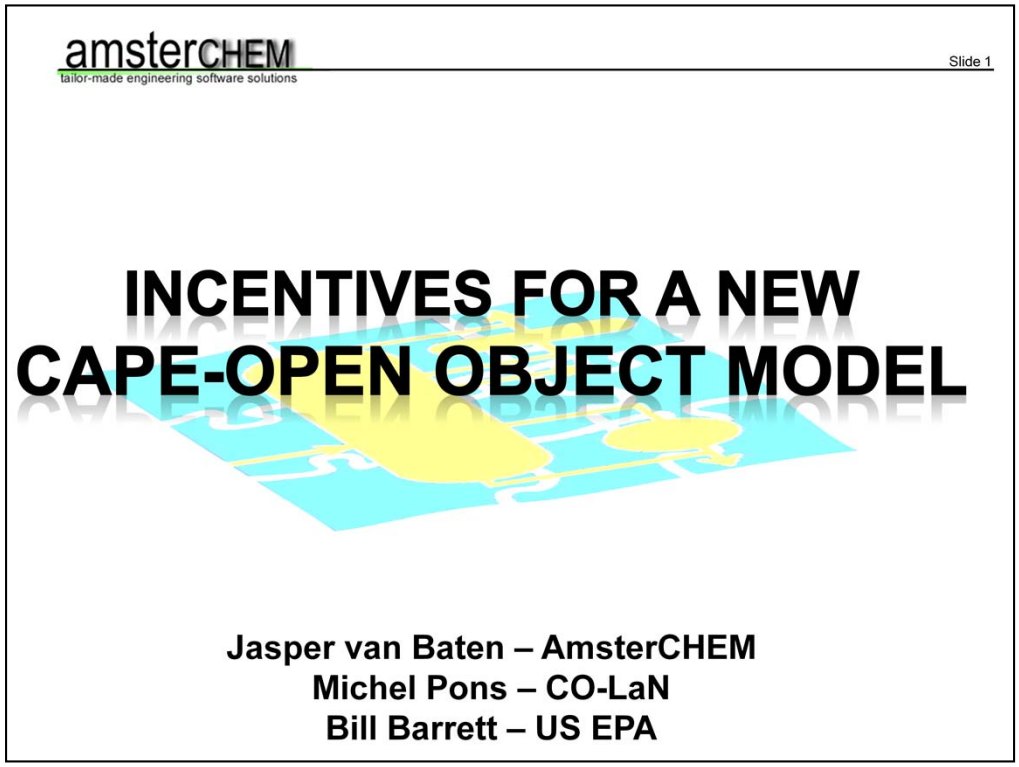
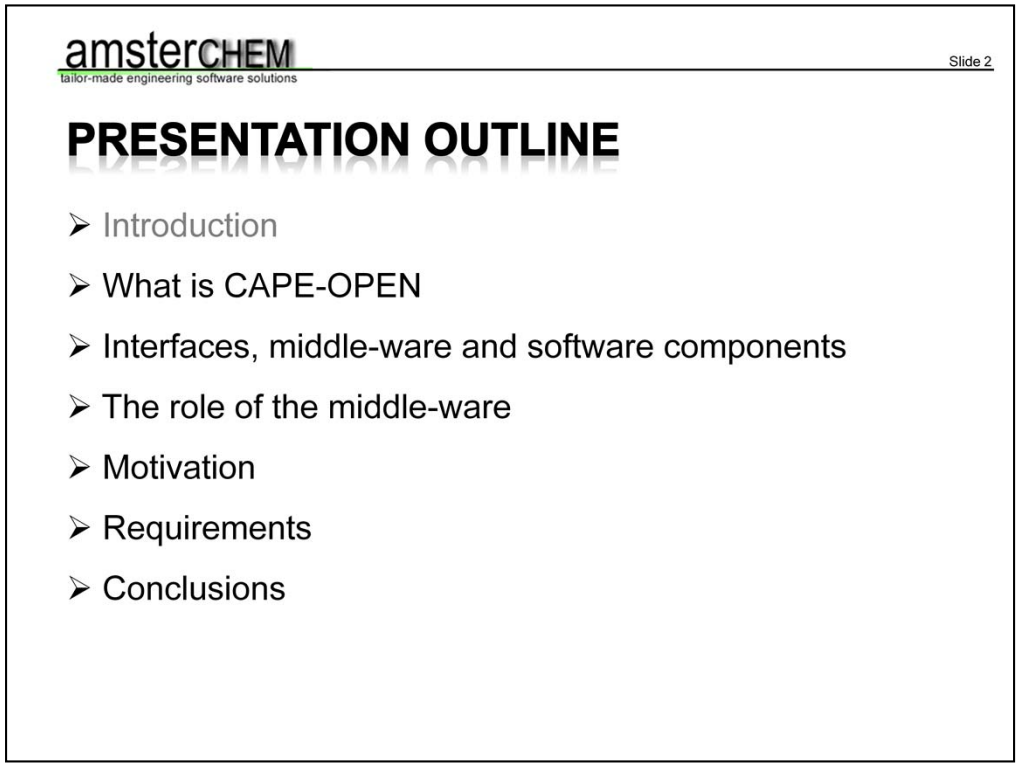


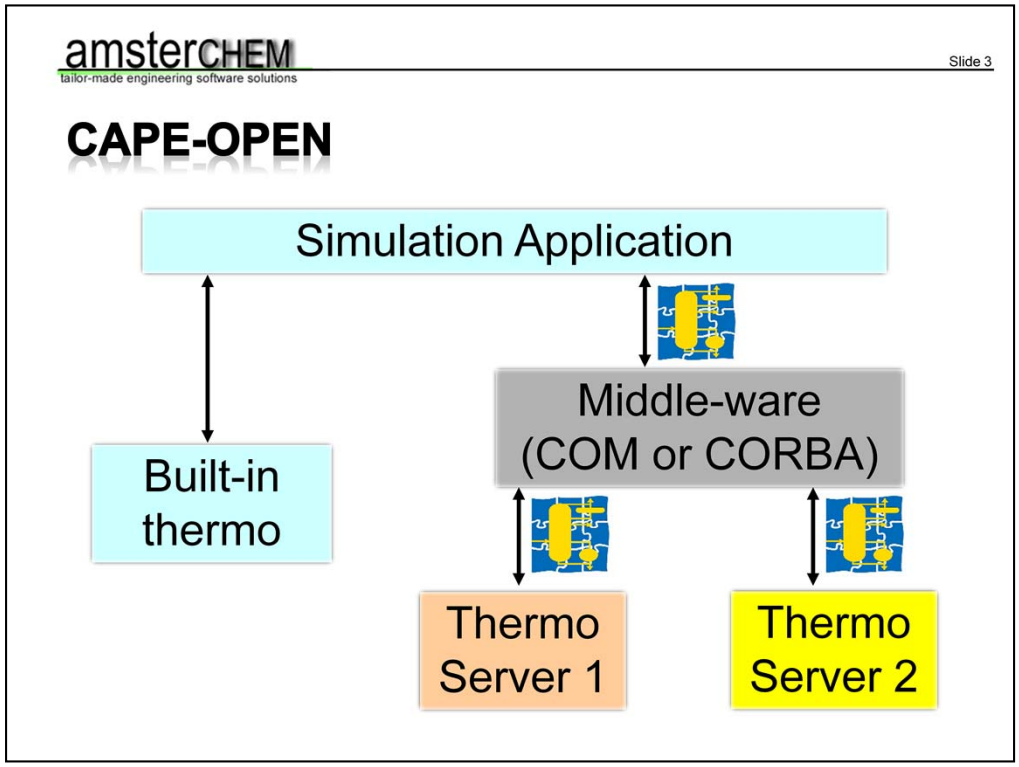
Good morning; my name is Jasper van Baten. I am an independent software consultant from AmsterCHEM, and the principal author of the COCO steady state flowsheet simulation software. This is a process flowsheet environment that is entirely based on CAPE-OPEN. CAPE-OPEN is an industrial interface standard specification that allows chemical engineering software components to be exchanged between simulation applications. CAPE-OPEN provides software components with a “language” to talk to each other. CAPE-OPEN suggests two means of communication, or middle-ware, COM or CORBA. In practice, pretty much all applications implementing CAPE-OPEN nowadays use COM. This means that CAPE-OPEN software generally is bound to Windows based platforms. COM programming is very flexible, but rather complex as well. In addition, the COM model causes some overhead, which has a negative effect on CAPE-OPEN performance. Finally, according to Microsoft the COM days are nearing their end. So it is now time to review whether COM is still the middle-ware to use, or perhaps to move to something more suitable.

This presentation was put together by myself, as well as Michel Pons, the chief technological officer of the CAPE-OPEN organization CO-LaN and Bill Barrett, of the US environmental protection agency, who has extensive experience with CAPE-OPEN in .NET.

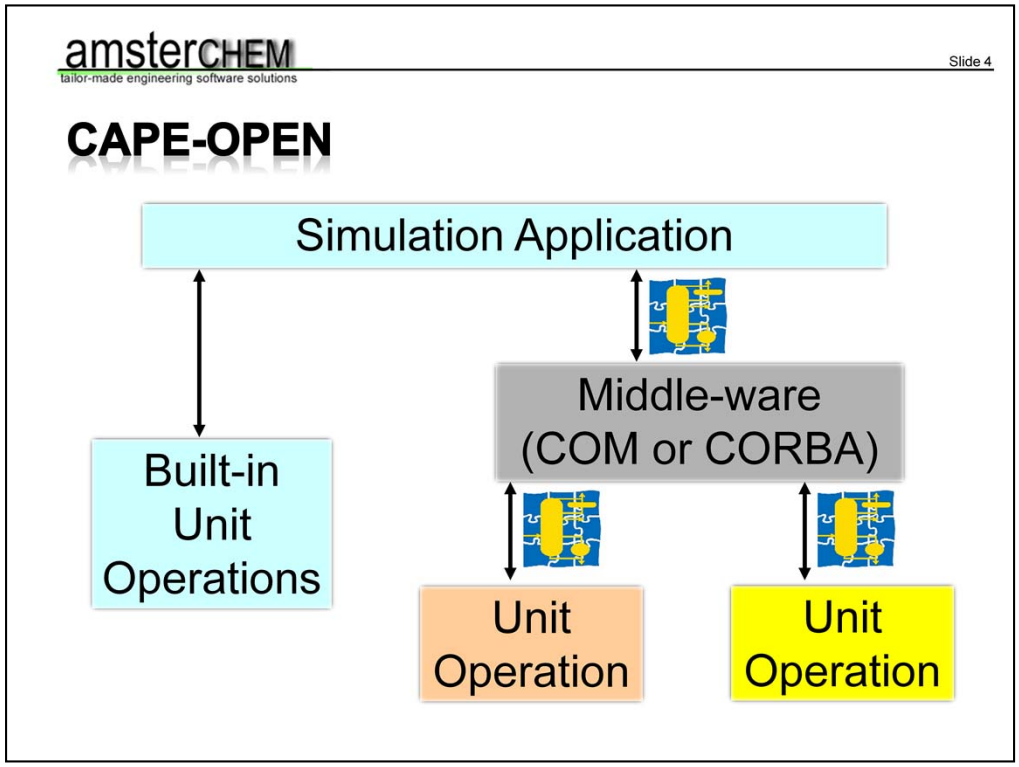


Obviously this presentation is about technical details of interfacing software packages. I will try to focus on conceptual aspects rather than implementation aspects as much as possible. The aim of this presentation is to suggest that for the purpose of CAPE-OPEN, COM should be replaced with an alternative. The alternative must be well geared towards ease of development and less error-prone implementations, performance, but also portability.

We will start by showing what CAPE-OPEN is, and what its role is in the interfacing of chemical engineering software applications. Then we connect the dots and point out that CAPE-OPEN is merely a set of interfaces, that is implemented in the context of a middle-ware and run by specific application implementations. We will proceed with stating why COM is currently the middle-ware of choice, and where there is room for improvement. Finally, we will suggest a course of action for future implementations. But first things first.

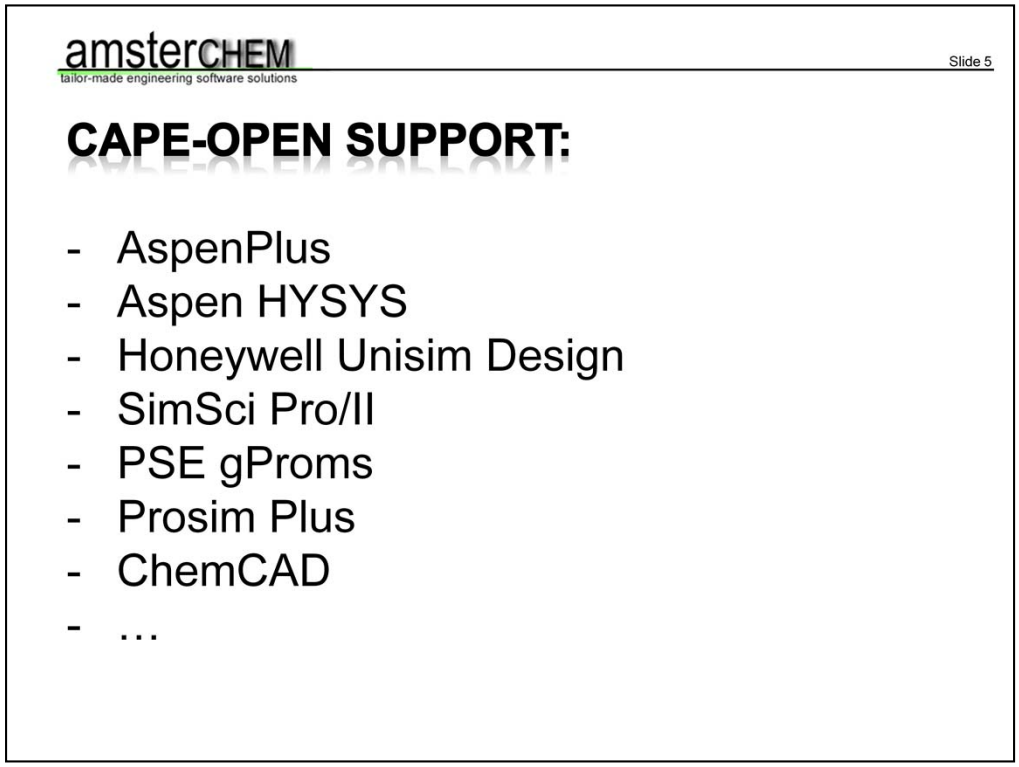


Here we see a schematic view of the thermodynamics in a simulation software package. Typical examples could be flowsheeting applications, but in fact any application that uses thermodynamics fits this picture. Typically, but not always, there is some built-in support for thermodynamics. Sometimes this built in support is not sufficient, sometimes users want their thermodynamics to be consistent with other applications, sometime specialty thermodynamics by thirds party vendors are required, or sometimes for other reasons the user may want to use thermodynamics of another software package. A simulation environment may provide a specific interface – such as user FORTRAN – to include customized thermodynamics, but an open interface exists. This is CAPE-OPEN. The simulation environment provides a CAPE-OPEN socket for thermodynamic systems, and any thermodynamics package implementing the plug side can now be used. The communication takes place using CAPE-OPEN interface definitions, and the objects are created and accessed via middle-ware. Typically this is the Common Object Model, or COM.

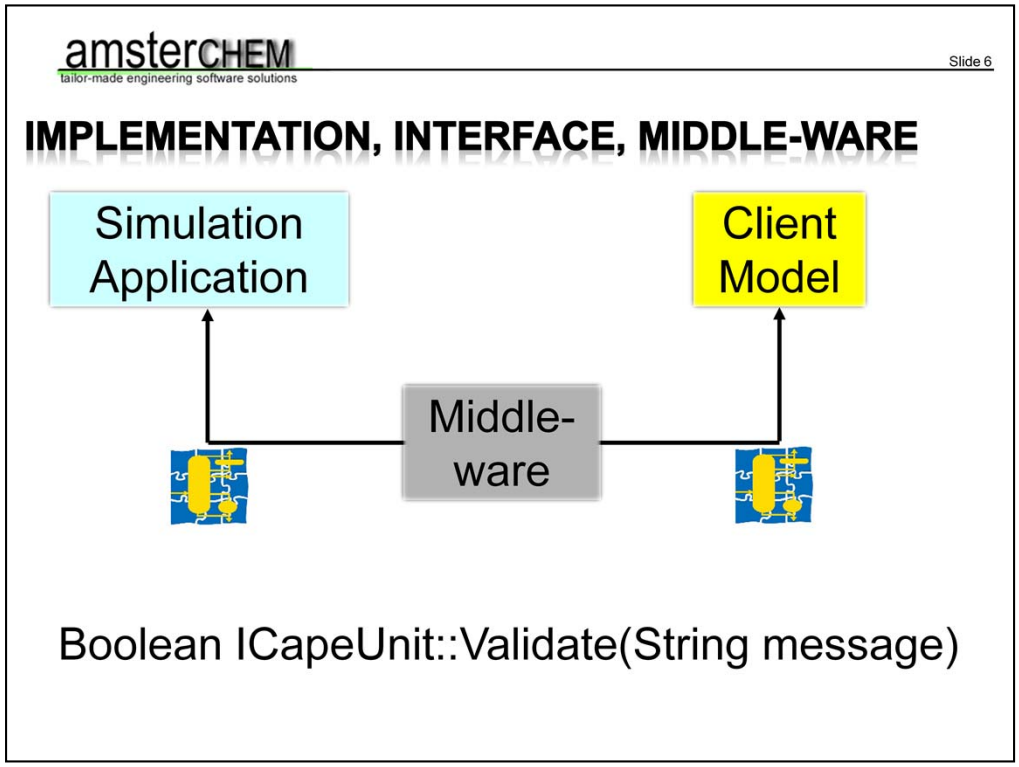


For flowsheeting applications, another class of models used is Unit Operations, that represent physical equipment, such as a pump, a reactor, a separation tower…. The same story holds; a simulation application will likely have built-in Unit Operation models, and likely provides some custom way of interfacing with user models. If in addition a CAPE-OPEN Unit Operation plug is implemented by the simulation environment, third party or user unit operations that expose a CAPE-OPEN plug can be used.

CAPE-OPEN contains many more interfaces than just thermodynamics and Unit Operations, but these are the two most frequently used and widely supported ones.



For those of you that have never heard CAPE-OPEN, it is not some exotic standard that is implemented only be a few programs. Support for CAPE-OPEN is present in nearly all main-stream flowsheeting applications. I have listed a few here that you may be familiar with. These are just a few simulation applications, there are others, including the freeware aplications COCO and DWSim. In addition, there are various thermodynamic plug and unit operation plug implementations. To name a few, Multiflash, VMG Thermo, Ansys Fluent, HTRI heat exchangers, Chemsep, and more. More-over, many chemical companies use in-house thermodynamics or unit operation models. Such companies can adapt their in-house models to use CAPE-OPEN and subsequently use them in all simulation platforms that support CAPE-OPEN.



So what is the role of the object model, or middle-ware?

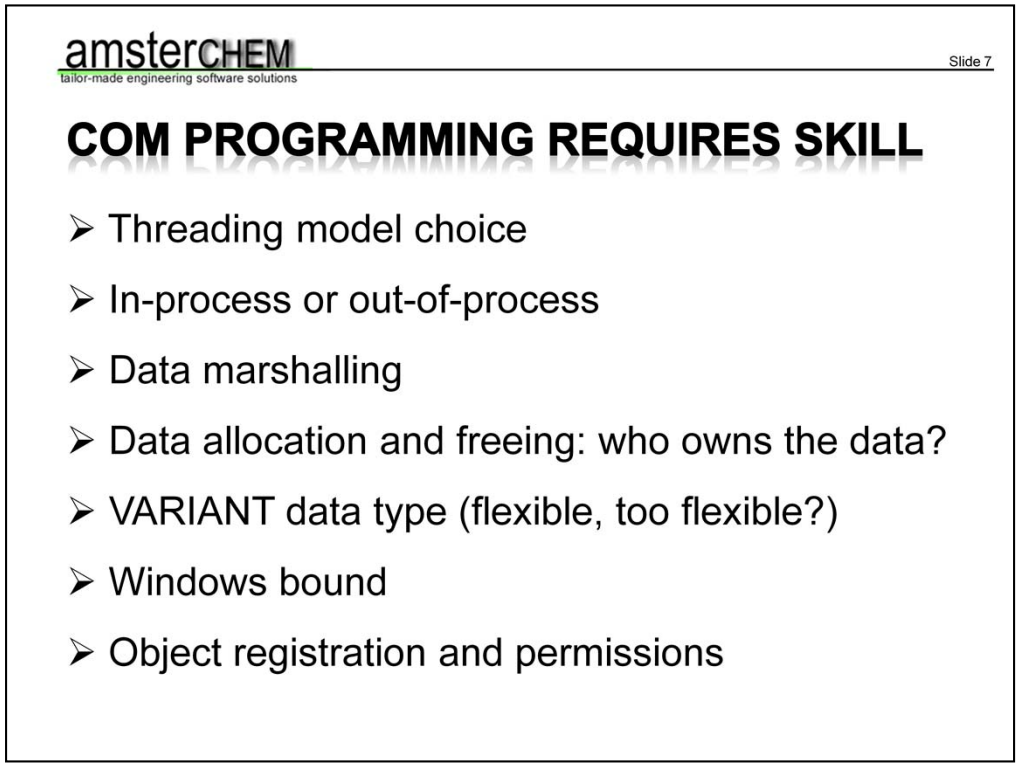
Here we see a simulation application talk to a client model. It wants to call the function Validate that is defined by an interface ICapeUnit. The Unit Operation interface ICapeUnit is part of the CAPE-OPEN standard specification. Hence, the simulation application and the client model both know that an interface ICapeUnit exists, and that it implements a number of functions, including Validate. This is a good start, but not sufficient.

First, the simulation application needs to create the client model. So it needs to be aware of all clients models on the system that implement ICapeUnit. Providing this information is the task of the middle-ware. Then the simulation application needs to create an instance of the client model. Again, the task of the middle-ware. The simulation application then needs to obtain a reference to the ICapeUnit interface, which is done using functions obtained by the middle-ware. On this interface, the Validate function needs to be called. This has a string argument. The way this argument is encoded is defined by the middle-ware. Also the memory that is allocated to store the string is arranged by the middle-ware.

For people familiar with COM programming, the registry lists which objects are available, CoCreateInstance takes care of object creation, QueryInterface obtains an interface reference, the string is an automation BSTR type that is allocated and released with SysAllocString and SysFreeString.

Typically, the simulation application and the client exist at the same computer and

run in the same process space. This is however not required, you can run the client at a different computer than the simulation application. The middle-ware takes care of the inter-process and inter-computer communication in this case.
So, CAPE-OPEN defines the types of objects that are available, what interfaces they expose and what functionality is implemented by an interface. Beyond that, all communication is done by the middle-ware. The implementation of the functionality is of course done by the simulation application and client model themselves.



So why do we think new middle-ware is in order? There are several reasons.

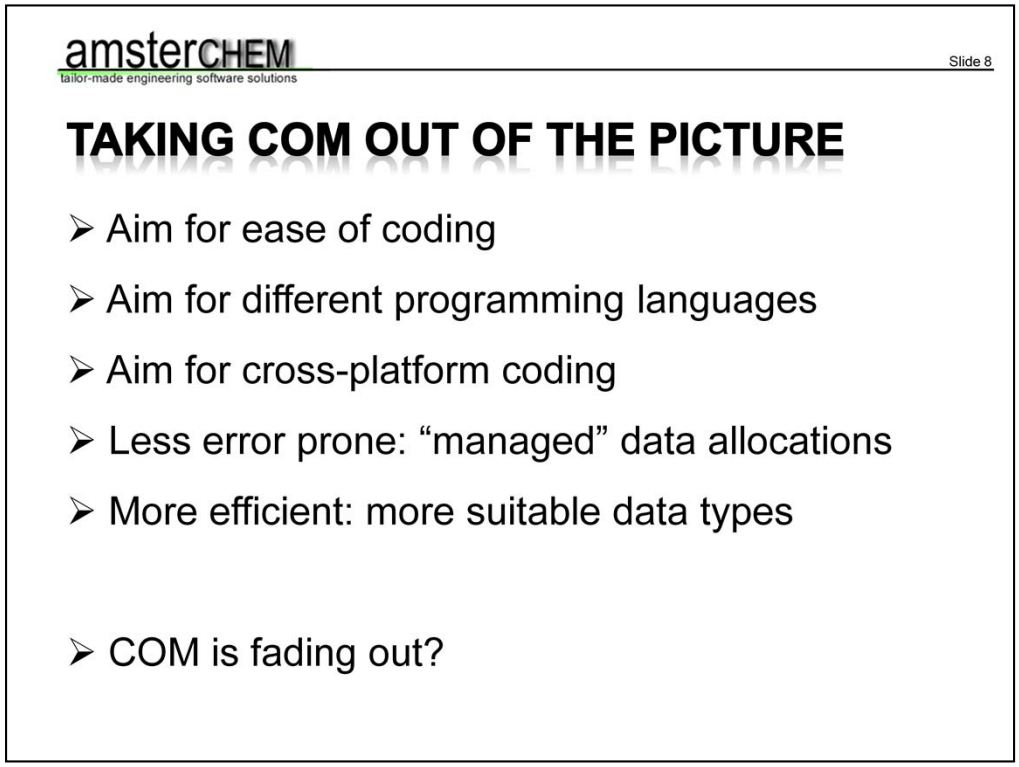
First, COM programming requires skill. Chemical engineers that want to implement their own process models are generally not computer scientists. Yet, programming any COM object requires making choices that are not always that simple, and skills to implement those choices, and even more skills to do so in an efficient manner.

We have to choose a threading model, and whether to write an in-process or out of process COM server. Generally we want an in-process COM server as this is better for performance. On Windows, this means we need to create a DLL. Then we have to worry about efficiency of data marshalling. We could take a short-cut and use for example a .NET language, but now we implicitly get marshalling of data from COM data types to .NET data types. Even though this is fairly quick, on a simple calculation such as the enthalpy of a vapor mixture, this can take up a considerable percentage of the time, and can therefore affect performance pretty drastically.

Next, we have to worry about who owns what data. If we do not own data we should not free the data that was allocated. If we do own the data but we fail to free the data, we end up with memory leaks. The COM data is also packed in a flexible, but complex manner: the VARIANT structure. An array of double values in CAPE-OPEN is packed in a SAFE ARRAY structure inside a VARIANT. Not only complex to code, but also not very efficient.

As a result of choosing COM, we are now bound to Windows based platforms, hence, no easy porting to Mac or linux.

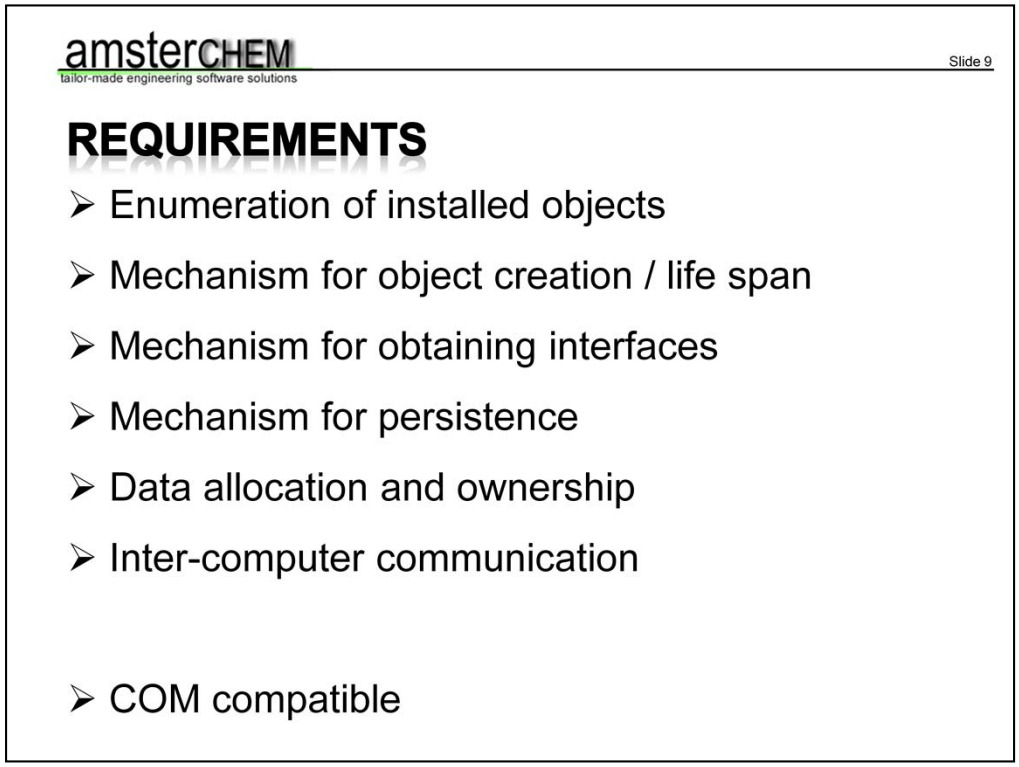
Finally we have to consider whether we register the object for a single user, or for all users. In the latter case we require administrative rights and in companies this is not always easy to obtain.



So, if we take COM out of the picture we can build an object model that is more suitable for CAPE-OPEN. The first aim is to make it more accessible to chemical engineers that do not have much coding experience. Hence, the framework for a CAPE-OPEN model should largely be generated. This should be done in a variety of languages; many process models are coded in C++ or FORTRAN. These should of course be supported. In addition, more and more .NET languages are being used. This should also be supported by the new middle-ware. This new middle-ware should of course not be limited to Windows platforms only, so should be available for Windows platforms, linux platforms, Mac platforms, and should be open source so that it can be compiled and used on any target platform.

Furthermore, the programmer should not have to worry about who owns the data. The data should be owned by the middle-ware, and automatically cleaned up in the destructor of the object that contains the data. The data itself can be allocated more efficiently. A double array could reside in a class called CapeDoubleArray, which would clean itself up and have function to access the data directly, or by element. This again makes life easier on the programmer, as it is not possible for a client to return data of the wrong type anymore, which is quite possible in the COM based setup. No more need to securely check the data types.

Finally – COM is fading out. Although Microsoft still does a lot of programming in COM itself, it advises developers not to do so anymore.



The new middle-ware is thus faced with a number of requirements. First, a simulation environment needs to be able to transparently load the middle-ware. This means it needs to be located in a DLL (on Windows computers) or shared object (on Mac or linux) that is located in the path. A simulation environment then loads the middle-ware DLL, and obtains access to functions that allows for enumerating the installed client models, e.g. thermo systems and unit operations and such. This requires maintaining a database or registry. New process models need to be able to make themselves available by registering themselves to the middle-ware.

Next, the simulation environment must be able to instantiate or create a client model. To the simulation environment, it should not matter whether the client model is coded in C++, or in .NET, lives on the local machine or somewhere else, etc. After creating a client model, it should be able to obtain the proper interfaces, if implemented by the client model. For example, it should be able to obtain an ICapeUnit interface from a unit operation. In COM it is required to keep track of such interfaces and release them. This should not be required for the new middle-ware, to make it easier on the developer; like the data types, interfaces should clean themselves up. As the client models gets terminated by the simulation environment, all interfaces automatically get cleaned up.

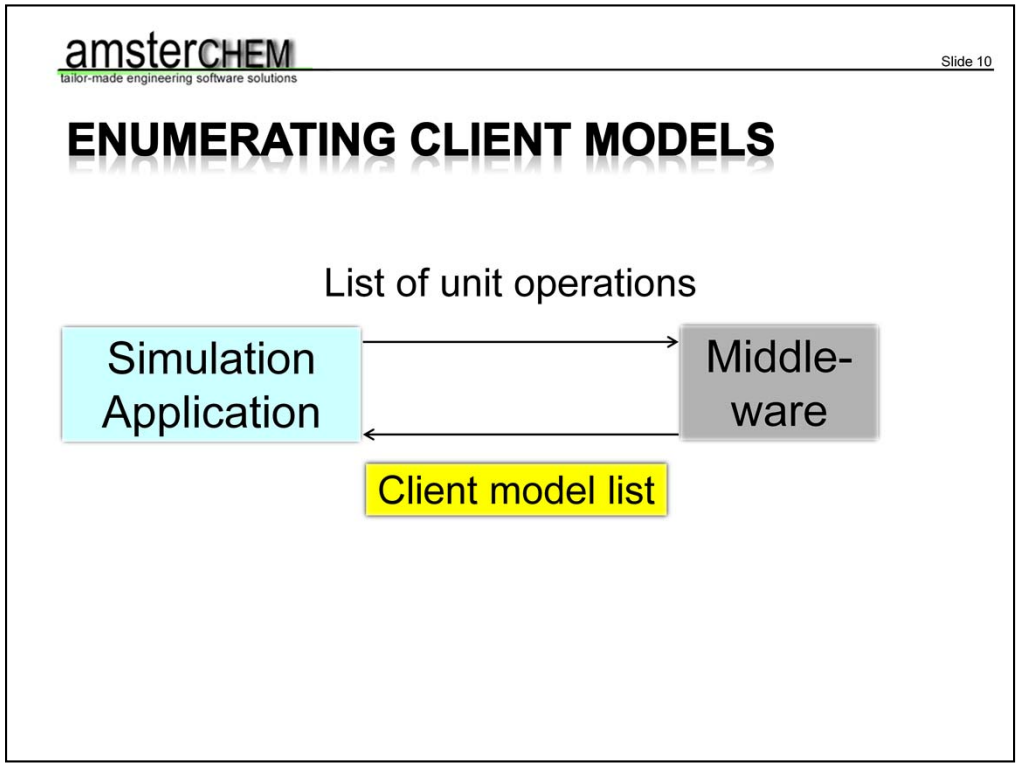
The client model and all of its options should be saved and loaded. A mechanism should be provided for this.

Data types such as arrays should be provided in classes of the proper data type, that do not require explicit releasing.

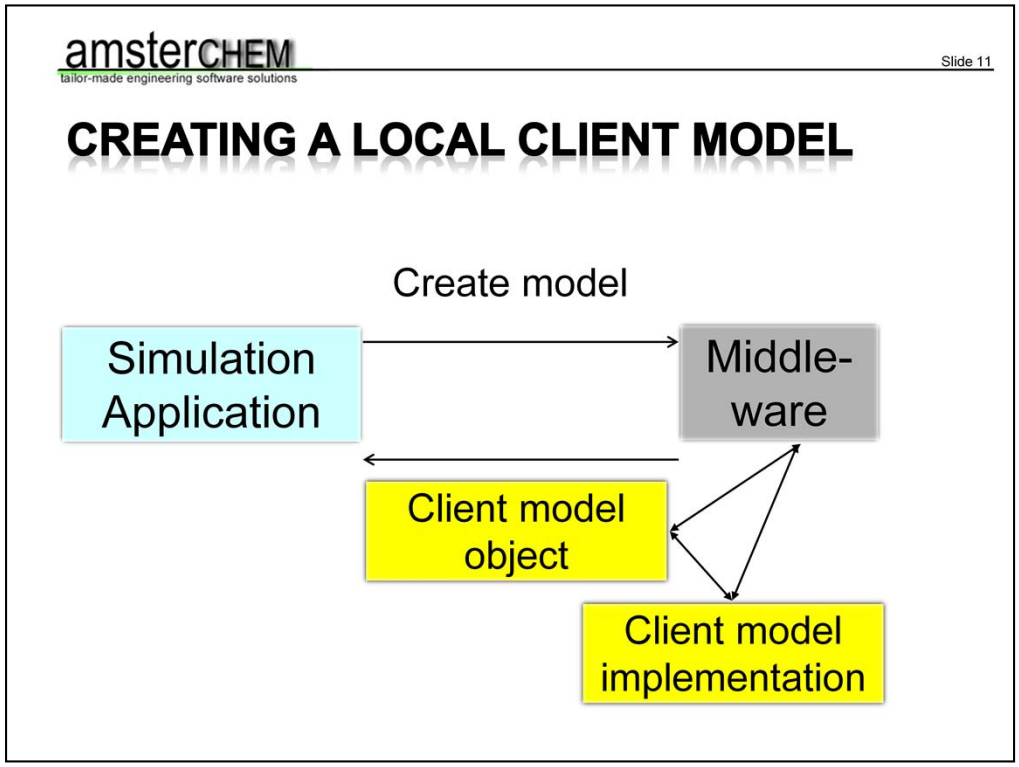
Clients that live on another computer or in another process space should be loaded transparently; this means it is up to the new middle-ware to deal with the inter-computer or inter-process communications.

Last, but not least, we cannot get around the fact that nearly all implementations are currently COM based. So applications that use the new middle-ware should automatically gain access to all COM based client models on the system, via the new middle-ware. Similarly, a new client model that uses the new middle-ware should automatically be made available as COM object to existing simulation environments that are COM based.

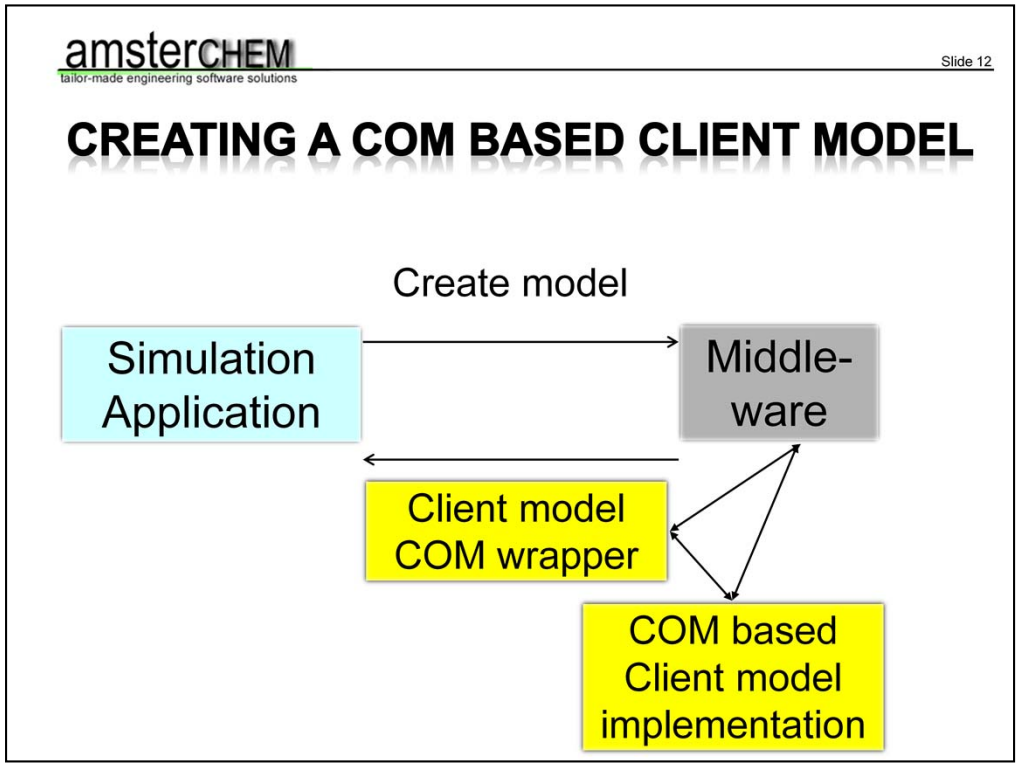
Let us illustrate this with some pictures.



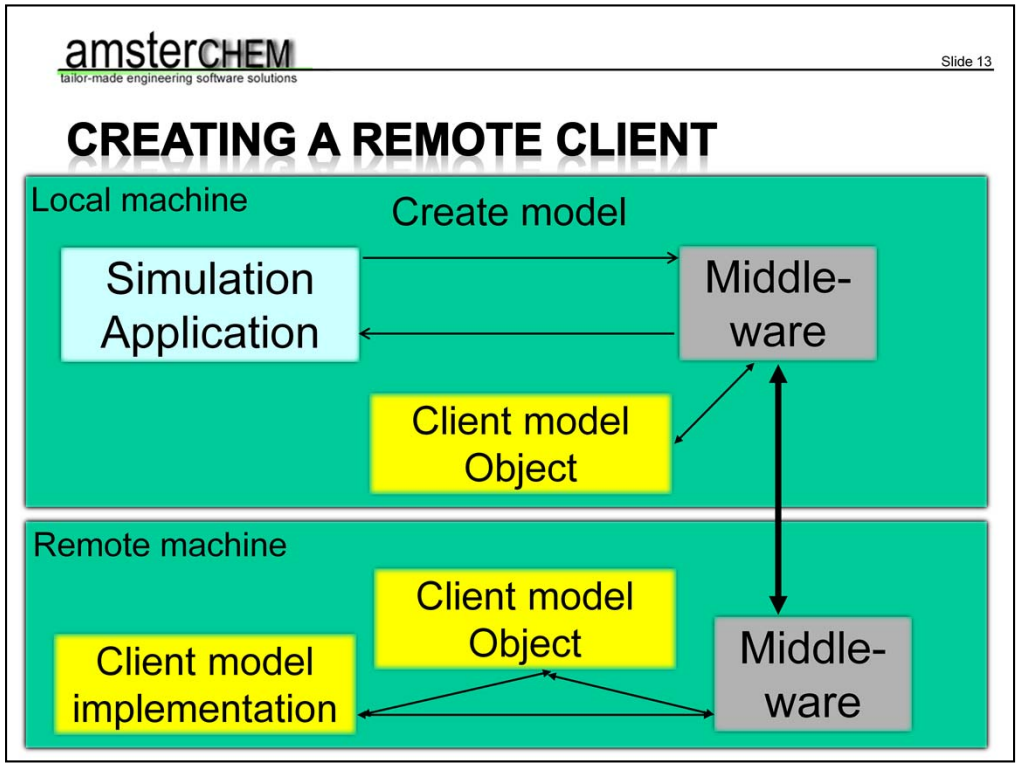
To start, the simulation application asks the middle-ware for a list of installed client models of a particular type, for example unit operations. The middle-ware returns an enumeration class with all locally and remotely registered unit operations, including COM based ones. The simulation application can have the user pick an entry from the list, based upon name and description, and instantiate it.



To create a client model, the simulation application can ask the middle-ware to do so, and the middle-ware returns an object representing the client model. This object will internally communicate with the client model implementation and with the middle-ware (for example to allocate data structures). In effect, the simulation application talks directly to the client model implementation. This is in the case the client model is implemented locally based upon the new middle-ware.

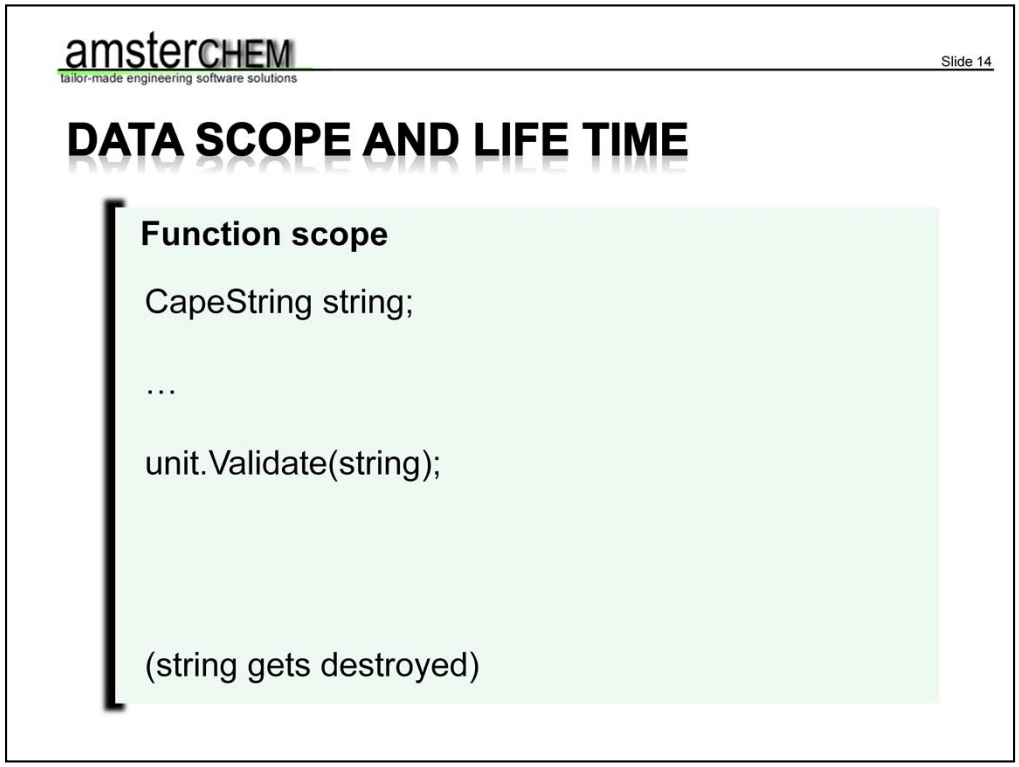


If the client model implementation is actually COM based, this does not matter to the simulation application. The returned object includes all the functionality required to access the COM based object. Some data marshalling is required, for example to convert between the CapeDoubleArray object type and a COM SAFE ARRAY of doubles. This does not occur additional penalty, as now the COM data packing is performed by the middle-ware, whereas previously the same data handling needed to be performed by the simulation application.



To create a model that lives on another computer, the simulation application does exactly the same; it asks the middle-ware to create the model. The middle-ware by means of registration knows that the object lives on another machine. It contacts the middle-ware on that particular machine to create the actual client object, and it returns a client model object that takes care of all communication with the remote machine. All communication thus runs via the middle-ware, and could for example be TCP/IP based.

For creating an out-of-process client model, the mechanism is the same, except the local object and remote object live on the same machine but in a different process space. Each process will load a copy of the middle-ware, and the middle-ware takes care of all communication. Instead of TCP/IP, an inter-process communication mechanism such as messaging can be used. However, TCP/IP would also work fine in this case. This is how CORBA works.



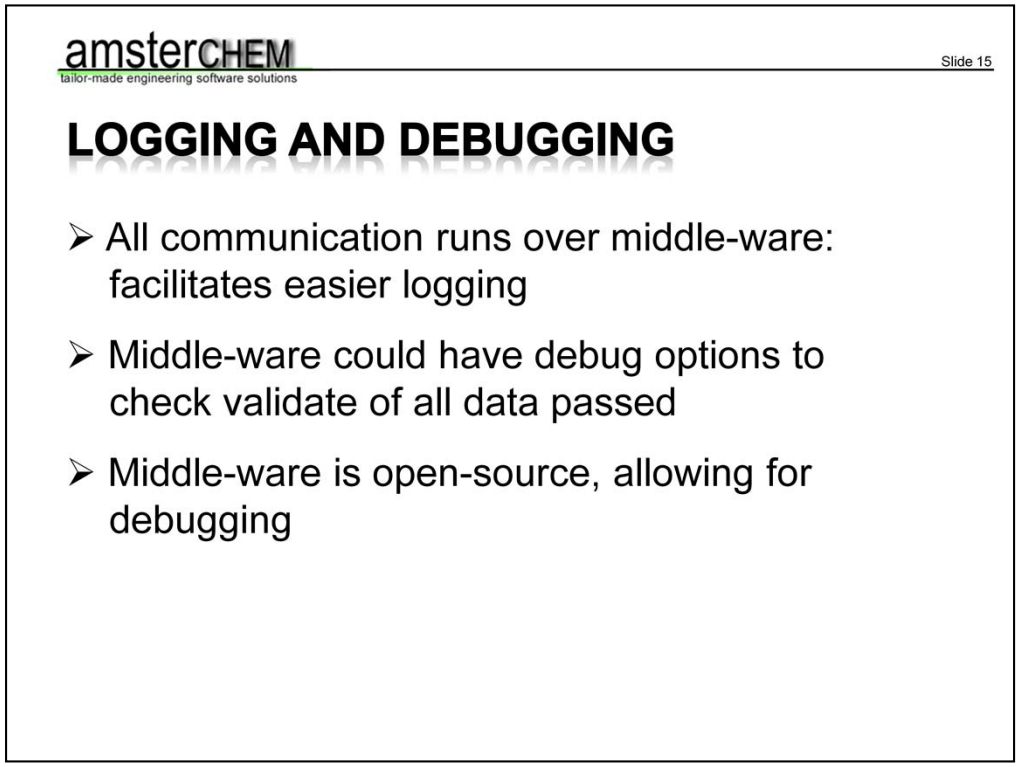
The data scope should be fully automatic. The coder should merely create an object of the proper data type. As an example, this slide depicts the scope of a function. Inside the function, a CapeString variable is declared. This string variable is passed to a CAPE-OPEN function, here for example Validate of an ICapeUnit interface. At the end of the function, the string variable automatically goes out of scope, which takes care of releasing the memory associated with the string data.

It would probably not be efficient to allocate and deallocate all memory in this way, but as it is transparent to the caller, the middle-ware can recycle memory objects as it pleases. For example, it could choose not to destroy the memory allocated with this particular string variable, but re-use it for the next string that gets allocated. How the middle-ware handles data allocation should of course not be a concern of the developer.

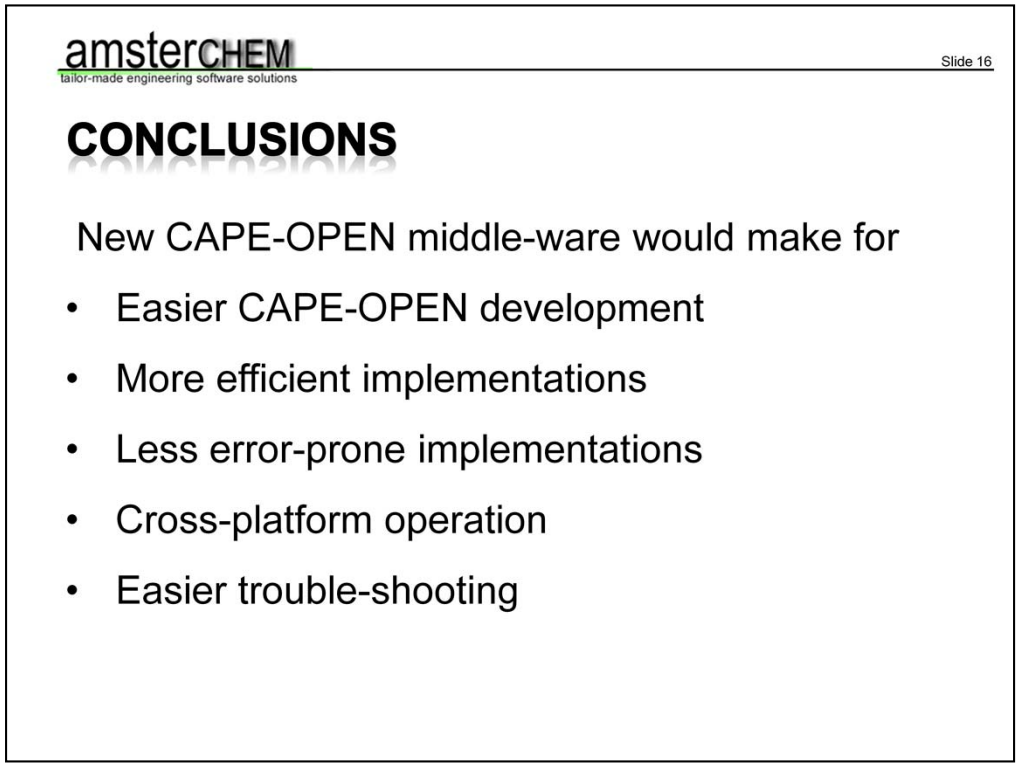
All data that requires allocation can be handled in this way. Quite often, arrays of strings and arrays of double variables are passed with CAPE-OPEN functions. The list of CAPE-OPEN data types is rather limited.

Note that scoping of class variables is required for this principle to work. Hence, object oriented languages such as C++, Fortran 90, java and .NET are more suitable than non-object oriented languages such as standard C or Fortran 77. If required,

wrappers can be written for C or FORTRAN 77.



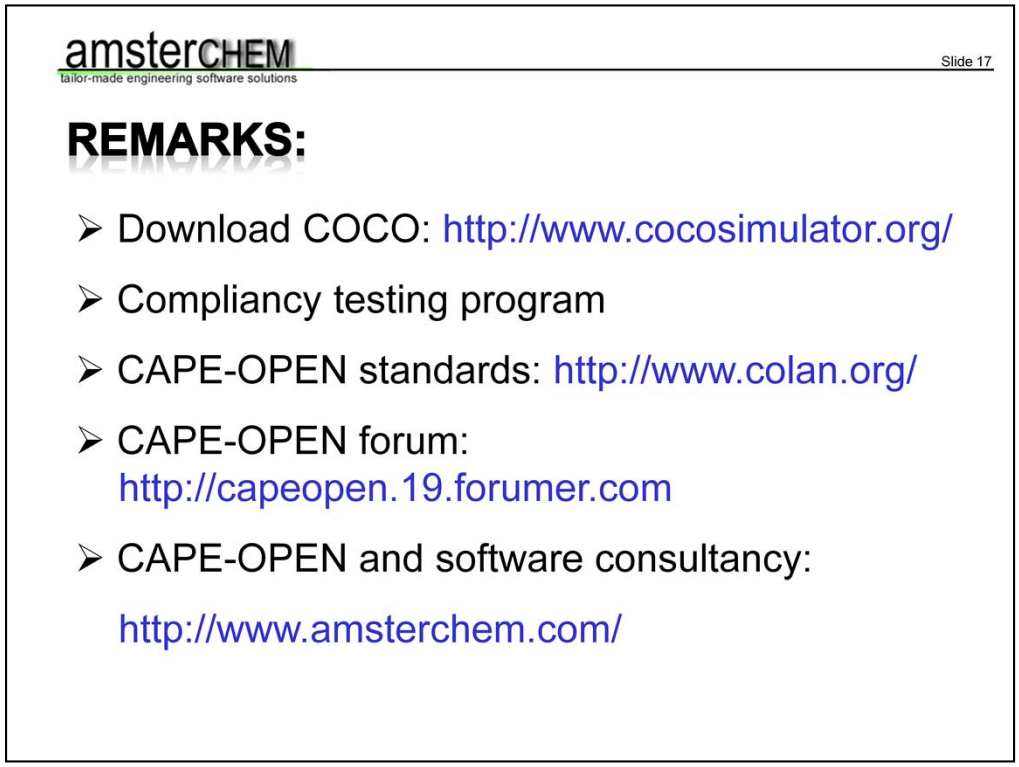
Having control over the middle-ware comes with some additional advantages. We have already seen that it makes for easier development, more targeted and thus more efficient data structures and less change of forgetting to de-allocate memory. In addition, as all communication runs via the middle-ware, the middle-ware itself could implement functionality for logging of the communication between a simulation environment and a client model. It could also allow for turning on some debug options during development, for example options that allow for checking all data for valid content. Finally, as the source code of the middle-ware itself would be available, the middle-ware can directly be debugged by the developers that write CAPE-OPEN software.



Some conclusions:

New middle-ware for CAPE-OPEN implementations would have the following advantages. It would make CAPE-OPEN more accessible for chemical engineers because knowledge of COM programming is no longer required. Because the COM data types would be replaced by more suitable data types, CAPE-OPEN implementations can communicate more efficiently. Also, as the data types will automatically get cleaned up and are bound to be of the correct type, the developer is not bothered with mundane tasks like tracking data ownership and checking data validity, which makes it easier not to make mistakes. CAPE-OPEN implementations that are currently around only run on Windows platforms; with new middle-ware this is no longer a restriction. Should trouble still be present once you are up and running, the open source aspect makes it easier to debug where it goes wrong.

Note that all of this would have to be supplied by the CO-LaN CAPE-OPEN organization, and it a major task. In my view the advantages justify such a task. CAPE-OPEN would not be the first dedicated object model. OpenOffice for example has their own platform independent object model: UNO. This was implemented for good reasons.



Some final remarks: the freeware CAPE-OPEN simulation environment COCO is available for download from cocosimulator.org; if you are currently a CAPE-OPEN developer, you may find this package useful for testing your implementation.

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.

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