of the applications mentioned here are available to thermodynamics software developers to test their implementations; COCO/COFE is available free-of-charge.

- Download COCO: <http://www.cocosimulator.org/>
- Compliancy testing program
- CAPE-OPEN standards: <http://www.colan.org/>
- CAPE-OPEN forum:
<http://capeopen.19.forumer.com>
- CAPE-OPEN and software consultancy:
<http://www.amsterchem.com/>

COCO is freely available for download from [cocosimulator.org](http://www.cocosimulator.org). Feel free to download a copy; many have done so already, from all over the world (next slide)

At the COCO simulator site you can also find the CAPE-OPEN compliancy testing programme; if you are a CAPE-OPEN developer, do check this site out.

The CAPE-OPEN standards are open; this means that they are available free-of-charge to everybody, and everybody can make their own client or server implementation. The standards are provided and maintained by the CAPE-OPEN Laboratories Network, and available from <http://www.colan.org/>

CAPE-OPEN developers may want to check out the CAPE-OPEN forum.

AmsterCHEM provides chemical engineering software consultancy, and specialized in CAPE-OPEN implementations. For more information, go to www.amsterchem.com.

amsterCHEM
tailor-made engineering software solutions

Slide 16

ChemSep
Modeling Separation Processes

HTRI
Xchanger Suite 5.0

EPA
ENVIRONMENTAL PROTECTION AGENCY
UNITED STATES OF AMERICA
EPA WAR & .NET libraries

NIST
National Institute of Standards and Technology
REFPROP 8.1

VMGThermo
5.0

gPROMS
PSE
PSE gPROMS 3.3

AixCAPE
Props 1.0

ANSYS
Fluent 12 & APECS

COMSOL
Multiphysics 4.2

COSMOlogic
Computational Chemistry and Fluid Thermodynamics
CosmoLogic
CosmoTherm C21

nel
TUV-NEL
PPDS v4.1.0.0

Scilab
DIGITEO
Scilab 5.2

Infochem
Multiflash 3.9

THERMODYNAMIK
Ruhr-Universität Bochum
GERG-2004 XT08

ProSim
ProSimPlus 3 / Simulis 1.3

SolidSim
Adding solids to the flowsheet

invensys
SIMSCI-ESSCOR
Pro/II 8.3

Finally, Jasper van Baten would like to thank all companies that generously provided software and licenses to AmsterCHEM for CAPE-OPEN development and testing.