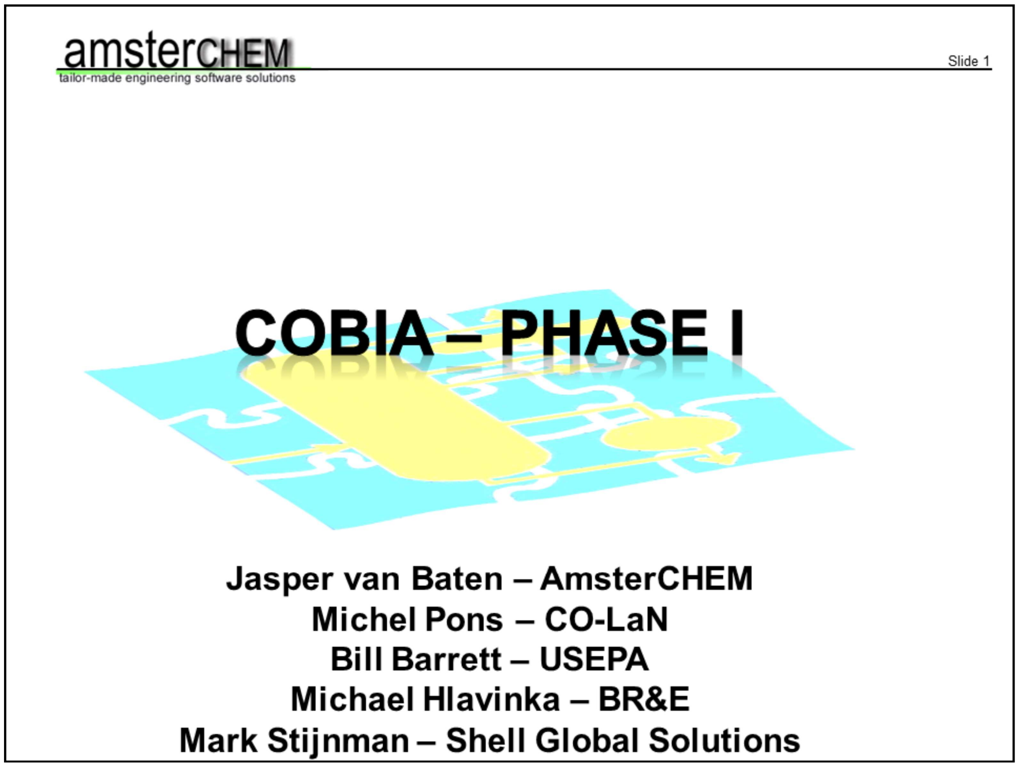
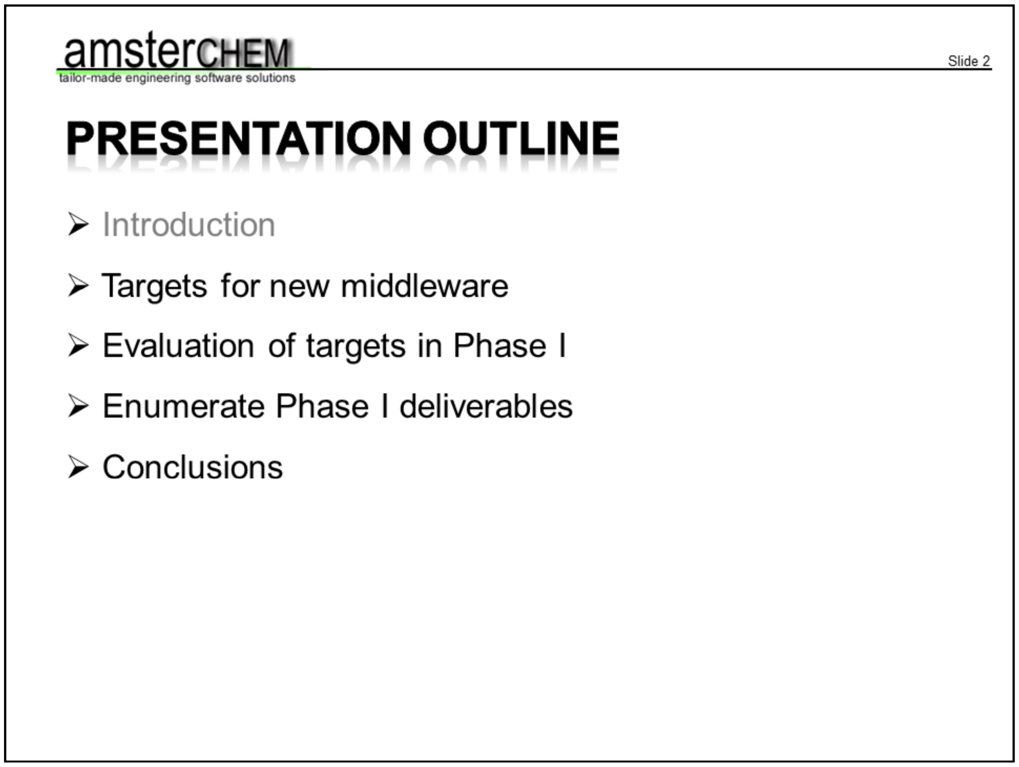


amsterCHEM
tailor-made engineering software solutions

COBIA – PHASE I

Jasper van Baten – AmsterCHEM
Michel Pons – CO-LaN
Bill Barrett – USEPA
Michael Hlavinka – BR&E
Mark Stijnman – Shell Global Solutions

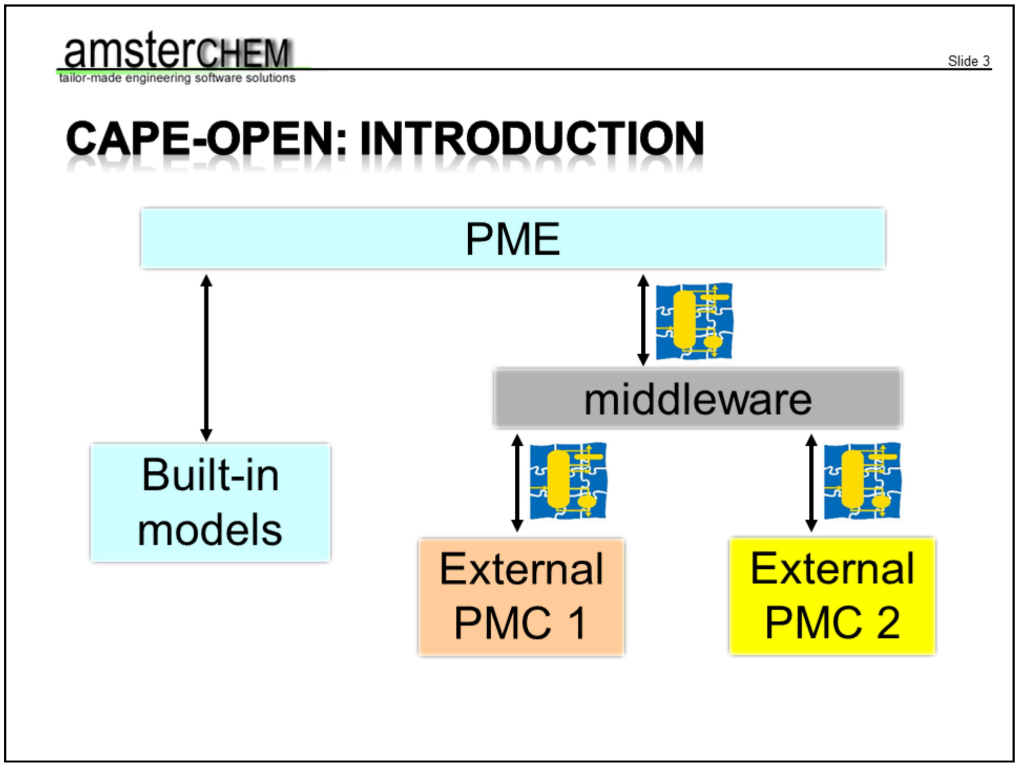

amsterCHEM
tailor-made engineering software solutions

Slide 2

PRESENTATION OUTLINE

- Introduction
- Targets for new middleware
- Evaluation of targets in Phase I
- Enumerate Phase I deliverables
- Conclusions

I will, in this presentation, briefly outline where the middle ware lives in CAPE-OPEN, what we are currently using and how we can improve that. This will lead to a formulation of a list of targets for a new middle ware. From there one, we will evaluate each of these targets in the light of the recent delivery of Phase I of COBIA. I will conclude with an enumeration of what has been delivered for Phase 1 and some final remarks. Warning: this presentation gets somewhat technical at times. Feel free to interrupt and ask questions.



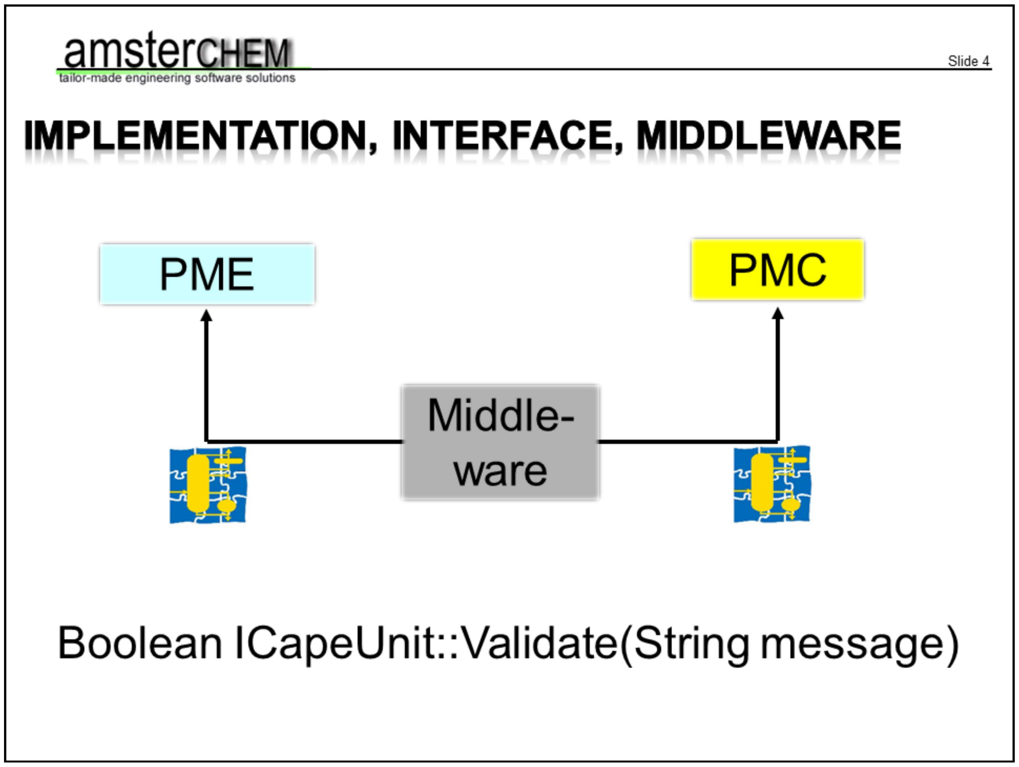
amsterCHEM
tailor-made engineering software solutions

Slide 3

CAPE-OPEN: INTRODUCTION

I will briefly iterate over the context of an object model. This was presented in Berlin, 2011, and Pittsburgh 2012, roughly, and sets the context.

CAPE-OPEN is, in short, a definition of a whole bunch of interfaces that define the functionality exposed by one object and accessed by another object, in terms of which functions are exposed, what are the arguments to these functions and what is expected calling order and behavior of objects that implement CAPE-OPEN interfaces. CAPE-OPEN lives at the boundary between a PME and external modelling components, PMCs. This implies that the PME and PMC can come from different software vendors, that use different compilers, etc. So declaring the interfaces is not enough, we also need to establish binary compatibility between the objects.

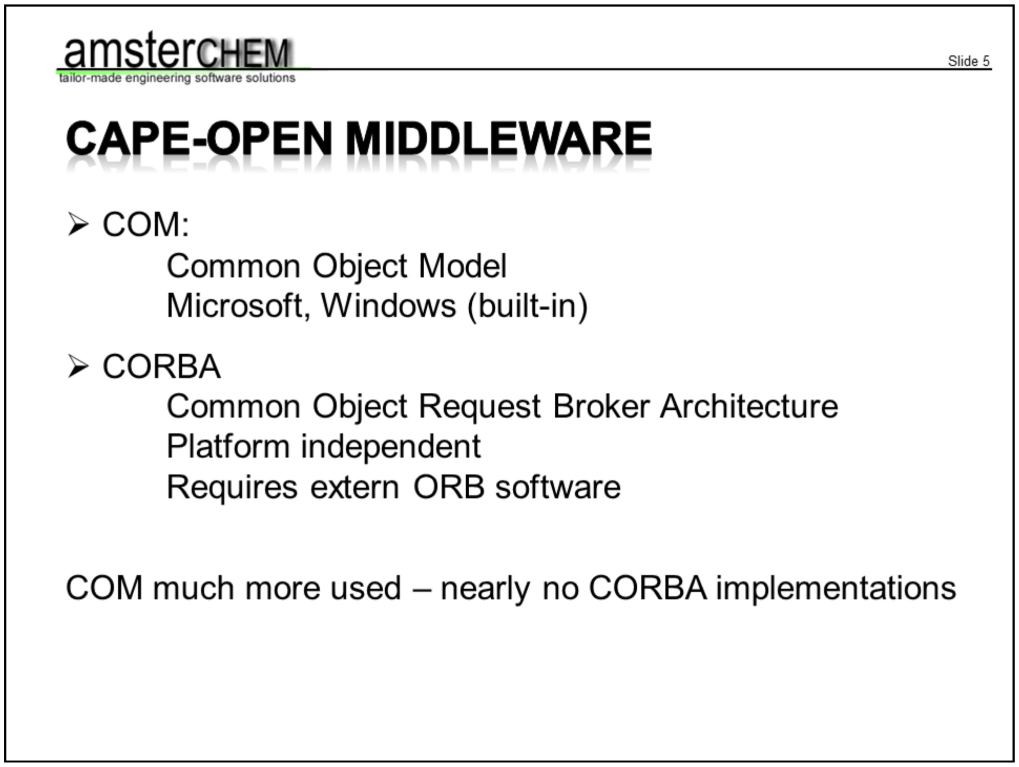

amsterCHEM
tailor-made engineering software solutions
Slide 4

IMPLEMENTATION, INTERFACE, MIDDLEWARE

PME
PMC
Middle-
ware

Boolean ICapeUnit::Validate(String message)

So CAPE-OPEN does not define what the PME or PMC implement exactly, it just defines the interfaces with the functions and function arguments that a PME must supply to a PMC and vice versa. For example, ICapeUnit has a function Validate that returns a Boolean and provide a textual message as output argument. It does not state which tests a unit operation must perform for validation to succeed or fail, that is up to the implementation. The piece that is missing is the binary definition of the string, which in COM is a COM BSTR, the calling convention, the exact data type of the Boolean, in other words, the details that allow for binary compatibility of the PME and PMC. More-over, the middleware also provides the functionality to locate and create PMC objects, defines how to deal with threading models, and defines how a PME and PMC communicate if they are not part of the same process, or architecture.



amsterCHEM
tailor-made engineering software solutions

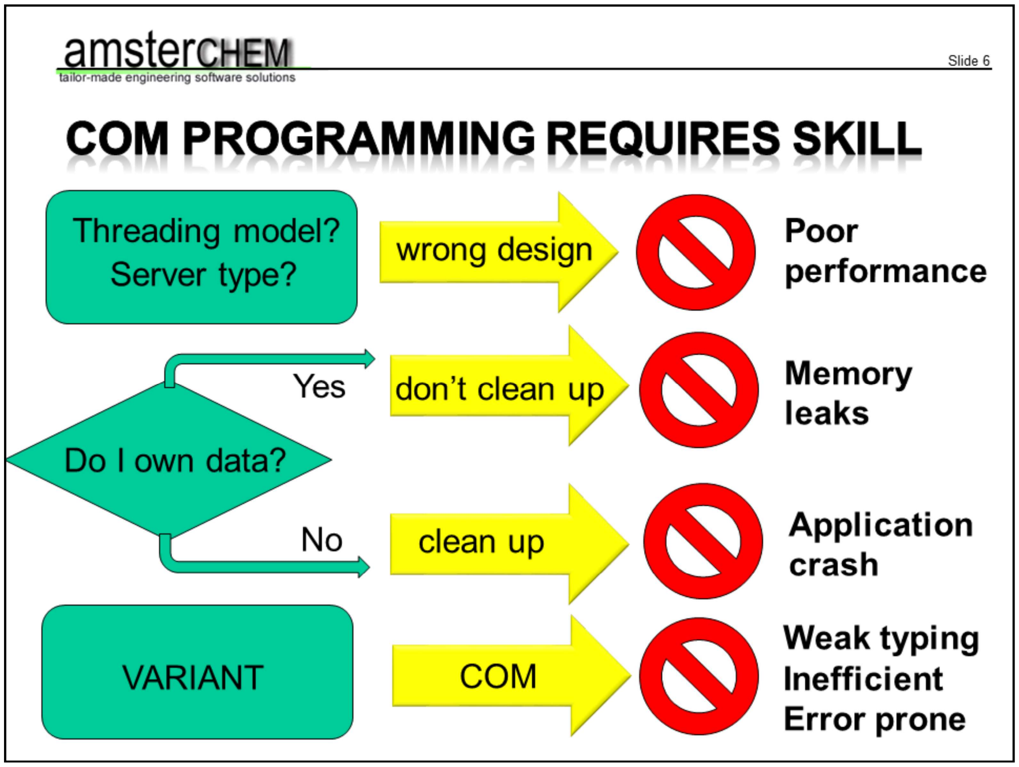
Slide 5

CAPE-OPEN MIDDLEWARE

- COM:
  - Common Object Model
  - Microsoft, Windows (built-in)
- CORBA
  - Common Object Request Broker Architecture
  - Platform independent
  - Requires extern ORB software

COM much more used – nearly no CORBA implementations

Until now, CAPE-OPEN has adopted two middlewares: COM and CORBA. COM is built-in to Microsoft Windows, only available on Windows, and automatically supported if you run Windows as it is built-in to Windows. CORBA is in principle platform independent, once you agree on the implementation; separate Object Request Broker software is required and must be installed. CORBA did not really take off in CAPE-OPEN due to the complexity of setting it up, so COM pretty much became the standard.



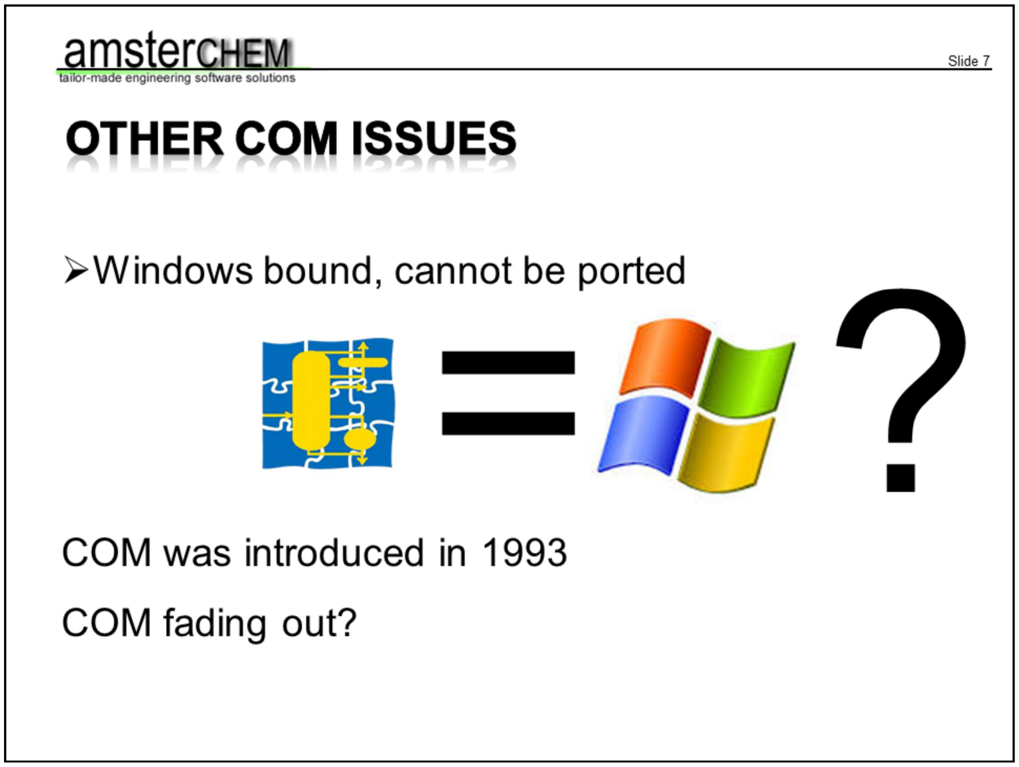
amsterCHEM

tailor-made engineering software solutions

Slide 6

COM PROGRAMMING REQUIRES SKILL

COM is very powerful, it can sing and dance. Using it however implies that a CAPE-OPEN PME must be a COM client, and a PMC must be a COM server, and implementing these requires programming skills. Choices need to be made regarding threading models, server type, etc. The wrong choice may lead to data marshalling, which may be detrimental to performance. Not following the COM rules may lead to memory leaks, crashes, and whatnot. The data in CAPE-OPEN via COM pretty much all goes over the pipeline as VARIANTs, which are generic weak typed data containers, and BSTR strings. Somebody must allocate these and somebody must free these, which is not always obvious. And of course you have to check that the data you are looking at is of the expected type to begin with. Not very efficient, and error prone.



amsterCHEM
tailor-made engineering software solutions

Slide 7

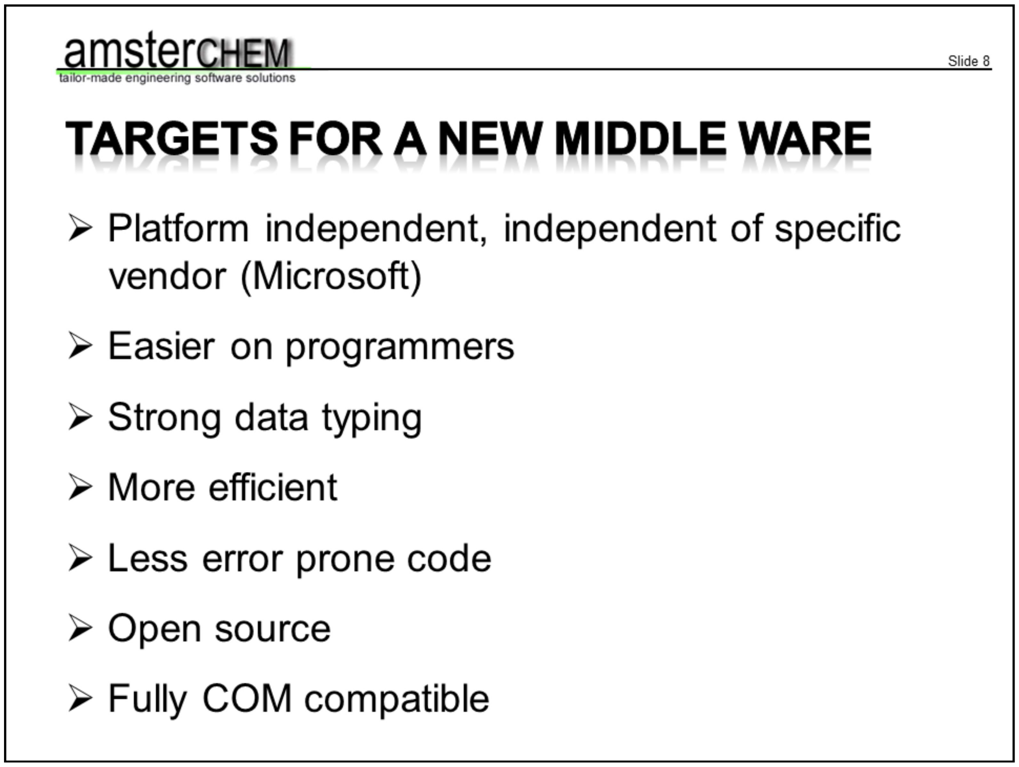
OTHER COM ISSUES

➢Windows bound, cannot be ported

COM was introduced in 1993

COM fading out?

In addition, COM is bound to Windows, and that binds CAPE-OPEN in its current state effectively to Windows. And perhaps it is time to re-evaluate that. COM was introduced 23 years ago. Microsoft announced to retire it sometime ago, and to replace it by .NET. Surely this will not happen, but COM will get the name of ‘old technology’.

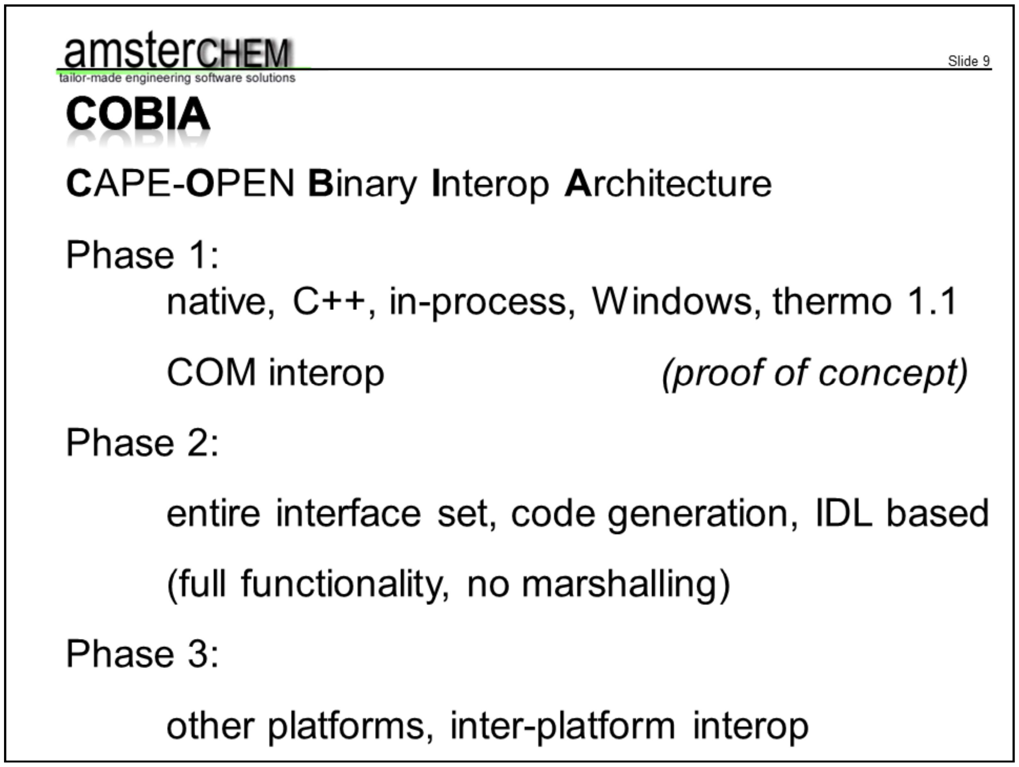

amsterCHEM
tailor-made engineering software solutions

Slide 8

TARGETS FOR A NEW MIDDLE WARE

- Platform independent, independent of specific vendor (Microsoft)
- Easier on programmers
- Strong data typing
- More efficient
- Less error prone code
- Open source
- Fully COM compatible

So these are our targets. Create something that will work on other operating systems, is easier on PME and PMC programmers, has strong and more efficient data typing and is less error prone. As CAPE-OPEN should be truly open, it would naturally need to be open sourced. But we cannot do away with what was built up in the last 15 years, so we need to be a 100% COM compatible. Let us address each of these points in the remainder of this presentation and see where we stand upon completion of COBIA phase I.



amsterCHEM
tailor-made engineering software solutions

Slide 9

COBIA

CAPE-**O**PEN **B**inary **I**nterop **A**rchitecture

Phase 1:

native, C++, in-process, Windows, thermo 1.1

COM interop *(proof of concept)*

Phase 2:

entire interface set, code generation, IDL based

(full functionality, no marshalling)

Phase 3:

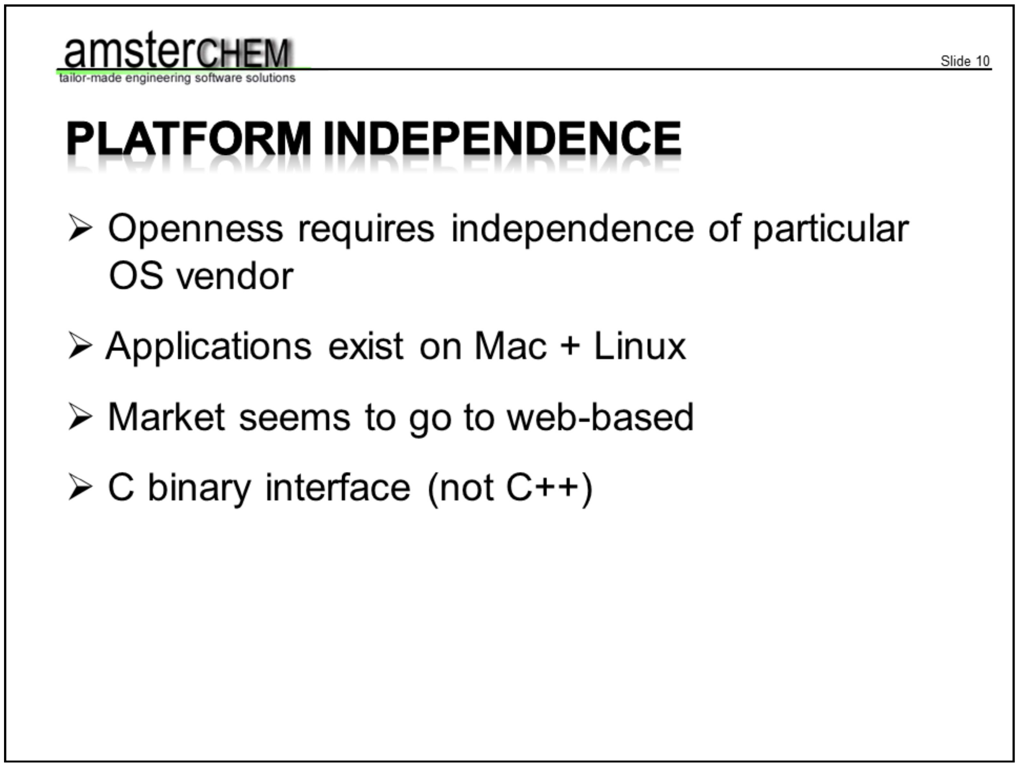
other platforms, inter-platform interop

The M&T sig came up with the name COBIA for the new middleware, which stands for CAPE-OPEN Binary Interop Architecture. Its development is planned in three phases.

Stage I include prototyping for a subset of the interfaces, on Windows only, thermo 1.1 only, with a test PME and test PMC and full COM interop. It is the proof of concept stage and has just been completed.

Stage II extends the functionality to the entire CAPE-OPEN interface set. Still Windows only, and still native only, with C++ as the only language binding. Which covers a good deal of the current CAPE-OPEN application field. At this stage the COBIA IDL will be determined and stub generation and marshalling can use information parsed directly from the IDL.

Stage III will introduce platform independence, and implementations on different platforms talking to each other, via marshaling.



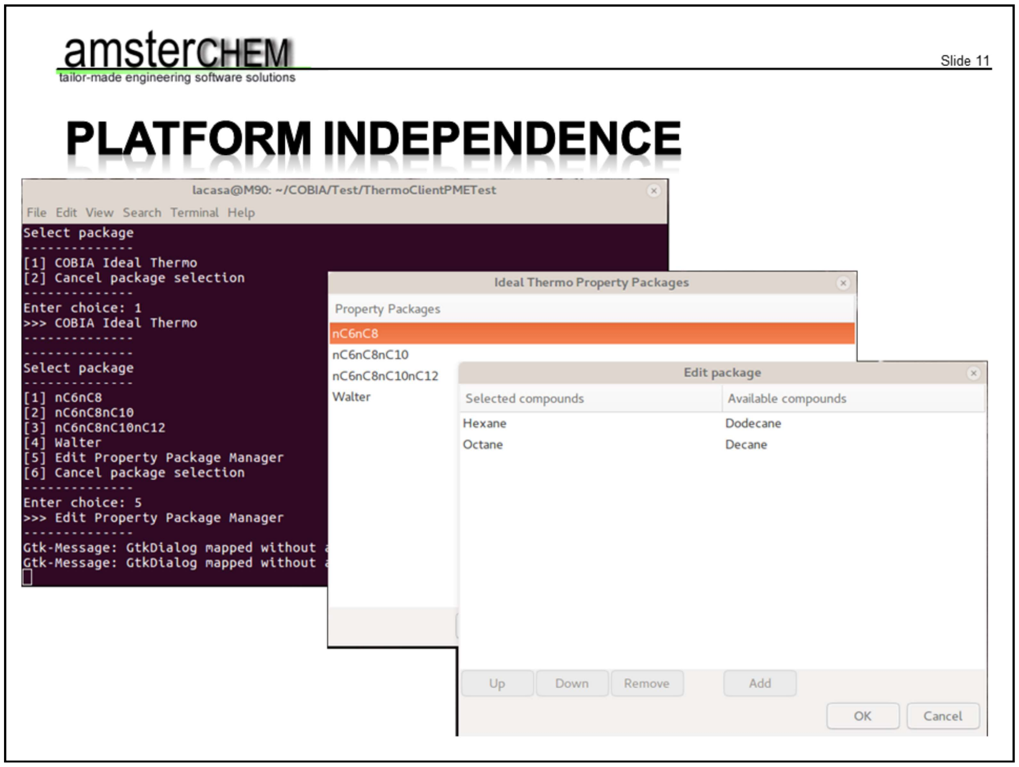
amsterCHEM
tailor-made engineering software solutions
Slide 10

PLATFORM INDEPENDENCE

- Openness requires independence of particular OS vendor
- Applications exist on Mac + Linux
- Market seems to go to web-based
- C binary interface (not C++)

Platform independence may seem to require a business case, which may be hard to demonstrate considering that most of us are using Windows based platforms for our process simulations. There is however a very strong argument for it: you cannot claim to be open if you depend on a particular OS vendor who's software is not open. There are practical advantages too of course. We are not all bound to Windows, we have both PMC and PME vendors that can operate on for example Apple or Linux platforms. And the market is shifting, why not have web based simulations where the PME runs on a web server and the client just has a tablet. In this cases, chances are that the web server is a unix or linux host.

Platform independence is easy to obtain: all interfaces have been written down as C. Not C++. C++ does not have single binary standard, but C has. And this binary standard is understood by virtually all compiler platforms.



amsterCHEM
tailor-made engineering software solutions

Slide 11

PLATFORM INDEPENDENCE

lacasa@M90: ~/COBIA/Test/ThermoClientPMETest

```
Select package
-------------
[1] COBIA Ideal Thermo
[2] Cancel package selection
-------------
Enter choice: 1
>>> COBIA Ideal Thermo
-------------
-------------
Select package
-------------
[1] nC6nC8
[2] nC6nC8nC10
[3] nC6nC8nC10nC12
[4] Walter
[5] Edit Property Package Manager
[6] Cancel package selection
-------------
Enter choice: 5
>>> Edit Property Package Manager
-------------
Gtk-Message: GtkDialog mapped without a
Gtk-Message: GtkDialog mapped without a
```

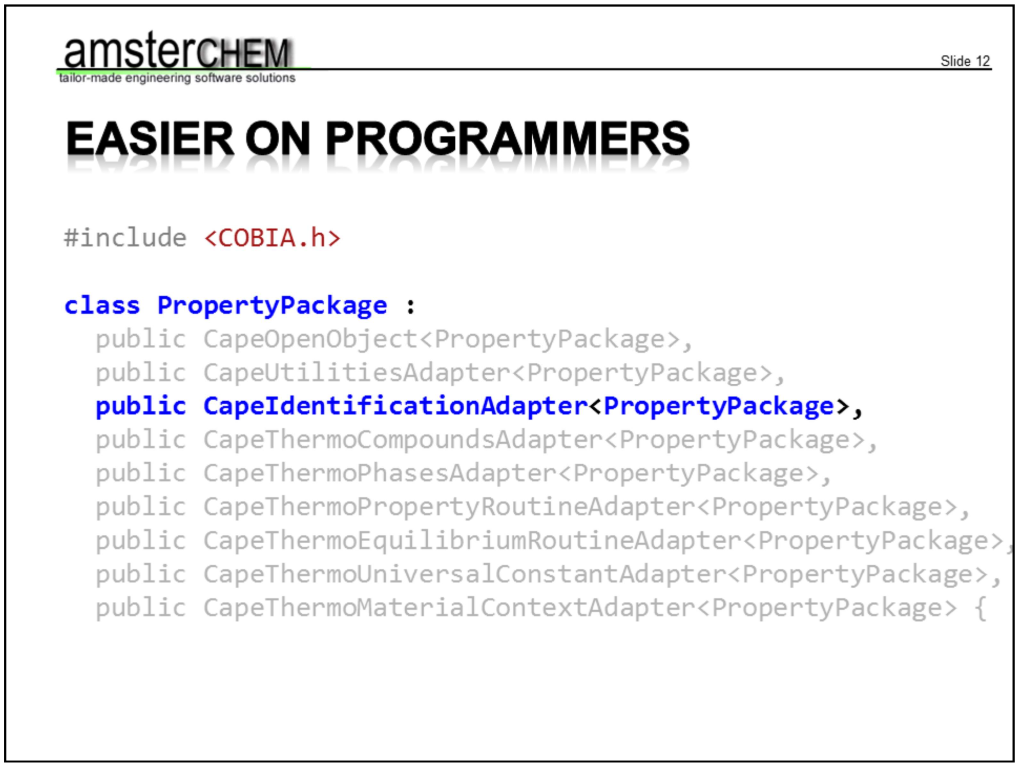
Ideal Thermo Property Packages

Property Packages
nC6nC8
nC6nC8nC10
nC6nC8nC10nC12
Walter

Edit package

Selected compounds	Available compounds
Hexane	Dodecane
Octane	Decane

Although not part of the Phase I deliverables, platform independence is easily demonstrated. These are snapshots of the text based COBIA test PME and the COBIA Ideal Thermo Test PMC, which are part of the deliverables, running on Ubuntu Linux. Not under WINE, the Windows emulator, but simply natively compiled for Linux. A GTK basis was used for the Edit interface of the Property Package Manager and Property Package, but clearly CAPE-OPEN does not decide on how to implement the windowing.

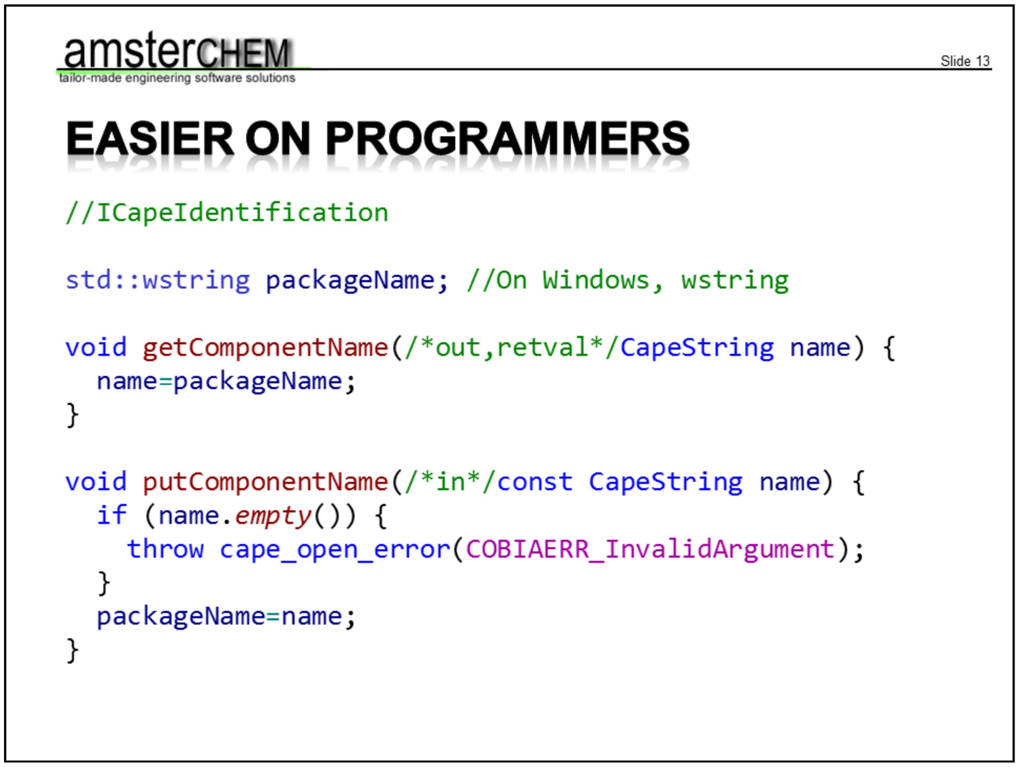

amsterCHEM
tailor-made engineering software solutions
Slide 12

EASIER ON PROGRAMMERS

```
#include <COBIA.h>

class PropertyPackage :
  public CapeOpenObject<PropertyPackage>,
  public CapeUtilitiesAdapter<PropertyPackage>,
  public CapeIdentificationAdapter<PropertyPackage>,
  public CapeThermoCompoundsAdapter<PropertyPackage>,
  public CapeThermoPhasesAdapter<PropertyPackage>,
  public CapeThermoPropertyRoutineAdapter<PropertyPackage>,
  public CapeThermoEquilibriumRoutineAdapter<PropertyPackage>,
  public CapeThermoUniversalConstantAdapter<PropertyPackage>,
  public CapeThermoMaterialContextAdapter<PropertyPackage> {
```

Easier on programmers I can only demonstrate for C++ at the moment, as that is the only language binding that so far has been provided. To use COBIA, you include COBIA.h. This will hide all details of the binary interface, and provide a bunch of C++ wrapper classes to deal with all data and all interfaces. Any programmer in this room of course knows that C++ is object oriented, and this implies that any variable that is defined as a class, comes with a constructor and a destructor, and the C++ compiler will take care of calling the destructors for you. This is used to clean up whatever we are doing, so as opposed to the COM interface, you typically do not have to take care who owns what object or who owns what data. Surely most C++ programmers will use smart pointers for COM objects already, but not so for the VARIANT and BSTR arguments to the methods. You need to know who owns the data, who allocates, and who cleans up. Not in COBIA's C++ binding. All data are simply represented, essentially, by smart pointers around interfaces to the data. In a moment I will demonstrate how this affects efficiency, but for now let's see what it does to programming.

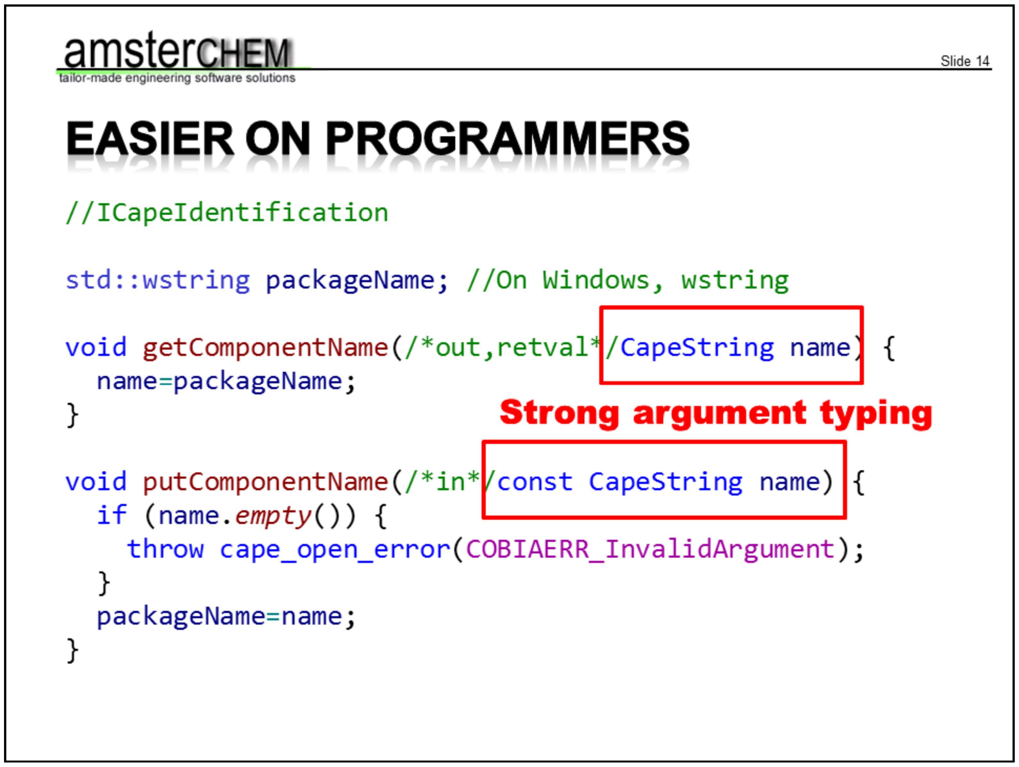
Sorry to have to bother you with code, but we cannot really get around that when talking about programming. Here's the initial bit of the test property package class. The class is defined as ant C++ class, but derives directly from a class that represents each implemented CAPE-OPEN interface. The corresponding methods need to be implemented of course. The base class in bold is ICapeIdentification. Let's see how it is implemented.


amsterCHEM
tailor-made engineering software solutions
Slide 13

EASIER ON PROGRAMMERS

```
//ICapeIdentification

std::wstring packageName; //On Windows, wstring

void getComponentName(/*out,retval*/CapeString name) {
  name=packageName;
}

void putComponentName(/*in*/const CapeString name) {
  if (name.empty()) {
    throw cape_open_error(COBIAERR_InvalidArgument);
  }
  packageName=name;
}
```

The arguments are strongly typed, and in this case are CapeString, for both input and output.


amsterCHEM
tailor-made engineering software solutions

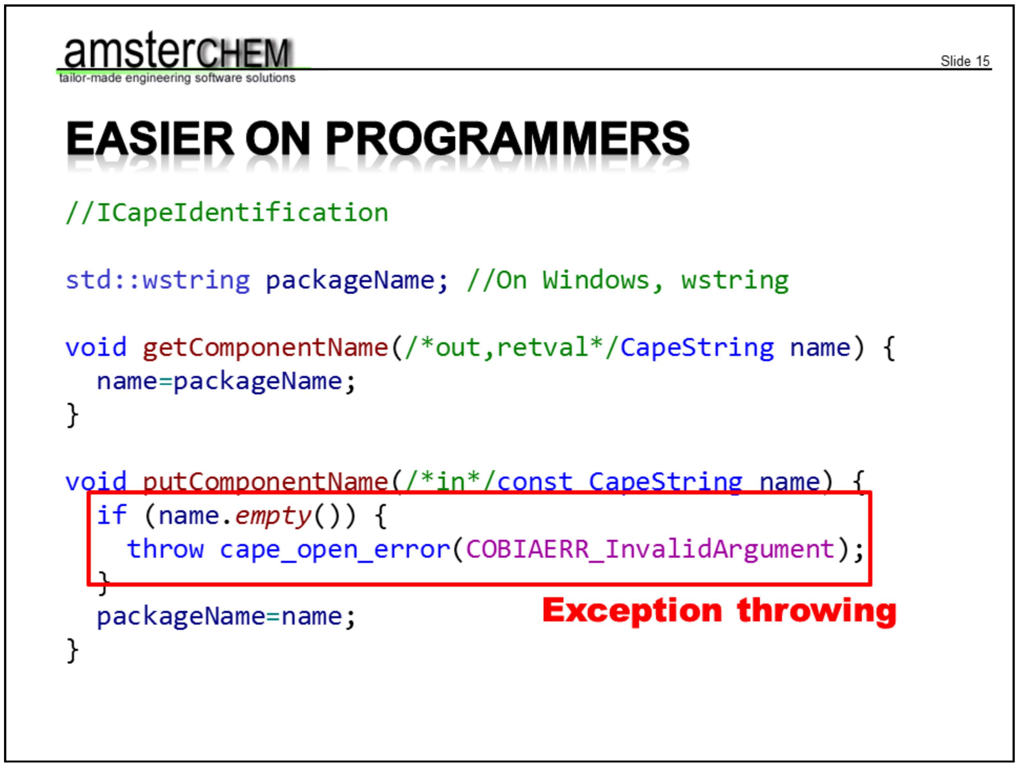
Slide 14

EASIER ON PROGRAMMERS

```
//ICapeIdentification

std::wstring packageName; //On Windows, wstring

void getComponentName(/*out,retval*/CapeString name) {
  name=packageName;
}

void putComponentName(/*in*/const CapeString name) {
  if (name.empty()) {
    throw cape_open_error(COBIAERR_InvalidArgument);
  }
  packageName=name;
}
```

Strong argument typing

They are not COM specific types such as BSTR, or VARIANT. Furthermore, they are C++ classes that wrap interfaces, so we do not need to complicated things about allocating data, or deallocating data. Common functionality of any string class is implemented, such as assignment, checking for empty, etc. The class is a wrapper around an interface, and because the data is owned by the caller, allocation and deallocation of the interface are the responsibility of the caller. The caller can use off-the-shelf wrapper classes for this, as shown later in this presentation, which take care of the life time of the interface. Therefore, there is no need to reference count this interface, it can just be used without any concerns. The class that wraps it has only one data member: the interface pointer. We can therefore safely pass the class by value instead of by reference.


amsterCHEM
tailor-made engineering software solutions

Slide 15

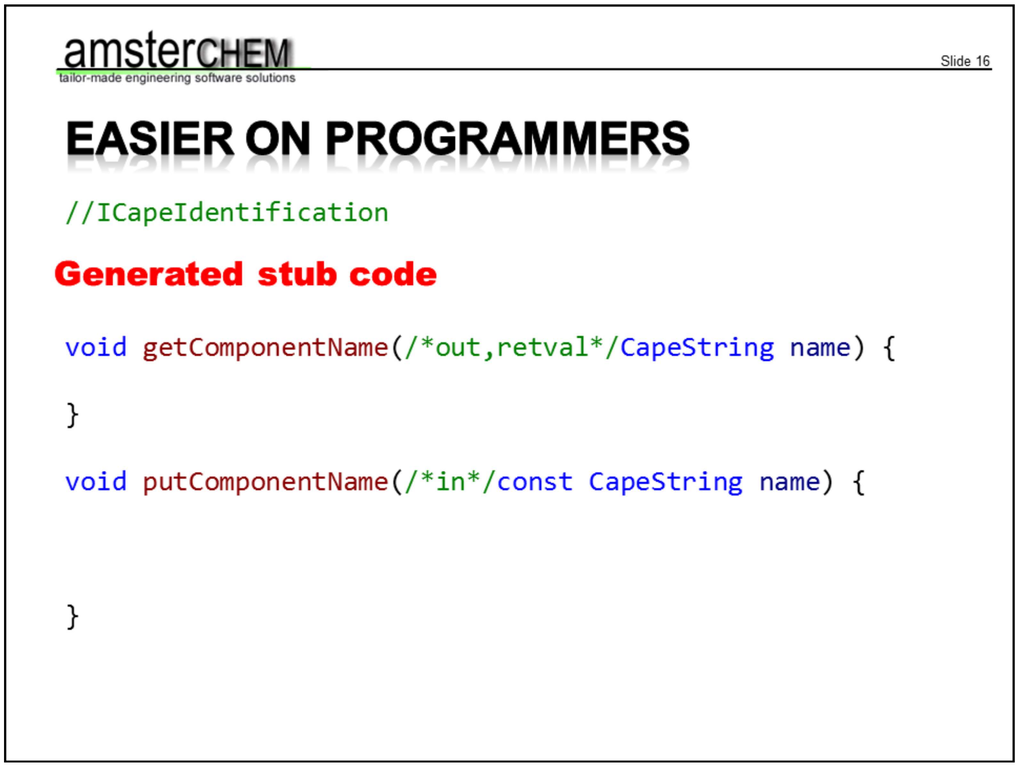
EASIER ON PROGRAMMERS

```
//ICapeIdentification

std::wstring packageName; //On Windows, wstring

void getComponentName(/*out,retval*/CapeString name) {
  name=packageName;
}

void putComponentName(/*in*/const CapeString name) {
  if (name.empty()) {
    throw cape_open_error(COBIAERR_InvalidArgument);
  }
  packageName=name;
}
```

Exception throwing

Note that this package does not accept an empty name. It simply throws an exception, which is defined in COBIA.h. The CapeIdentificationAdapter that we saw on the previous page, catches the C++ specific exception and translates it into a valid CAPE-OPEN error.

Please notice again the absence of any allocation or deallocation. Same holds for double arrays, etc. Everything in the C++ binding of COBIA is a smart pointer to an interface. All of that is defined in, or included via, COBIA.h.


amsterCHEM
tailor-made engineering software solutions

Slide 16

EASIER ON PROGRAMMERS

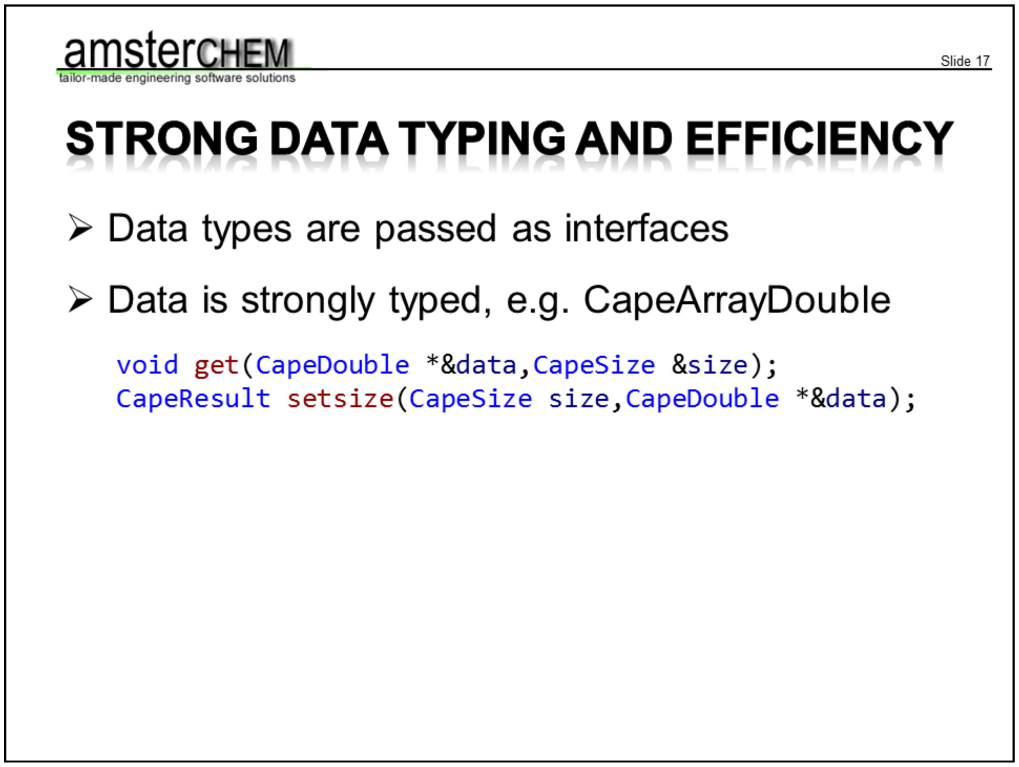
```
//ICapeIdentification
```

Generated stub code

```
void getComponentName(/*out,retval*/CapeString name) {

}

void putComponentName(/*in*/const CapeString name) {


}
```

Of course you want to have some mechanism that declares your class as seen on the previous slide, and that writes down the function definitions as seen on this slide of all methods that require implementation, with the proper arguments. This is called STUB code generation, and part of Phase II. The STUB code to be generated is along the lines of the code that was prototyped in Phase I.

PMC registration, enumeration and instantiation has also been made very simple. If you are interested, please study the test PMC and test PME that were part of the Phase I deliverables.


amsterCHEM
tailor-made engineering software solutions

Slide 17

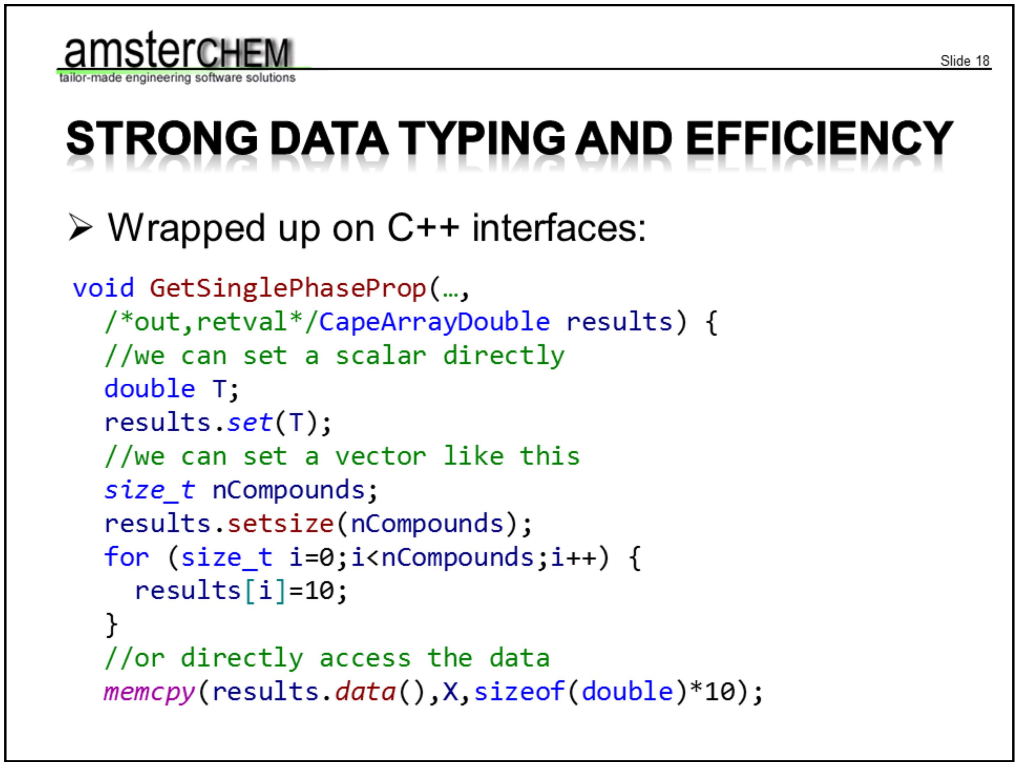
STRONG DATA TYPING AND EFFICIENCY

- Data types are passed as interfaces
- Data is strongly typed, e.g. CapeArrayDouble

```
void get(CapeDouble *&data,CapeSize &size);
CapeResult setsize(CapeSize size,CapeDouble *&data);
```

Basic data types like CapeInteger, CapeDouble, CapeBoolean are passed directly by reference. Other data types that require memory management are passed as interfaces. The original idea was for the underlying data to be managed by the middleware, in one central place, where we can implement per-thread memory caching pools to speed things up. The PME and PMC could then just request an array of particular size, and the middleware would take care of all allocations and deallocations, by reference counting. Mark however had a better idea, and we followed that route instead. All datatypes are actually interfaces that allow read and write access to the memory of the caller directly. Much like passing a vector by reference in C++, but then in a system and implementation independent form.

The promise was to make data access both easier, less error prone and more efficient, so let us have a look on how it is done. Firstly, all data is strongly typed. In COM, GetSinglePhaseProperty passes the data in a VARIANT, which contains a SafeArray of type double. The VARIANT can contain anything and everything, so the sender of the data must go through some steps to properly allocate this, and the receiver must to type checking and a bunch of additional steps to access the array data. In COBIA, what goes over the pipe line is an interface to the data, with essentially two members: a member to change the length of the array, and a member to access the data content and its length.


amsterCHEM
tailor-made engineering software solutions

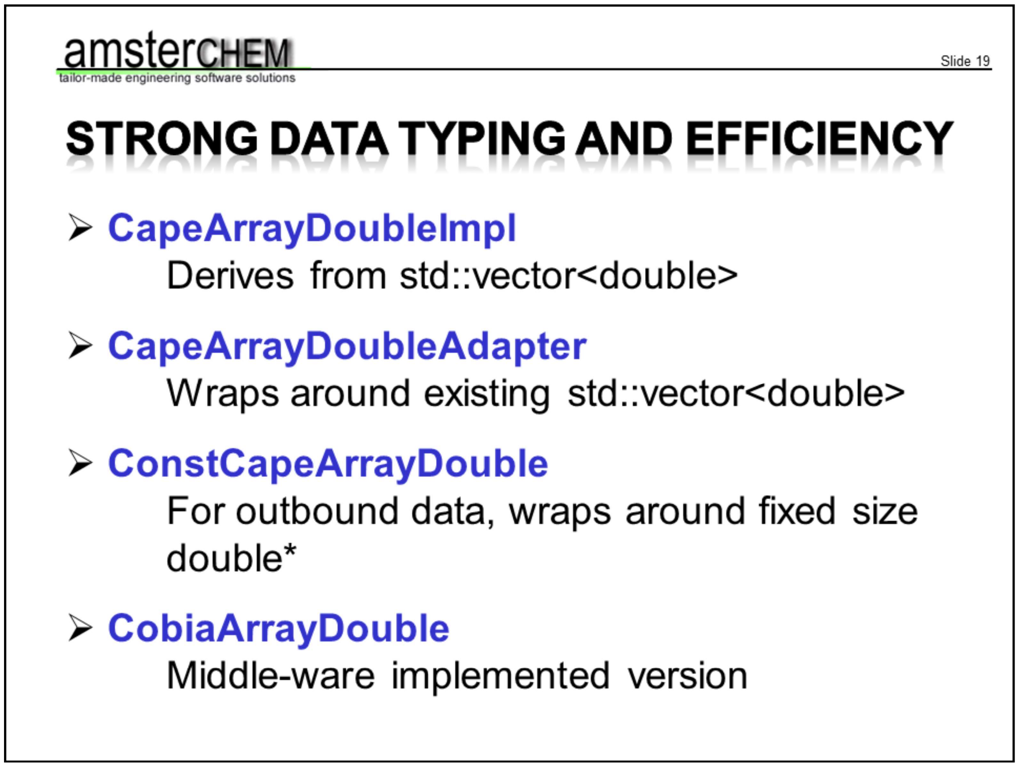
Slide 18

STRONG DATA TYPING AND EFFICIENCY

➢ Wrapped up on C++ interfaces:

```
void GetSinglePhaseProp(…,
  /*out,retval*/CapeArrayDouble results) {
  //we can set a scalar directly
  double T;
  results.set(T);
  //we can set a vector like this
  size_t nCompounds;
  results.setsize(nCompounds);
  for (size_t i=0;i<nCompounds;i++) {
    results[i]=10;
  }
  //or directly access the data
  memcpy(results.data(),X,sizeof(double)*10);
```

The details of the interface are hidden to the programmer by the language binding. In C++ this is naturally wrapped up into a class that will take care of the reference counting, and provides functionality that you would expect from an array class, such as an indexing operator, a size() operator, etc. A function that populates the array, which is passed as argument to the function, must set the size of the array and the data of the array. There is a special shortcut for setting a scalar value. For vector data, the size is set, and the content can be set via an indexing operator or directly via the data pointer, as shown in this example slide.

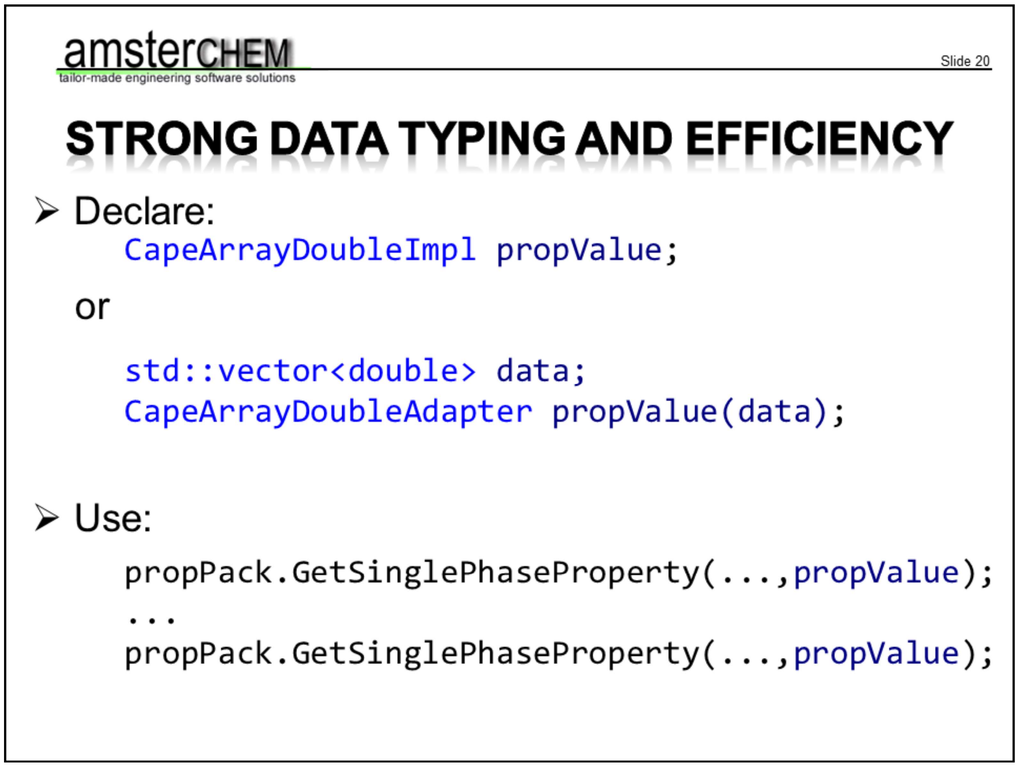

amsterCHEM
tailor-made engineering software solutions
Slide 19

STRONG DATA TYPING AND EFFICIENCY

- **CapeArrayDoubleImpl**
  Derives from std::vector<double>
- **CapeArrayDoubleAdapter**
  Wraps around existing std::vector<double>
- **ConstCapeArrayDouble**
  For outbound data, wraps around fixed size double*
- **CobiaArrayDouble**
  Middle-ware implemented version

This implies that the caller must provide a class that actually stores the data, and implements the interface to the data. Four basic classes are provided for the C++ language binding. The first copy derives from a std::vector and can be used as a normal standard vector. The second copy assumes you have std::vector already somewhere in your code and puts an interface around that. The third copy assumes that the data is outbound and not inbound, so the other side will not try to resize the data or write to the data. In this case a double pointer with fixed size can be used directly. The resize function of this implementation raises an error of course. The last implementation is for completeness, and an array implemented by the middleware. This is not likely to be used by C++ applications, but other languages may perhaps benefit from it.

Of course you can provide your own implementation around your own data if you want. For example the COM wrappers that I will discuss in a moment implement a data type that wraps directly around a VARIANT and accesses directly the VARIANT data.


amsterCHEM
tailor-made engineering software solutions

Slide 20

STRONG DATA TYPING AND EFFICIENCY

- Declare:

```
CapeArrayDoubleImpl propValue;
```

or

```
std::vector<double> data;
CapeArrayDoubleAdapter propValue(data);
```

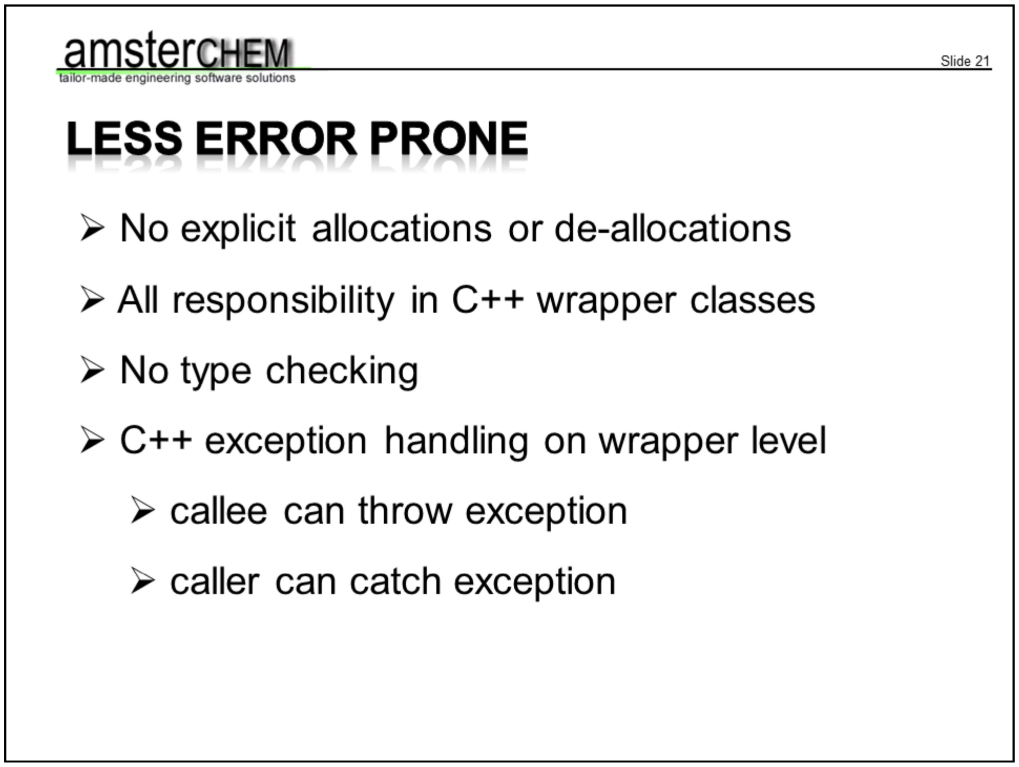
- Use:

```
propPack.GetSinglePhaseProperty(...,propValue);
...
propPack.GetSinglePhaseProperty(...,propValue);
```

So this is how it works. We can declare either of the array data types. The first one implies the std::vector, the second one uses an explicitly declared one.

The resulting variable implements the ICapeArrayDouble interface, so we can simply pass it to an external function. This function will then write data to our array. We can re-use the variable at will. C++ programmers will know that a std::vector only re-allocates itself in case it grows, not in case it shrinks. So this approach will pretty much lead to eliminating all memory allocations altogether, except from a few initial ones.

For SetSinglePhaseProperty, we do not expect the callee to write to the data, so we can pass a wrapper around any non-resizable double array.


amsterCHEM
tailor-made engineering software solutions

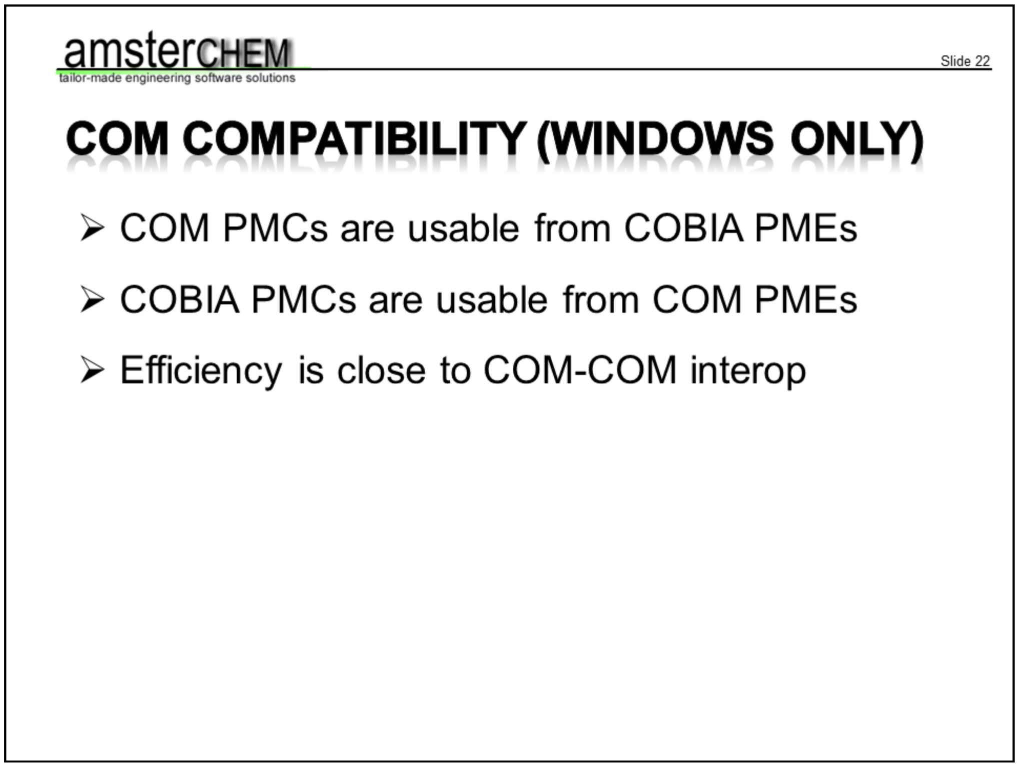
Slide 21

LESS ERROR PRONE

- No explicit allocations or de-allocations
- All responsibility in C++ wrapper classes
- No type checking
- C++ exception handling on wrapper level
  - callee can throw exception
  - caller can catch exception

Clearly CAPE-OPEN programming becomes easier this way. Especially if we integrate the wizards that will create the CAPE-OPEN object framework for you. But it also becomes less error prone. You may have noticed the absence of any explicit allocations, deallocations, AddRef, Release etc, in the previous slide. All that responsibility has been taken to C++ wrapper classes. As all data is strongly typed, you no longer have to type check your data, and it is not possible to make any wrong assumptions there.

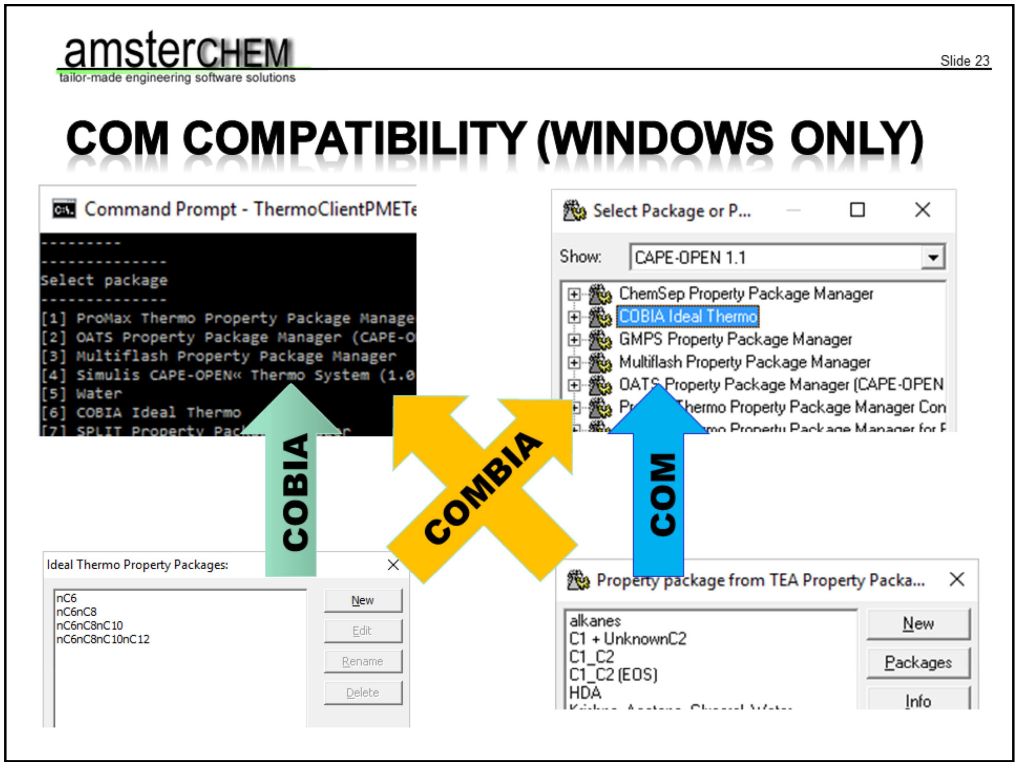
Also error handling is easier: any method can simply throw a CAPE-OPEN specific exception. The C++ wrapper code will catch this exception, and translate it into a CAPE-OPEN error. The C++ wrapper code on the caller side will check for this error and translate it back into an exception. In other words: the C++ language binding can simply use exceptions. Other language bindings that support exceptions can do the same.


amsterCHEM
tailor-made engineering software solutions
Slide 22

COM COMPATIBILITY (WINDOWS ONLY)

- COM PMCs are usable from COBIA PMEs
- COBIA PMCs are usable from COM PMEs
- Efficiency is close to COM-COM interop

As nearly all current implementations are COM based, and we cannot expect everybody to migrate at the same time, COM-COBIA interop is essential, and was made part of COBIA Phase 1. COM PMCs can transparently be used from COBIA PME's, of course only on Windows. Similarly, COBIA PMCs can be used transparently from COM PMEs. At no point will there be any need for any implementation to provide a double interface. And even better, as we have seen the data access in COBIA is set up such that we can directly read from and write to the caller's variable, COM-COBIA interop is by good approximation as efficient as COM-COM interop.


amsterCHEM
tailor-made engineering software solutions
Slide 23

COM COMPATIBILITY (WINDOWS ONLY)

Command Prompt - ThermoClientPMETe

```
---------
-------------
Select package
-------------
[1] ProMax Thermo Property Package Manage
[2] OATS Property Package Manager (CAPE-O
[3] Multiflash Property Package Manager
[4] Simulis CAPE-OPEN« Thermo System (1.0
[5] Water
[6] COBIA Ideal Thermo
[7] SPLIT Property Pack
```

Select Package or P...

Show: CAPE-OPEN 1.1

- ChemSep Property Package Manager
- COBIA Ideal Thermo
- GMPS Property Package Manager
- Multiflash Property Package Manager
- OATS Property Package Manager (CAPE-OPEN
- Pr... Thermo Property Package Manager Con

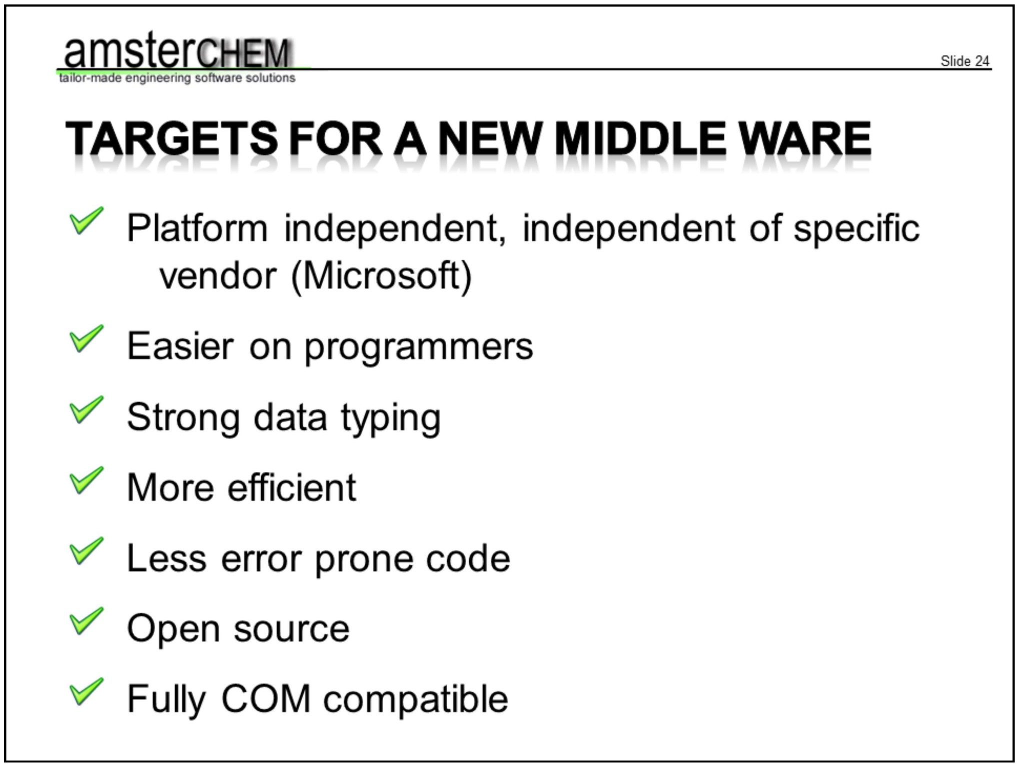
Ideal Thermo Property Packages:

- nC6
- nC6nC8
- nC6nC8nC10
- nC6nC8nC10nC12

Property package from TEA Property Packa...

- alkanes
- C1 + UnknownC2
- C1_C2
- C1_C2 (EOS)
- HDA

Clearly COBIA-COBIA interop remains within the COBIA realm, and COM-COM interop does not change and remains COM. The cross interaction is provided by the glue, which was dubbed COMBIA, and has been tested with a variety of COM PMEs and PMEs.

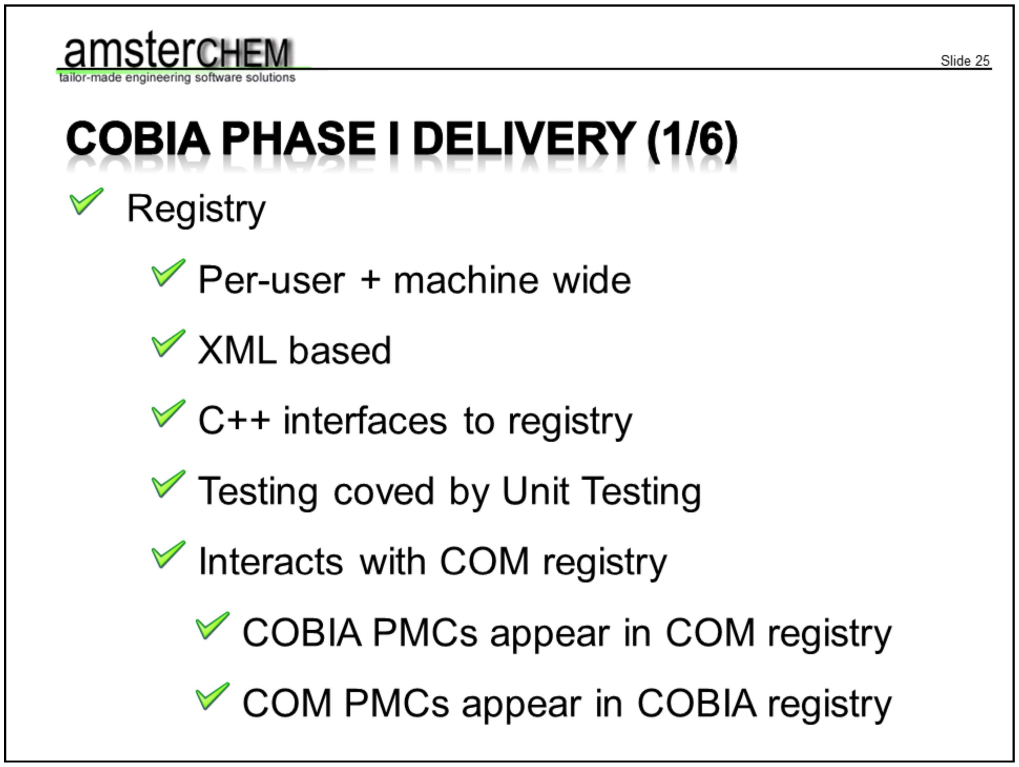

amsterCHEM
tailor-made engineering software solutions

Slide 24

TARGETS FOR A NEW MIDDLE WARE

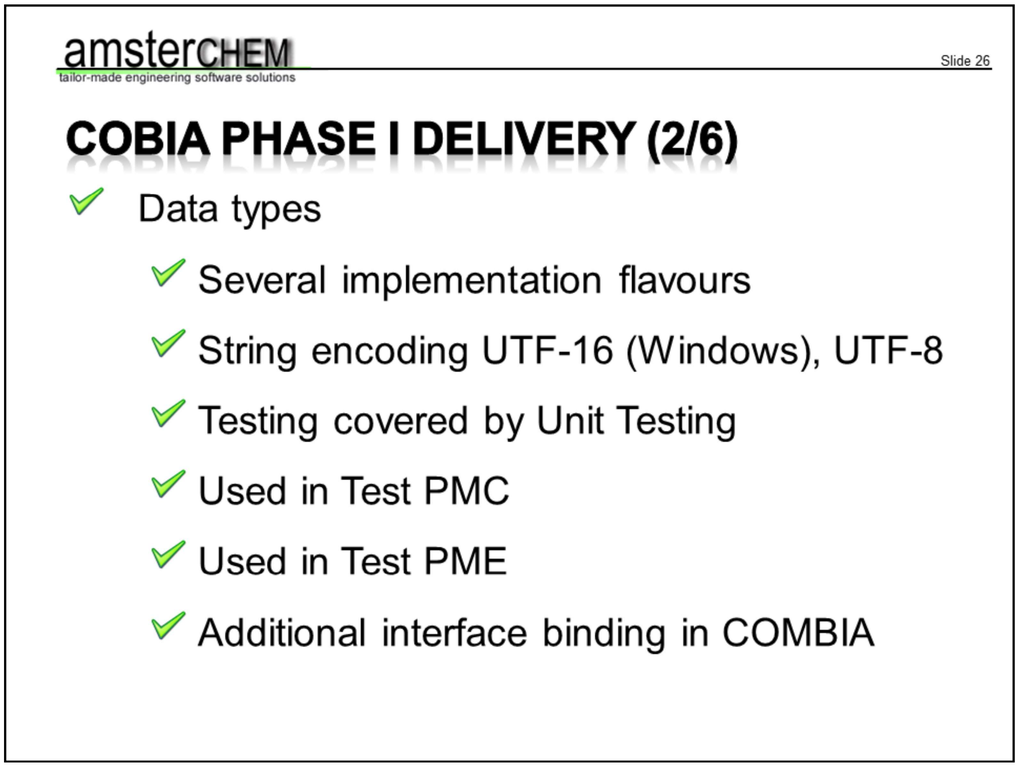
- ✓ Platform independent, independent of specific vendor (Microsoft)
- ✓ Easier on programmers
- ✓ Strong data typing
- ✓ More efficient
- ✓ Less error prone code
- ✓ Open source
- ✓ Fully COM compatible

Going back to our check list of targets, I hope it was clear from this presentation that all targets have already been demonstrated in the Phase I part of the implementation.


amsterCHEM
tailor-made engineering software solutions

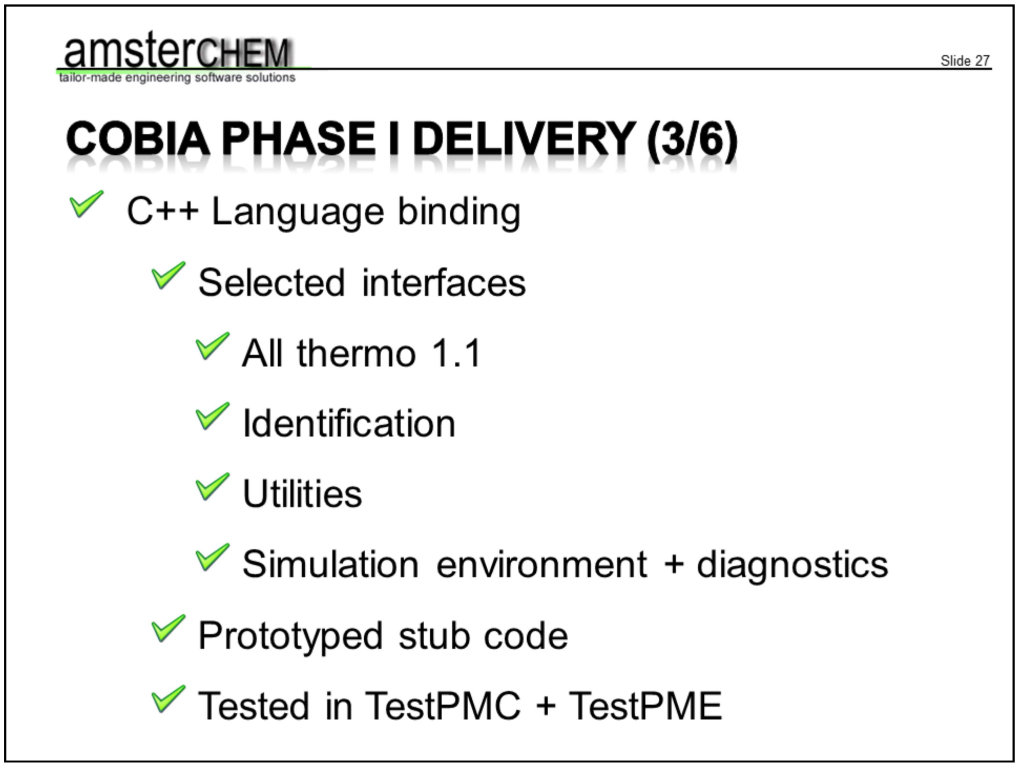
COBIA PHASE I DELIVERY (1/6)

- ✓ Registry
  - ✓ Per-user + machine wide
  - ✓ XML based
  - ✓ C++ interfaces to registry
  - ✓ Testing coved by Unit Testing
  - ✓ Interacts with COM registry
    - ✓ COBIA PMCs appear in COM registry
    - ✓ COM PMCs appear in COBIA registry


amsterCHEM
tailor-made engineering software solutions

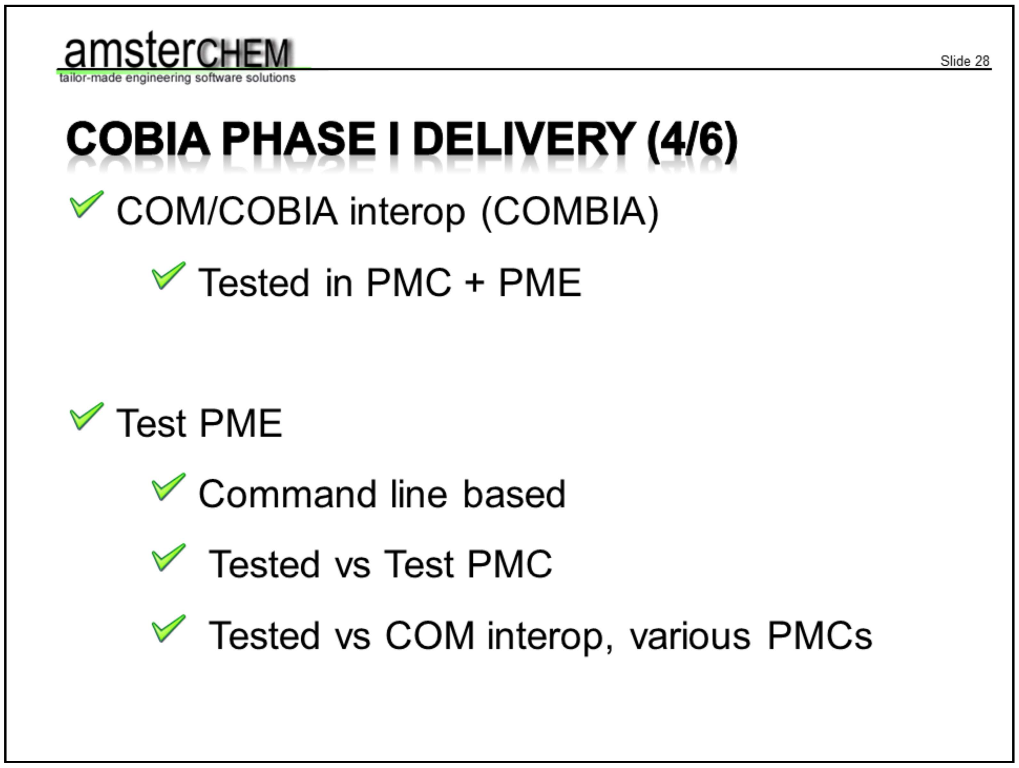
COBIA PHASE I DELIVERY (2/6)

- ✓ Data types
  - ✓ Several implementation flavours
  - ✓ String encoding UTF-16 (Windows), UTF-8
  - ✓ Testing covered by Unit Testing
  - ✓ Used in Test PMC
  - ✓ Used in Test PME
  - ✓ Additional interface binding in COMBIA


amsterCHEM
tailor-made engineering software solutions

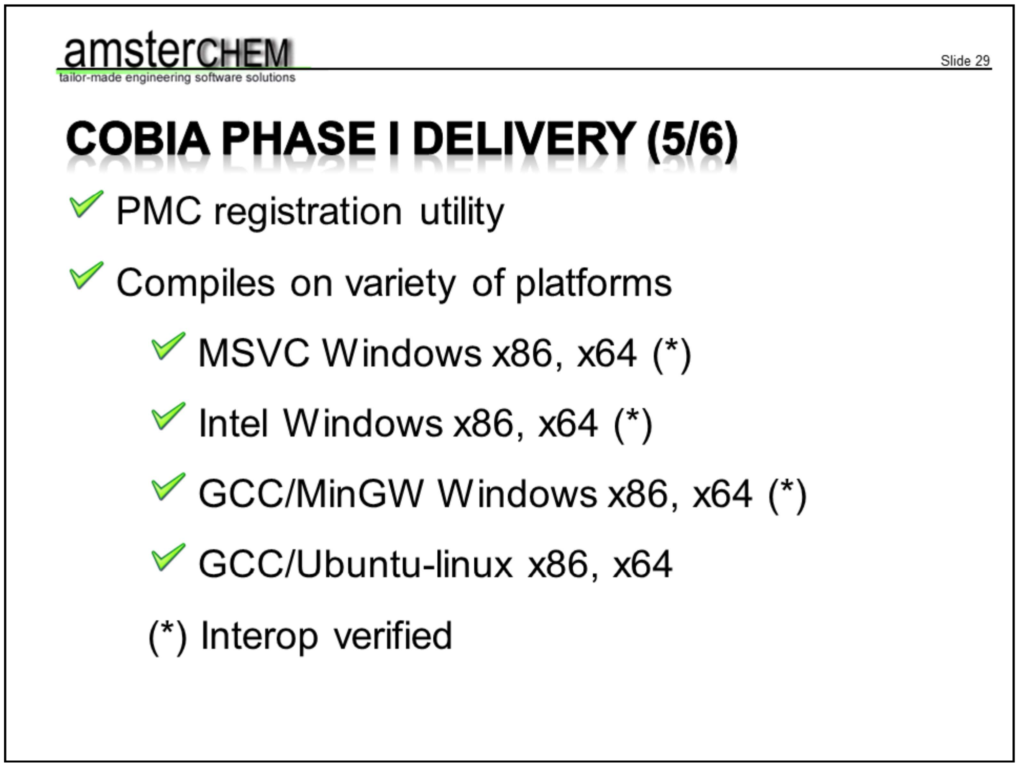
COBIA PHASE I DELIVERY (3/6)

- ✓ C++ Language binding
  - ✓ Selected interfaces
    - ✓ All thermo 1.1
    - ✓ Identification
    - ✓ Utilities
    - ✓ Simulation environment + diagnostics
  - ✓ Prototyped stub code
  - ✓ Tested in TestPMC + TestPME


amsterCHEM
tailor-made engineering software solutions

COBIA PHASE I DELIVERY (4/6)

- ✓ COM/COBIA interop (COMBIA)
  - ✓ Tested in PMC + PME
- ✓ Test PME
  - ✓ Command line based
  - ✓ Tested vs Test PMC
  - ✓ Tested vs COM interop, various PMCs

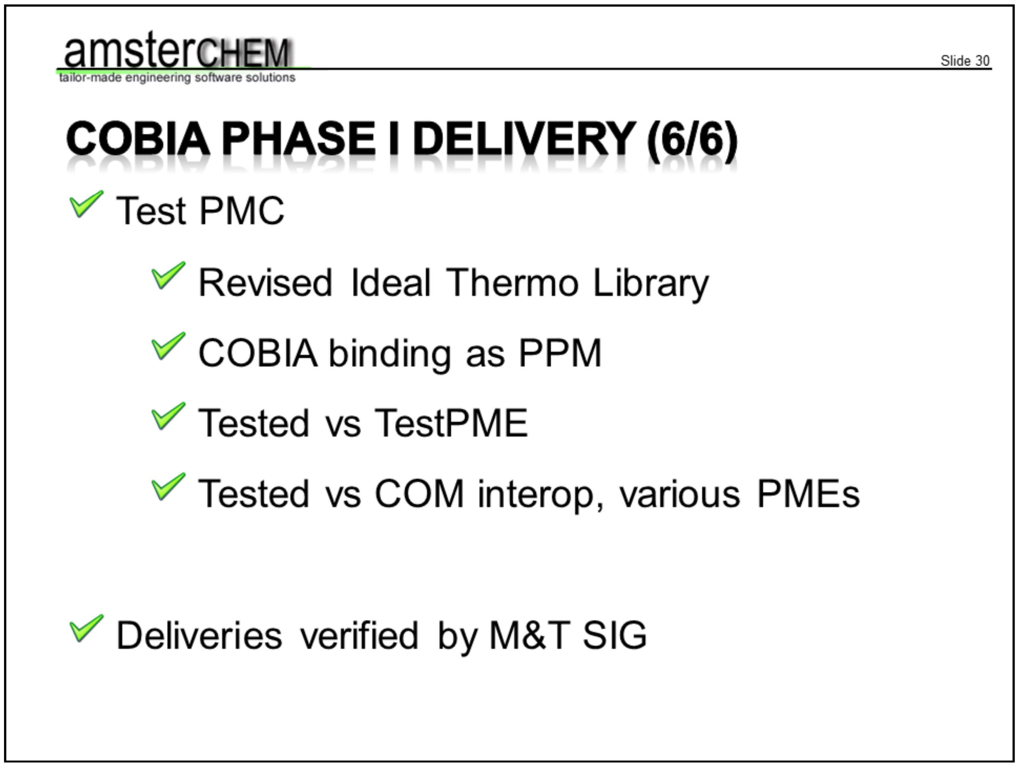


amsterCHEM
tailor-made engineering software solutions

COBIA PHASE I DELIVERY (5/6)

- ✓ PMC registration utility
- ✓ Compiles on variety of platforms
  - ✓ MSVC Windows x86, x64 (*)
  - ✓ Intel Windows x86, x64 (*)
  - ✓ GCC/MinGW Windows x86, x64 (*)
  - ✓ GCC/Ubuntu-linux x86, x64

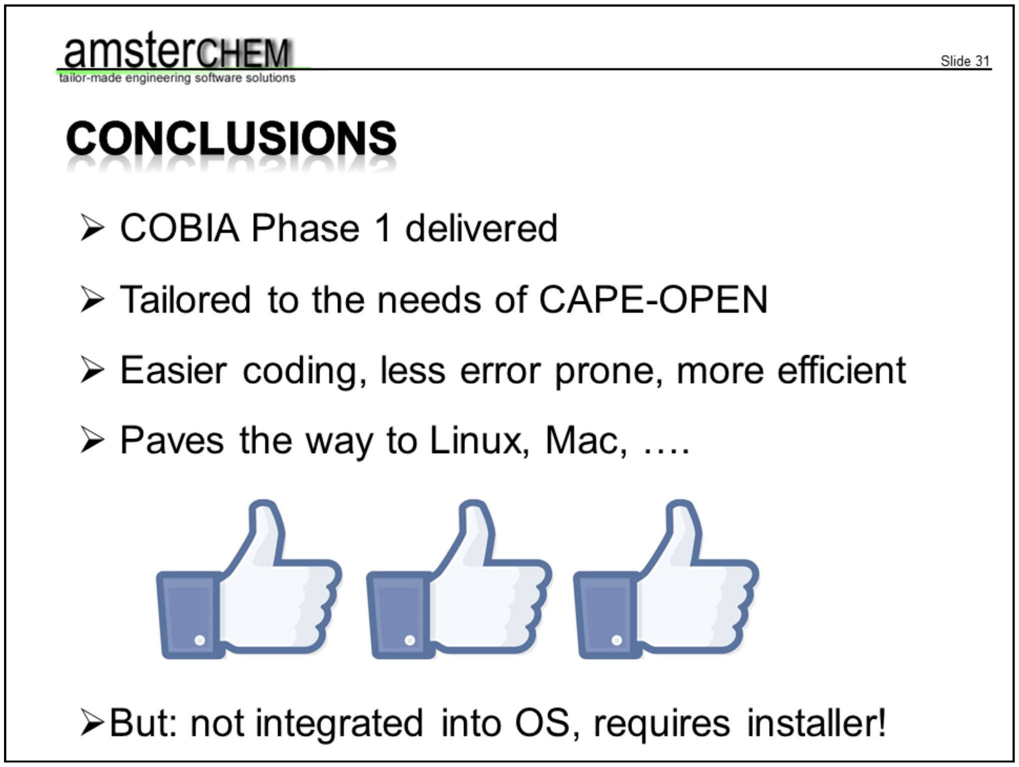
  (*) Interop verified



amsterCHEM
tailor-made engineering software solutions

COBIA PHASE I DELIVERY (6/6)

- ✓ Test PMC
  - ✓ Revised Ideal Thermo Library
  - ✓ COBIA binding as PPM
  - ✓ Tested vs TestPME
  - ✓ Tested vs COM interop, various PMEs
- ✓ Deliveries verified by M&T SIG



amsterCHEM
tailor-made engineering software solutions

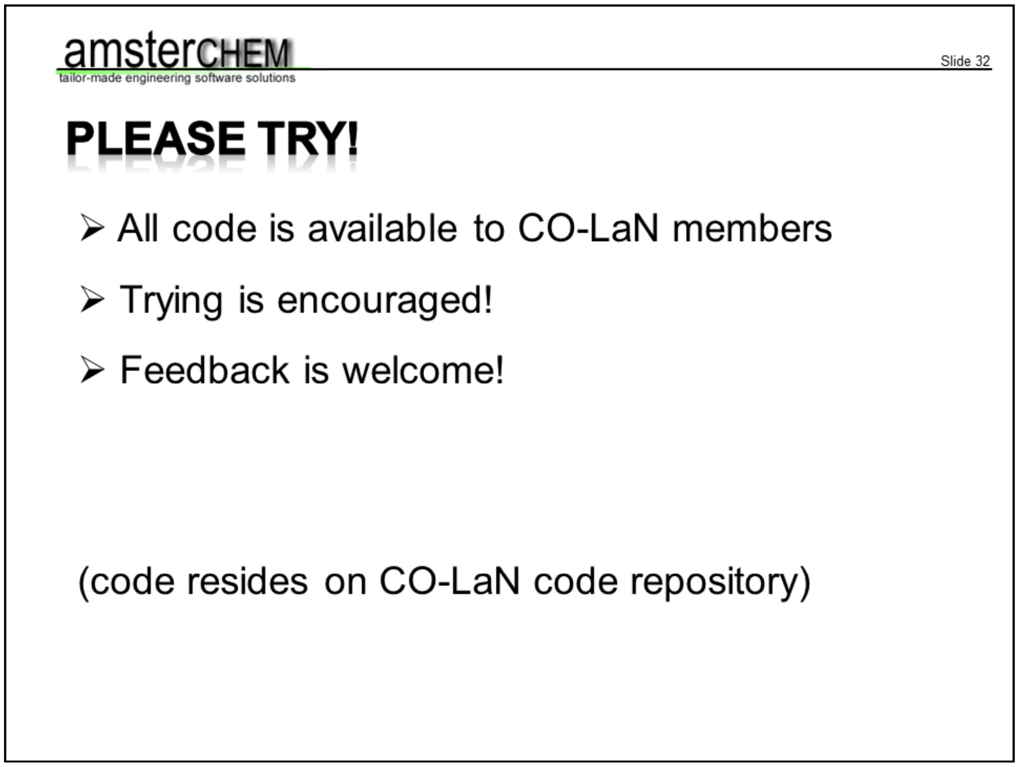
Slide 31

CONCLUSIONS

- COBIA Phase 1 delivered
- Tailored to the needs of CAPE-OPEN
- Easier coding, less error prone, more efficient
- Paves the way to Linux, Mac, ….
- But: not integrated into OS, requires installer!

I can talk about 3 more hours about COBIA, I id not go into the details of error handling, component registry and whatnot. But this presentation has a limited time slot. So to conclude: we started a new middleware, it is called COBIA. It does exactly what we need it to do, no more and no less. Therefore, programming becomes easier and less error prone, interop becomes more efficient. And we no longer are bound to only Windows. I myself am not disappointed, I think this will be a lot more effective than COM to drive CAPE-OPEN for the next decades. So from me it gets three thumbs up.

There is only one down side compared to COM that I can think of: COM is very integrated into Windows itself, while COBIA requires a separate CO-LaN installer. For COM we have a type lib and PIA installer, so I think this is not a big issue.


amsterCHEM
tailor-made engineering software solutions

Slide 32

PLEASE TRY!

- All code is available to CO-LaN members
- Trying is encouraged!
- Feedback is welcome!

(code resides on CO-LaN code repository)

But do please feel free to try it youself. All CO-LaN members have access to the COBIA code via the CO-LaN code repository; please ask Michel for further details if you do not have access yet. Now is the time to make corrections and improvements as the code is not yet frozen. Feedback is welcome from you all.