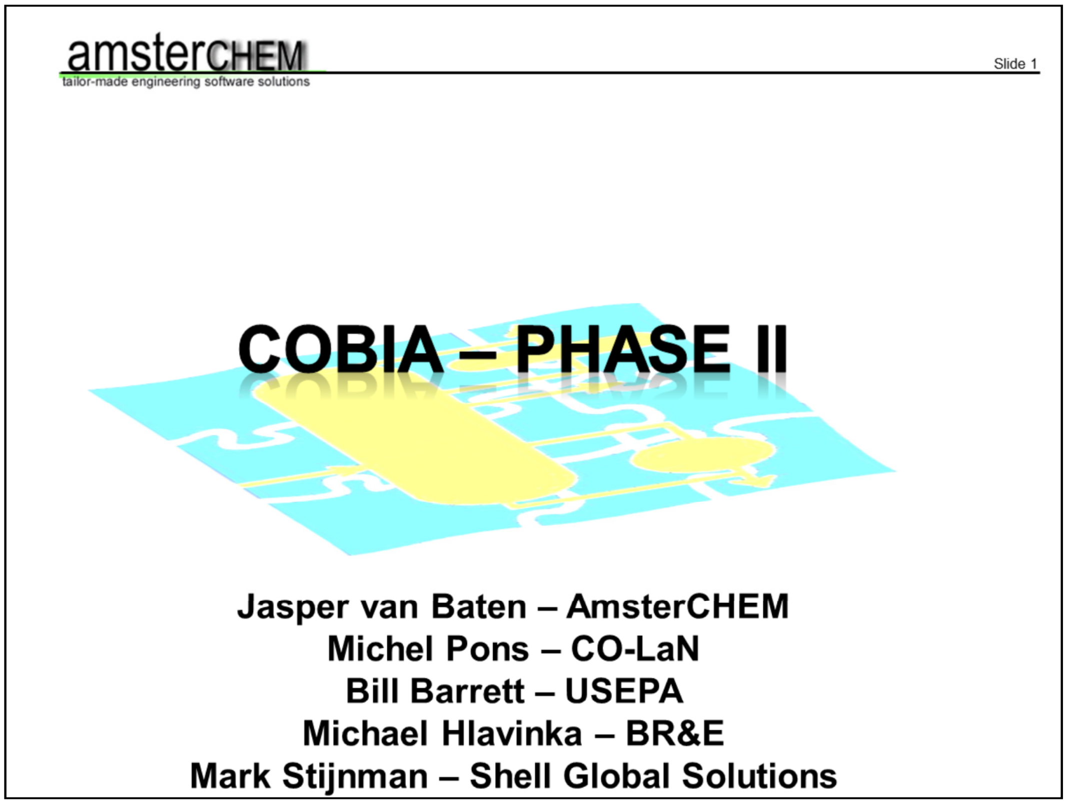
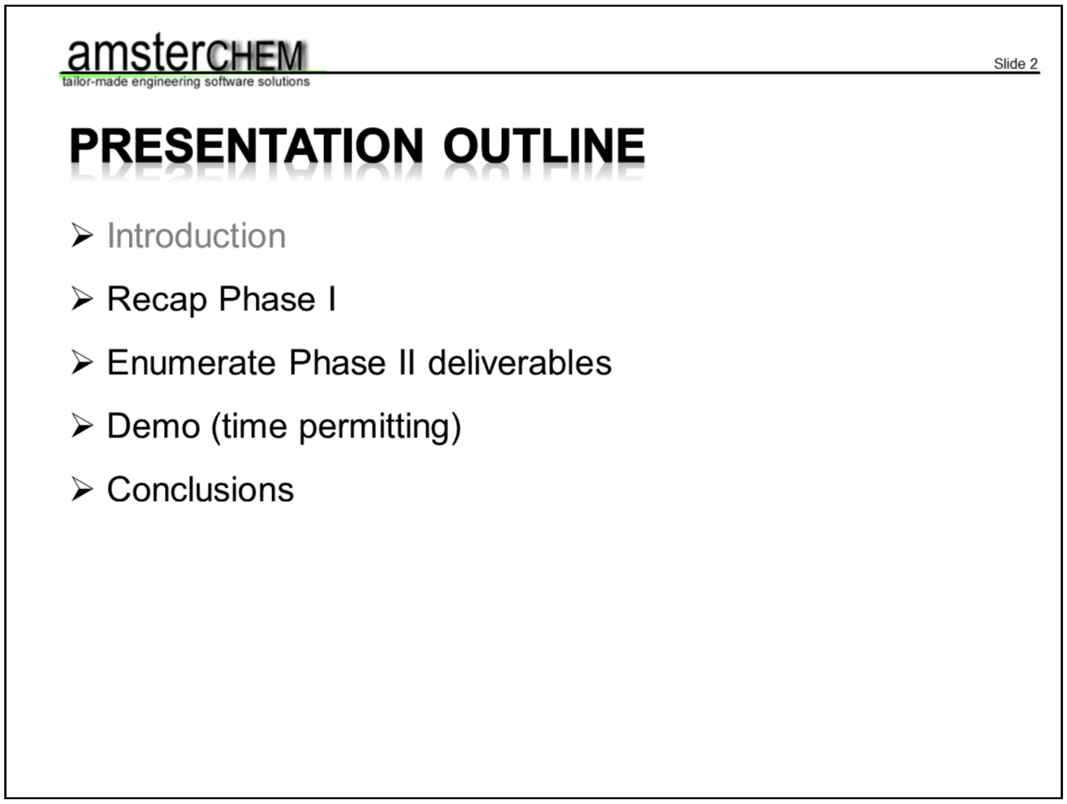


# COBIA – PHASE II

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

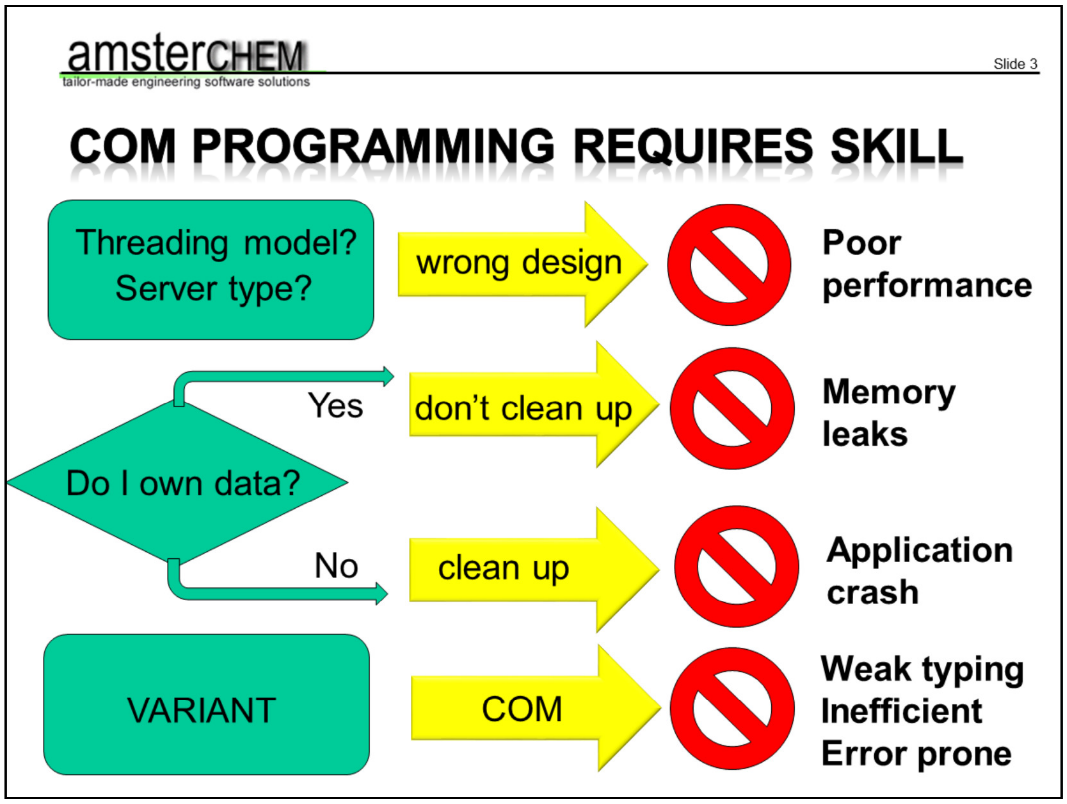

amsterCHEM
tailor-made engineering software solutions

Slide 2

## PRESENTATION OUTLINE

- Introduction
- Recap Phase I
- Enumerate Phase II deliverables
- Demo (time permitting)
- Conclusions

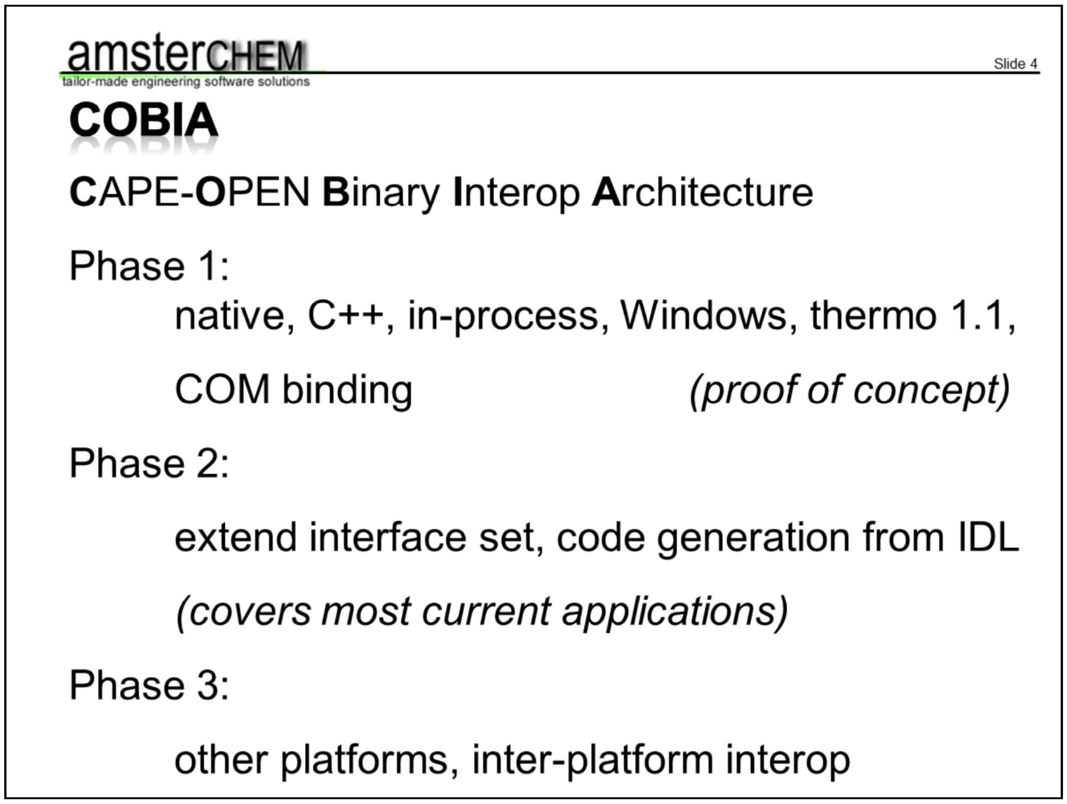
As presented in Pullach in October 2016, phase I of COBIA was completed in 2016. For those that are not aware, COBIA is to replace COM and CORBA as the middleware for CAPE-OPEN.  COBIA delivers a middleware specific to CAPE-OPEN while so far CAPE-OPEN implementations relied on COM or CORBA middleware, which are not CAPE-OPEN specific. We will briefly recapitulate what was done for phase I. Then we will see what was to be done for phase II and what has been done so far for phase II. I would be happy to show a demo if time allows. Finally, some conclusions....


amsterCHEM
tailor-made engineering software solutions

Slide 3

## COM PROGRAMMING REQUIRES SKILL

This is a slide from last year, stating why we are doing this. With the new middleware we want to tackle some difficulties in CAPE-OPEN programming and lower the threshold necessary to successfully develop a CAPE-OPEN Process Modelling Component or Environment. For example, using COM, one has to decide which threading model to use, and the wrong decision may lead to low performance. One has to know which data is owned and which are references to data owned by some external components. If you do this wrong, either you are leaking memory, or you will probably crash the application, neither of which is good. And then there are a number of issues resulting from weak typing, including the possibility of passing the wrong data type or getting the wrong data type passed to you, and poor performance because you must make repeatedly such checks. And of course, not shown here, is that COM is married to Windows, and CAPE-OPEN as an open standard should not rely on any commercial operating system. For COBIA we target Windows, linux, OSX, and perhaps others.


amsterCHEM

tailor-made engineering software solutions

Slide 4

## COBIA

**C**APE-**O**PEN **B**inary **I**nterop **A**rchitecture

Phase 1:

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

COM binding *(proof of concept)*

Phase 2:

extend interface set, code generation from IDL

*(covers most current applications)*

Phase 3:

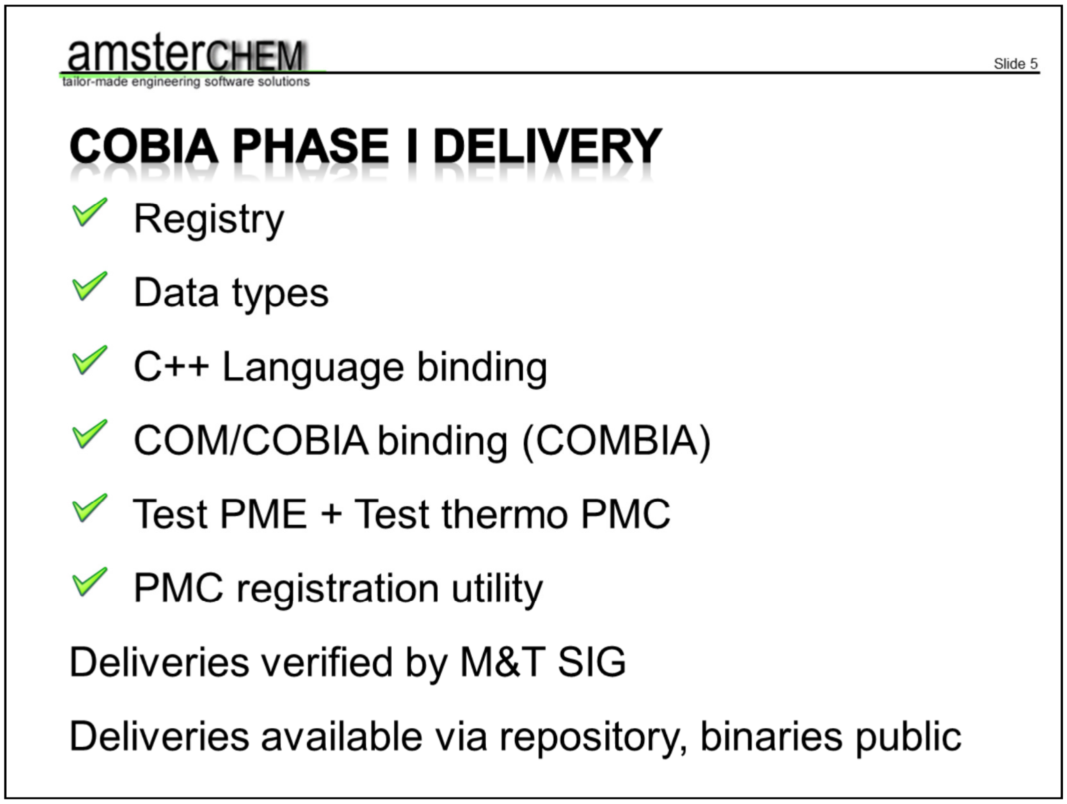
other platforms, inter-platform interop

COBIA stands for CAPE-OPEN Binary Interop Architecture. Its development is planned in three phases.

Phase 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 was completed last year.

Phase II extends the CAPE-OPEN interface set beyond just thermodynamics. 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 created and stub generation and any marshaling process can use information parsed directly from the IDL. Marshaling itself will not be implemented though, but as we operate directly from IDL, custom interfaces are implemented. Phase two is partly completed, and this is what we will be talking about today.

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


amsterCHEM
tailor-made engineering software solutions

Slide 5

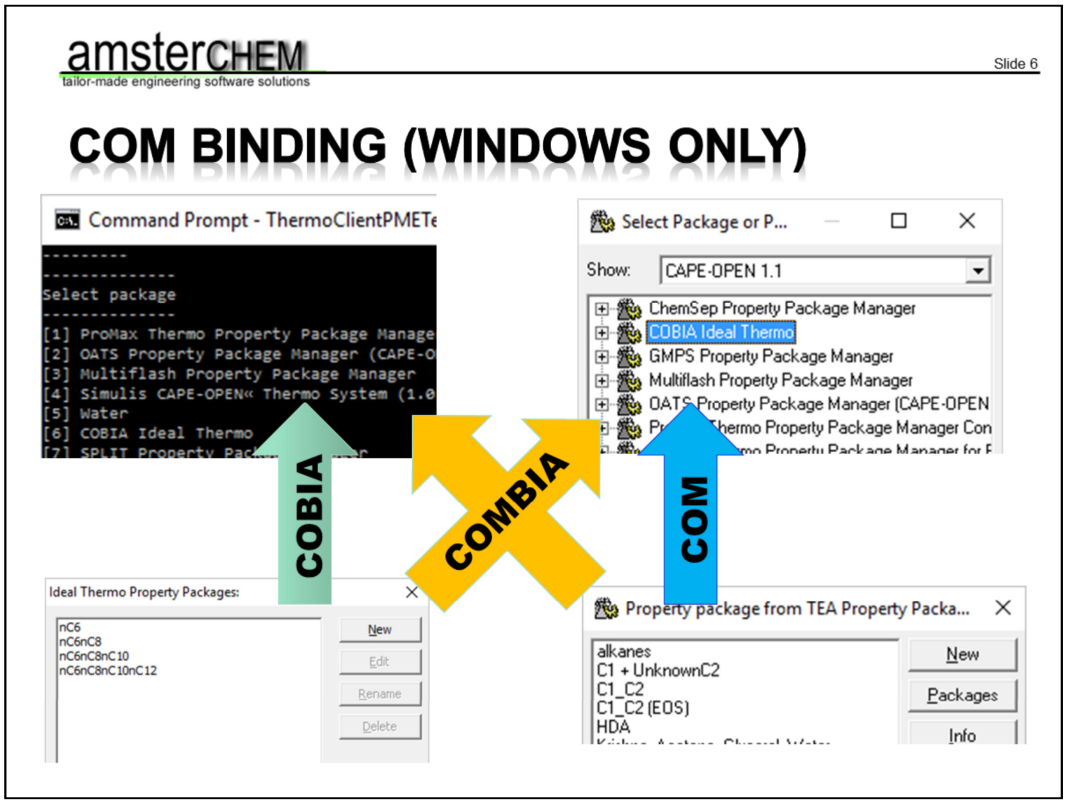
## COBIA PHASE I DELIVERY

- ✓ Registry
- ✓ Data types
- ✓ C++ Language binding
- ✓ COM/COBIA binding (COMBIA)
- ✓ Test PME + Test thermo PMC
- ✓ PMC registration utility

Deliveries verified by M&T SIG

Deliveries available via repository, binaries public

Here's a recap of phase 1. In phase 1 all the ground work was done: we have a CAPE-OPEN registry, we have defined how all CAPE-OPEN data types work and provided basic implementations for C++. We have formulated the C++ language binding. We have put the glue between COM and COBIA, COMBIA, in place. A test PME and test PMC were written, which are running on Windows and linux, and tested against a variety of compilers, thereby providing proof of concept of a binary interface. And a PMC registration utility was provided. All of this was verified by the M&T Special Interest Group.



amsterCHEM
tailor-made engineering software solutions

Slide 6

## COM BINDING (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

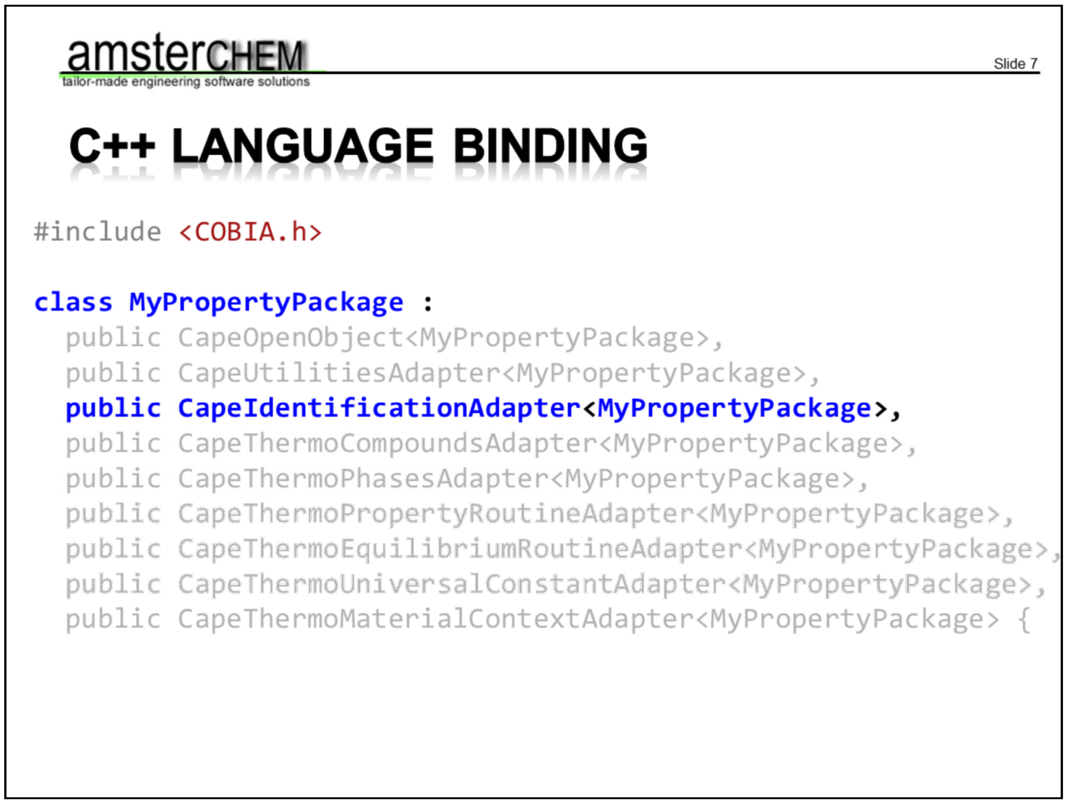
Ideal Thermo Property Packages:

- nC6
- nC6nC8
- nC6nC8nC10
- nC6nC8nC10nC12

Property package from TEA Property Packa...

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

So to refresh your memory on COMBIA: COM components will talk directly to COM components, as before. Similarly, COBIA components will talk directly to COBIA components. However, on Windows, we want to be able to use COBIA PMCs directly from COM PMEs and vice versa. This is where COMBIA comes in. From a COM PME, COMBIA is invoked as a wrapper around COBIA PMCs. This implies that registering a COBIA PMC also makes the equivalent COM registry entry. From a COBIA PME, additional registration requirements are not present, as COBIA on Windows will directly look into the COM registry for PMCs and will instantiate a COM PMC using COMBIA as a wrapper.


amsterCHEM
tailor-made engineering software solutions

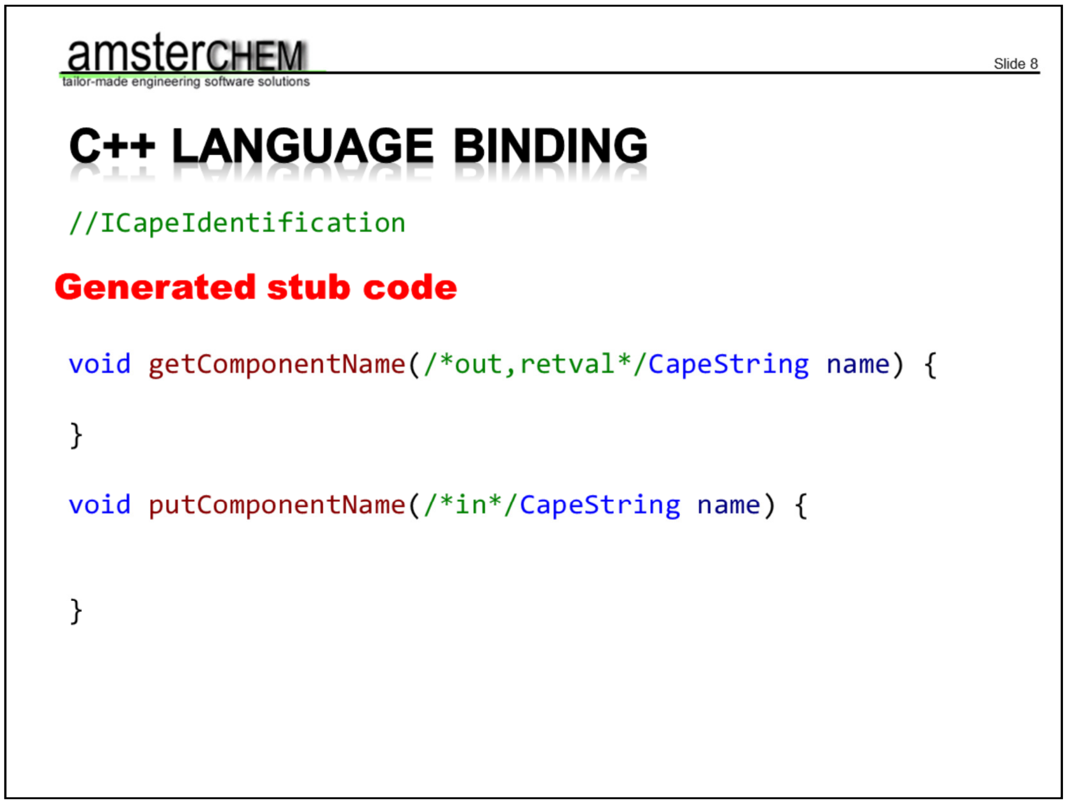
Slide 7

## C++ LANGUAGE BINDING

```
#include <COBIA.h>

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

The C++ language binding resulting from phase 1 makes that a COBIA based CAPE-OPEN object can look like this: a regular C++ class, derived from COBIA base object. Each CAPE-OPEN interface is implemented by deriving from a CAPE-OPEN adapter class. The adapter implements the raw CAPE-OPEN interface. The adapter will subsequently depend on functions being implemented in your class....


amsterCHEM
tailor-made engineering software solutions

Slide 8

## C++ LANGUAGE BINDING

```
//ICapeIdentification
```

**Generated stub code**

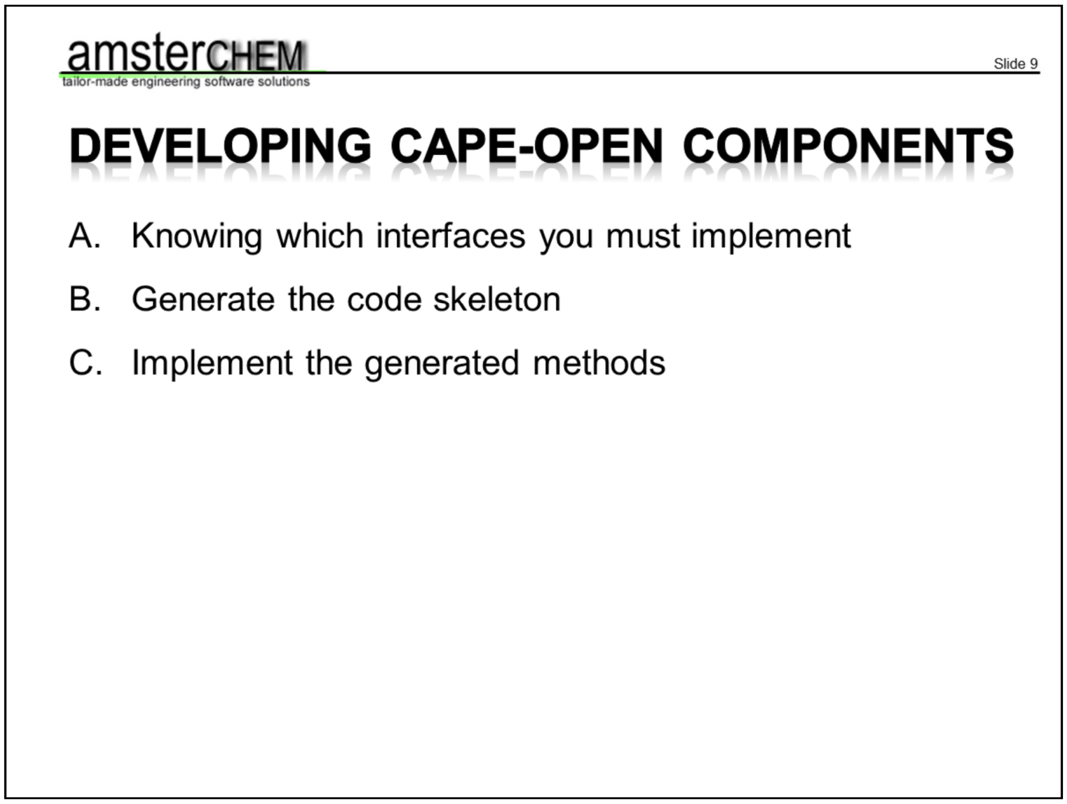
```
void getComponentName(/*out,retval*/CapeString name) {

}

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


}
```

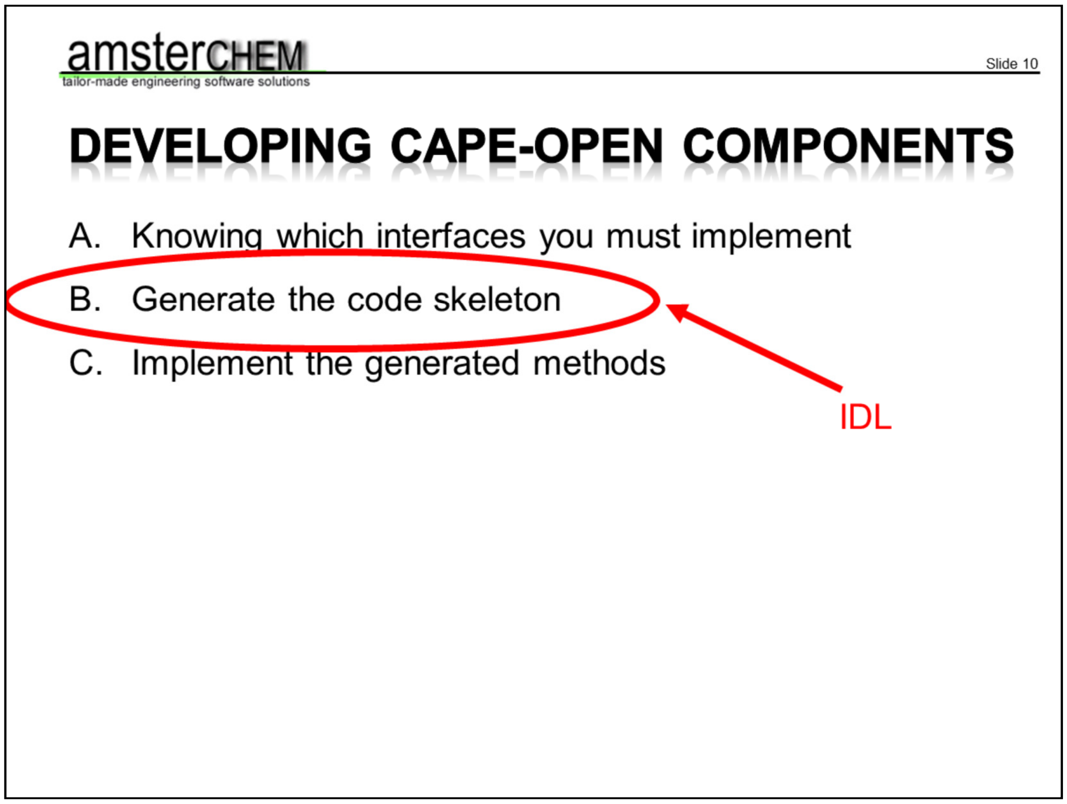
... as shown here for ICapeIdentification. These functions are called by the adapter base class. Phase II of COBIA revolved largely around generating all required code, the raw interfaces, the C++ wrapper classes, the adapter classes, manipulating the CAPE-OPEN classes to derive from the proper classes, and generating the stub code, directly from the IDL.


amsterCHEM
tailor-made engineering software solutions
Slide 9

## DEVELOPING CAPE-OPEN COMPONENTS

A. Knowing which interfaces you must implement
B. Generate the code skeleton
C. Implement the generated methods

With code generation in place, development of CAPE-OPEN components is then reduced to: knowing which interfaces to implement, generating the code skeleton, and implementing the methods that are generated for you.


amsterCHEM
tailor-made engineering software solutions

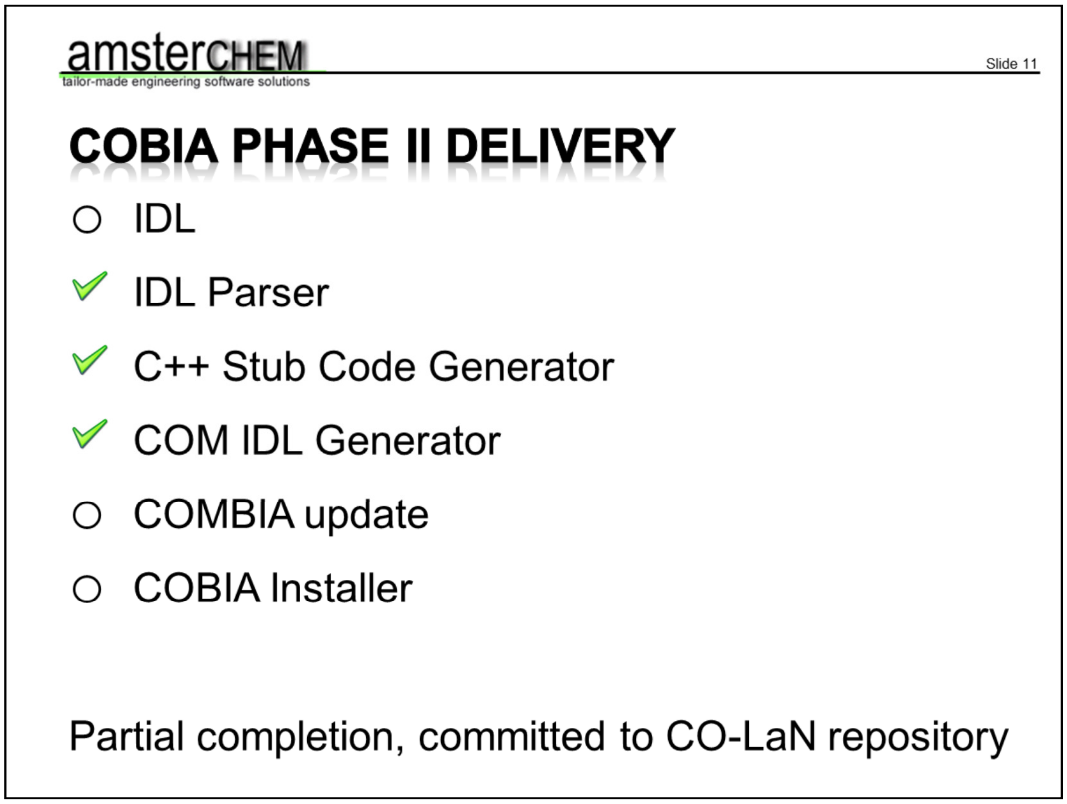
## DEVELOPING CAPE-OPEN COMPONENTS

A. Knowing which interfaces you must implement

B. Generate the code skeleton

C. Implement the generated methods

COBIA Phase II focuses on the code generation.



amsterCHEM
tailor-made engineering software solutions

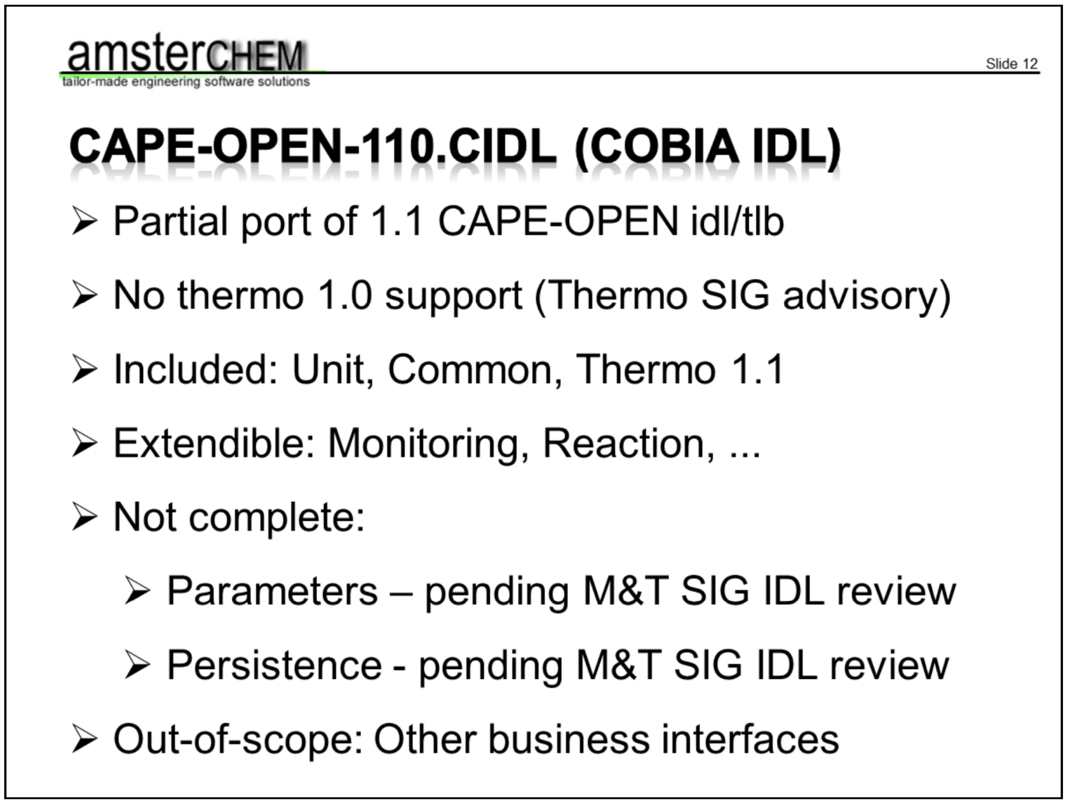
Slide 11

## COBIA PHASE II DELIVERY

- ○ IDL
- ✓ IDL Parser
- ✓ C++ Stub Code Generator
- ✓ COM IDL Generator
- ○ COMBIA update
- ○ COBIA Installer

Partial completion, committed to CO-LaN repository

So let’s look closer at the phase II deliveries. In the following slides I will go over all of these separately. In short, we need the complete IDL, the IDL parser, the C++ stub code generator, the COM IDL code generator, an update to COMBIA to reflect all the above changes, and an installer to put it all on the user’s system. For now. Let’s have a closer look.


amsterCHEM
tailor-made engineering software solutions
Slide 12

## CAPE-OPEN-110.CIDL (COBIA IDL)

- Partial port of 1.1 CAPE-OPEN idl/tlb
- No thermo 1.0 support (Thermo SIG advisory)
- Included: Unit, Common, Thermo 1.1
- Extendible: Monitoring, Reaction, ...
- Not complete:
  - Parameters – pending M&T SIG IDL review
  - Persistence - pending M&T SIG IDL review
- Out-of-scope: Other business interfaces

We decided to not name the IDL files IDL, as this is used for both CORBA and COM idl and will open with system defined tools, such as Microsoft's MIDL. The file extension for CAPE-OPEN IDL will be CIDL. The COBIA CAPE-OPEN 1.1 type library is not the same as the COM one, as several differences between COM and COBIA are addressed in the type library, including strong typing. But in the future we hope to have one CAPE-OPEN IDL, and generate the COM idl from there.

The COBIA CAPE-OPEN 1.1 IDL is therefore a partial port of the COM IDL. Thermo 1.0 is not included. The Thermo SIG has advised that all new developments should use CAPE-OPEN 1.1. This includes COBIA. The Thermo group has expressed that implementing direct support for 1.0 is undesired, as all new implementations will then still remain with the situation of having to implement dual thermo support. The Thermo SIG also advises against an automatic conversion between CAPE-OPEN 1.0 and 1.1 thermo in the translation between COM and COBIA, COMBIA, as this translation is not 1:1 and open to interpretation, and there is not much business drive anymore for such an implementation, and it will incur a substantial amount of undesired future maintenance overhead because if we provide this layer, people will actually get to depend on it. So no Thermo 1.0.

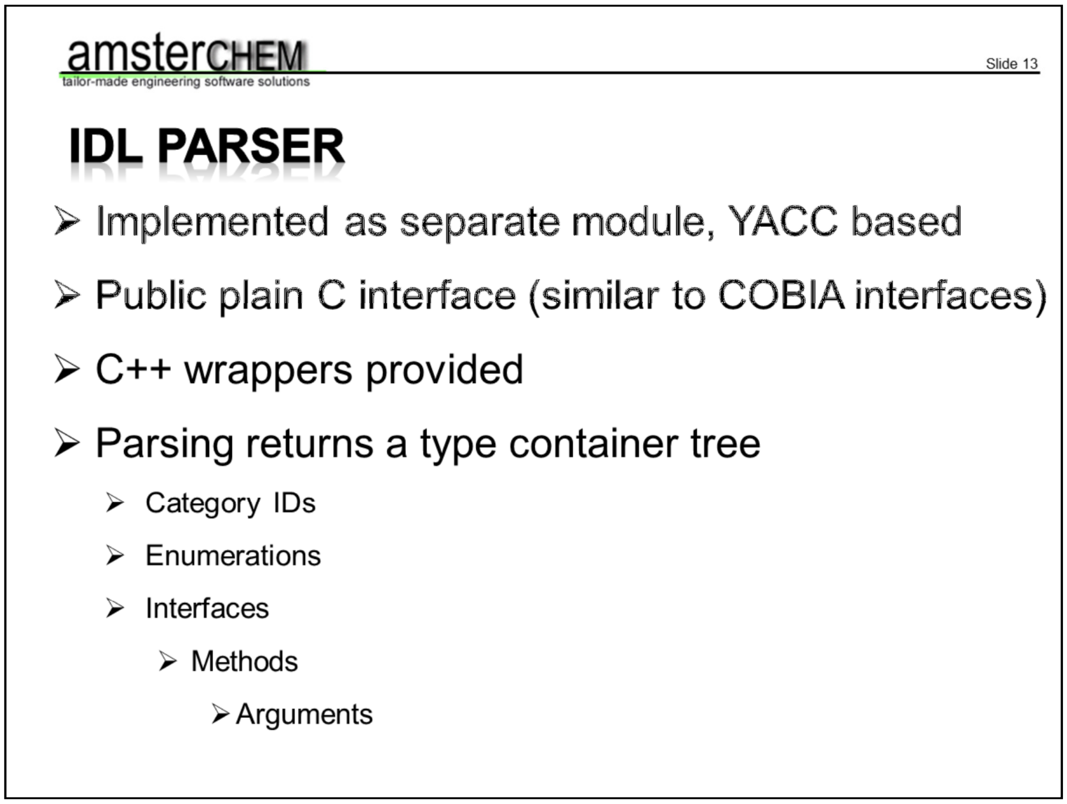
Included is Unit, Thermo 1.1 and all common interfaces. Interfaces that are currently under development, such as reaction and monitoring, can be added once published. Pending items are:

Parameters – which were deemed for a COBIA specific redesign as part of the

COBIA target is strong typing, and the current definition of parameters is not. An IDL has been proposed and is pending M&T SIG review.

Persistence – current persistence interfaces are defined by COM, so COBIA persistence interfaces must be defined. We can make a lot of improvements here. Again, an IDL has been proposed and is pending M&T SIG review.

And then there are interfaces that are not widely used, for which we have no business cases of porting them, until somebody points out that there are business cases. These include all numeric interfaces.


amsterCHEM

tailor-made engineering software solutions

Slide 13

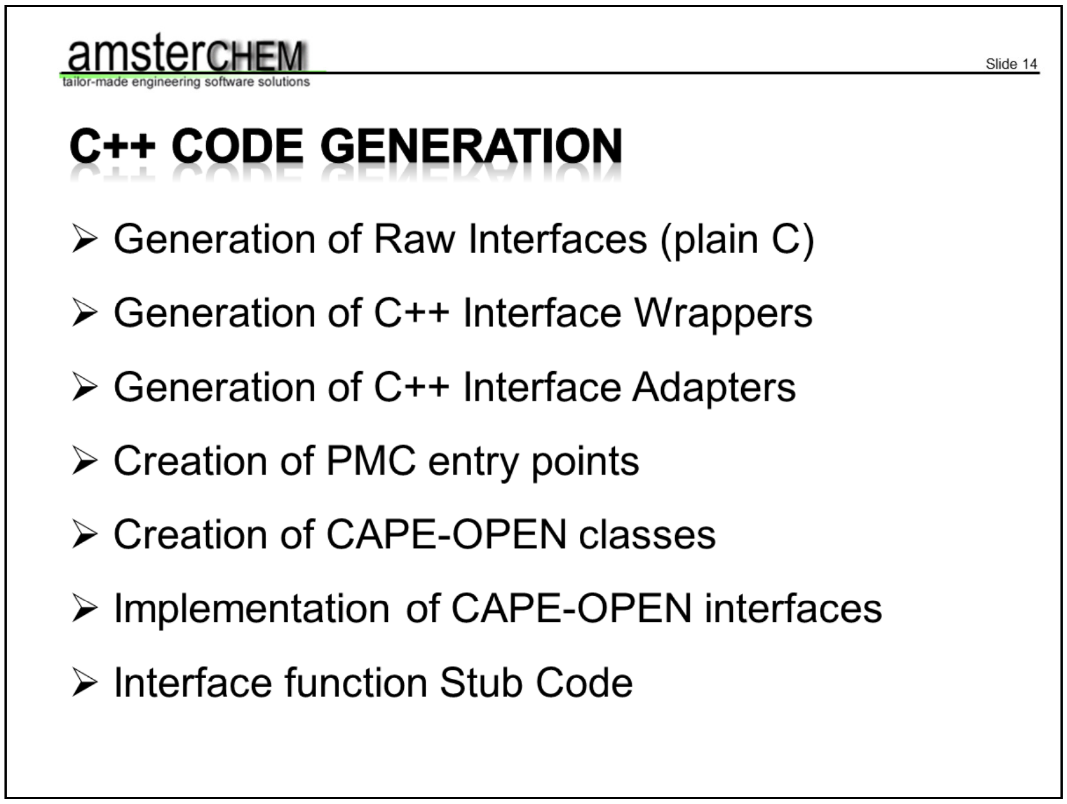
## IDL PARSER

- Implemented as separate module, YACC based
- Public plain C interface (similar to COBIA interfaces)
- C++ wrappers provided
- Parsing returns a type container tree
  - Category IDs
  - Enumerations
  - Interfaces
    - Methods
      - Arguments

The parser is built on Berkeley YACC, which is freeware. Alternatively one could use GNU's Bison to compile the grammar file, as Bison and YACC are compatible. Both the YACC input and output will be part of the COBIA source code, so one does not actually need to install YACC to compile COBIA. The parser is implemented as a separate shared library, with a public plain C interface for binary compatibility between compilers, much like the COBIA interfaces itself. C++ wrappers for these interfaces are also provided, so that you can immediately write a C++ application yourself that takes COBIA IDL files and uses the data in it.

The data is returned as interfaces to a parser tree, containing Category IDs (which are in COBIA IDL but not in COM IDL), Enumerations and Interfaces. Interfaces contain a collection of Methods, Methods contain a collection of Arguments. All of these contain a collection of IDL attributes.

Note that the data objects out of the parser will also be available from the COBIA registry – but that will be part of COBIA phase III as this is required not for code generation, but for marshaling.



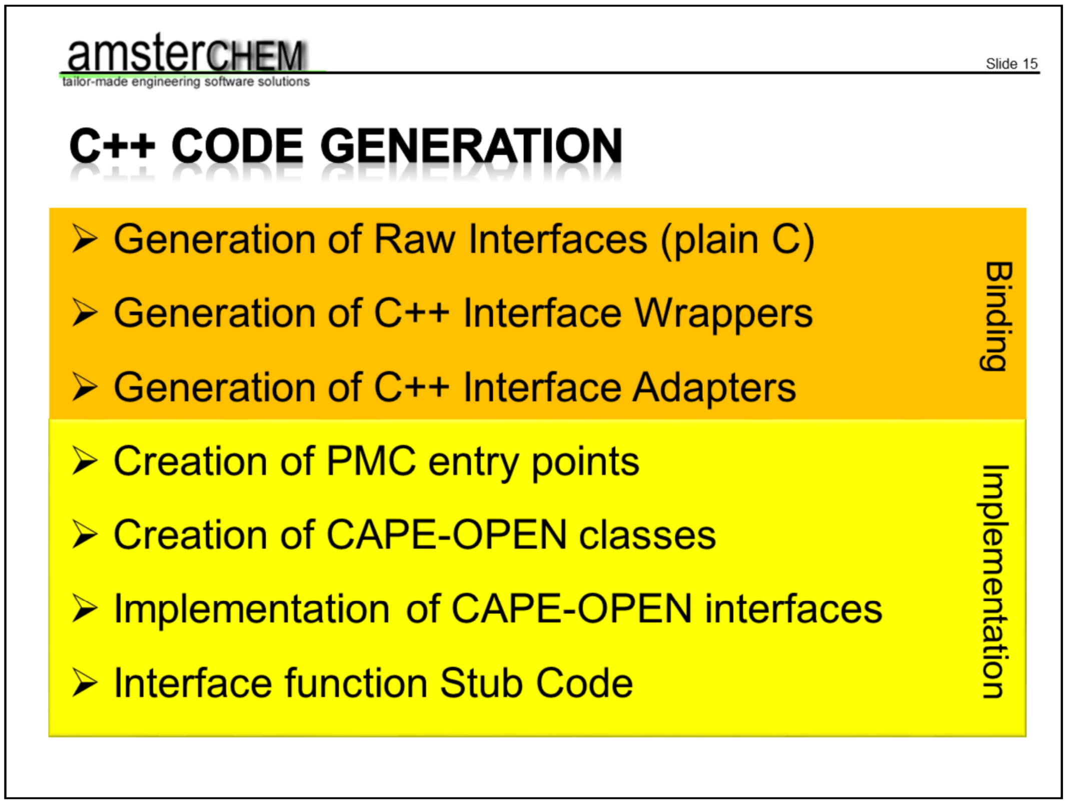
amsterCHEM
tailor-made engineering software solutions

Slide 14

## C++ CODE GENERATION

- Generation of Raw Interfaces (plain C)
- Generation of C++ Interface Wrappers
- Generation of C++ Interface Adapters
- Creation of PMC entry points
- Creation of CAPE-OPEN classes
- Implementation of CAPE-OPEN interfaces
- Interface function Stub Code

The IDL parse tree is subsequently used for code generation, and in a later stage marshaling. The C++ code generator is part of phase II. Seven distinct steps can be identified in the code generation process. The IDL interfaces must be compiled to the raw, plain C, interfaces. We need C++ wrapper classes around these interfaces that do things like reference counting. Think of these as smart pointers, but they do more, they also use C++ rather than plain C access to all CAPE-OPEN data types and deal with exception handling. Then we need to generate interface adapters, which are base classes of CAPE-OPEN objects. These will take care of implementing the plain C interface for you and adapting all types to something more convenient for C++. This includes the data types and exception handling as well. Just like COM servers, COBIA PMC modules must have entry points for the class factory and object registration. These entry points are generated by the code generator upon request. Finally the code generator can create CAPE-OPEN classes for you, and you can tell it to implement interfaces on these classes. The latter means that the class must derived from the appropriate adapters and must implement certain functions, the stub code for which is generated for you.


amsterCHEM
tailor-made engineering software solutions

Slide 15

## C++ CODE GENERATION

- Generation of Raw Interfaces (plain C)
- Generation of C++ Interface Wrappers
- Generation of C++ Interface Adapters

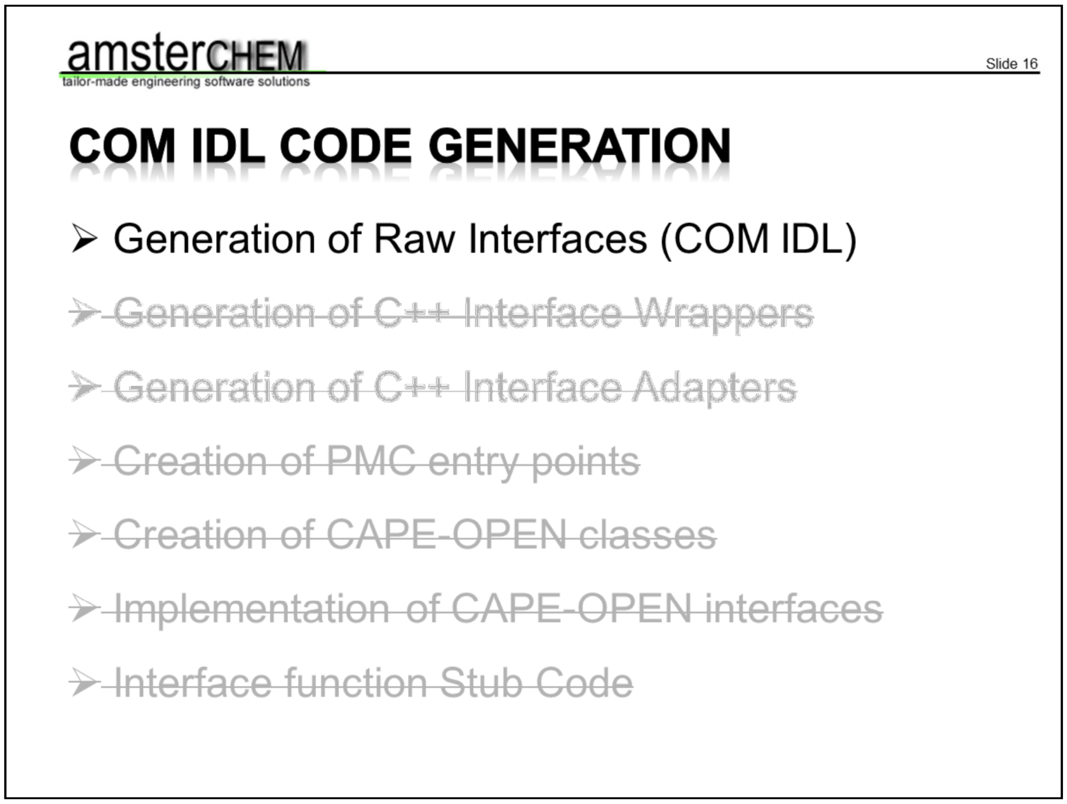
Binding

- Creation of PMC entry points
- Creation of CAPE-OPEN classes
- Implementation of CAPE-OPEN interfaces
- Interface function Stub Code

Implementation

Roughly, the code generation functions divide into two categories: creating C++ language binding from the IDL, in orange, and implementing CAPE-OPEN objects, in yellow. As such, the part in orange is used to create the COBIA client header files from the COBIA IDL. So COBIA code is now used to generate part of the COBIA code. But developers can also use this part to create a language binding for their custom interfaces in their own IDL.

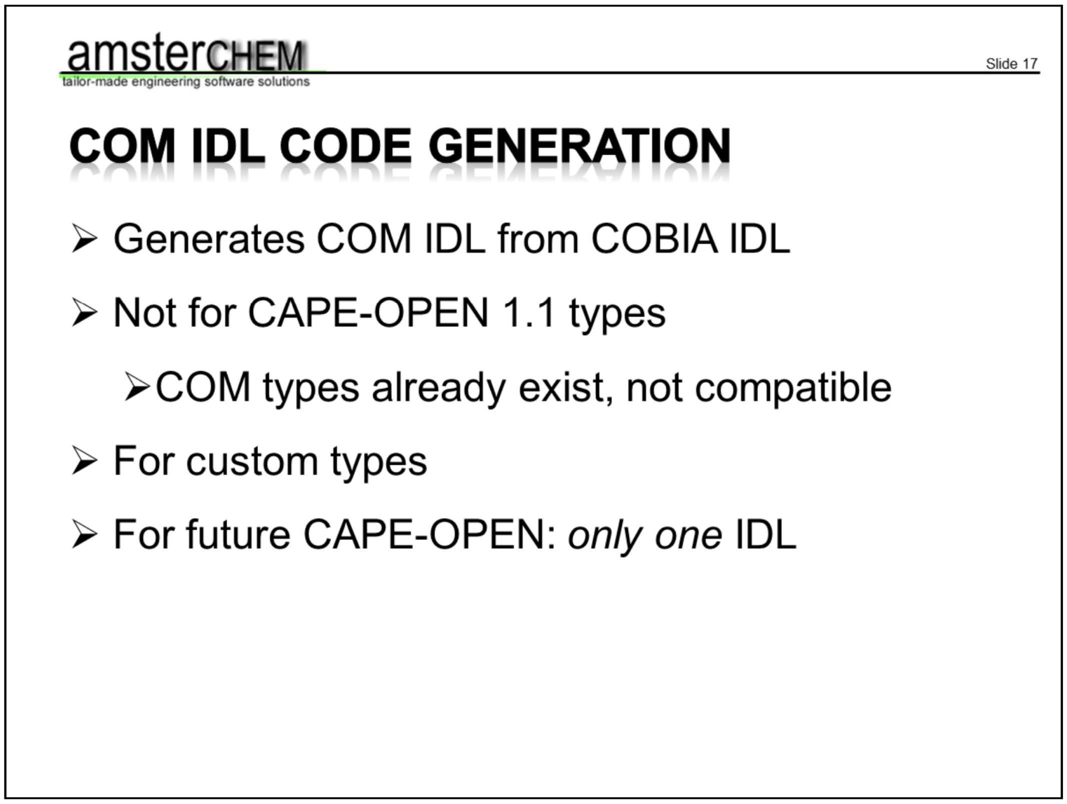
The part in yellow is really meant to construct code specific to a particular implementation.


amsterCHEM
tailor-made engineering software solutions

## COM IDL CODE GENERATION

- Generation of Raw Interfaces (COM IDL)
- ~~Generation of C++ Interface Wrappers~~
- ~~Generation of C++ Interface Adapters~~
- ~~Creation of PMC entry points~~
- ~~Creation of CAPE-OPEN classes~~
- ~~Implementation of CAPE-OPEN interfaces~~
- ~~Interface function Stub Code~~

The code generators themselves are implemented along interfaces that are plain C and look much like the COBIA version of CAPE-OPEN interfaces. Therefore, code generators for other language bindings will have the same 7 tasks. One of those is the code generator for COM IDL. It has only one of these 7 tasks implemented, which is the Raw Interfaces. If you ask the COM code generator to do anything else, it will tell you that it is not supported for the COM language binding.

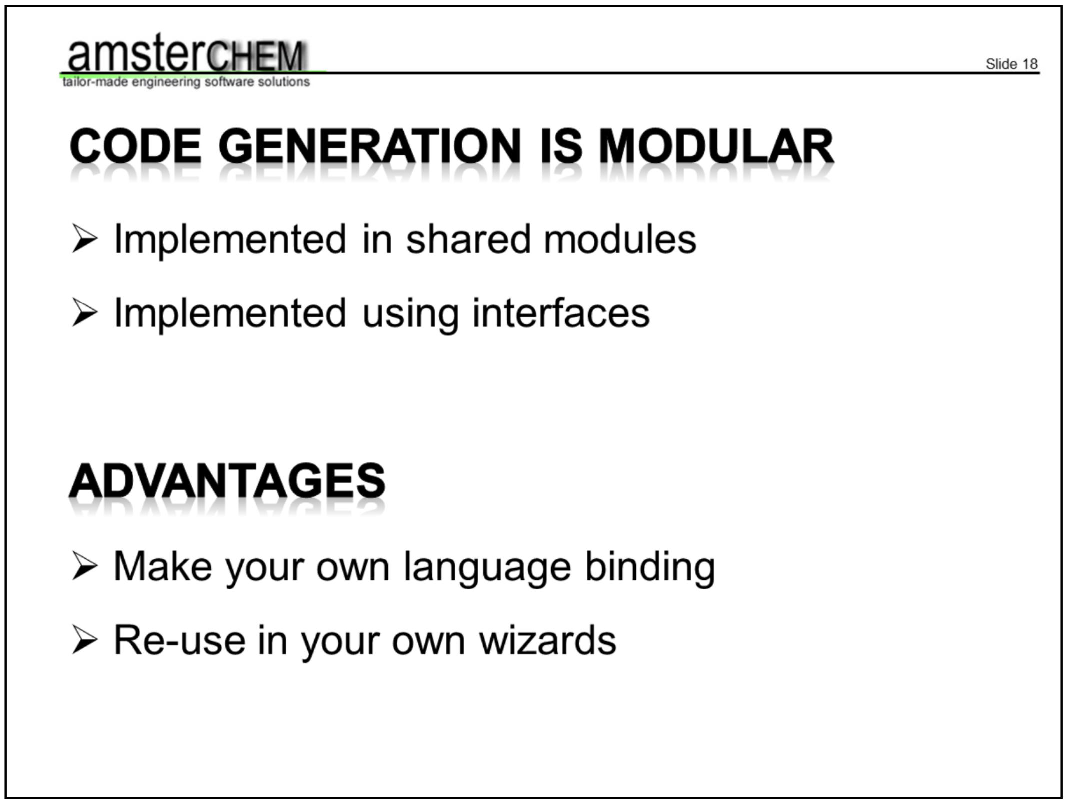

amsterCHEM
tailor-made engineering software solutions

Slide 17

## COM IDL CODE GENERATION

- Generates COM IDL from COBIA IDL
- Not for CAPE-OPEN 1.1 types
  - COM types already exist, not compatible
- For custom types
- For future CAPE-OPEN: *only one* IDL

The COM code generator generates COM IDL from COBIA IDL. Which is not useful for the 1.1 CAPE-OPEN type library itself, as these have a COM definition that differs from the COBIA definition. But in the future hopefully we'll define the interface set in COBIA IDL and generate compatible COM IDL from there. Also this can be done for custom types in IDL written by developers for interop between COM and COBIA.


amsterCHEM
tailor-made engineering software solutions

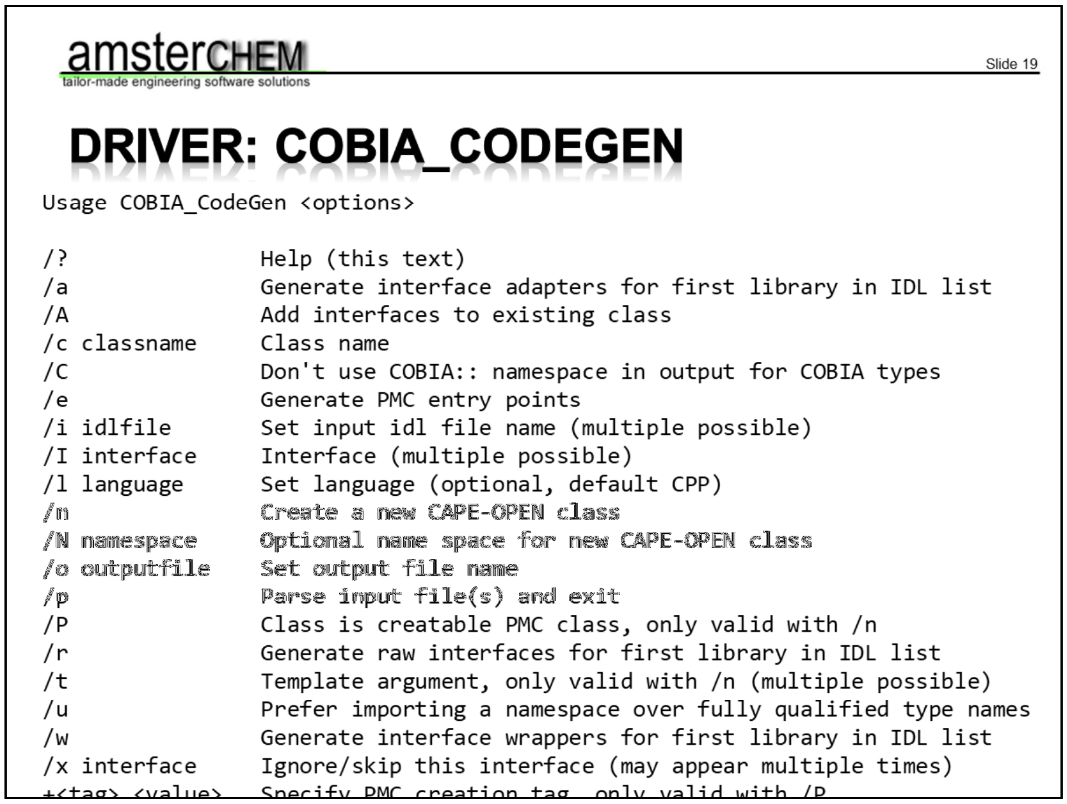
## CODE GENERATION IS MODULAR

- Implemented in shared modules
- Implemented using interfaces

## ADVANTAGES

- Make your own language binding
- Re-use in your own wizards

The code generators are provided in separate modules, and can be used from a generic interface, for which C++ wrappers and adapters are provided. So you can write your own language binding just by implementing a code generator, but you can also use the code generators to set up your own wizards, such as a CAPE-OPEN object wizards in Visual Studio.

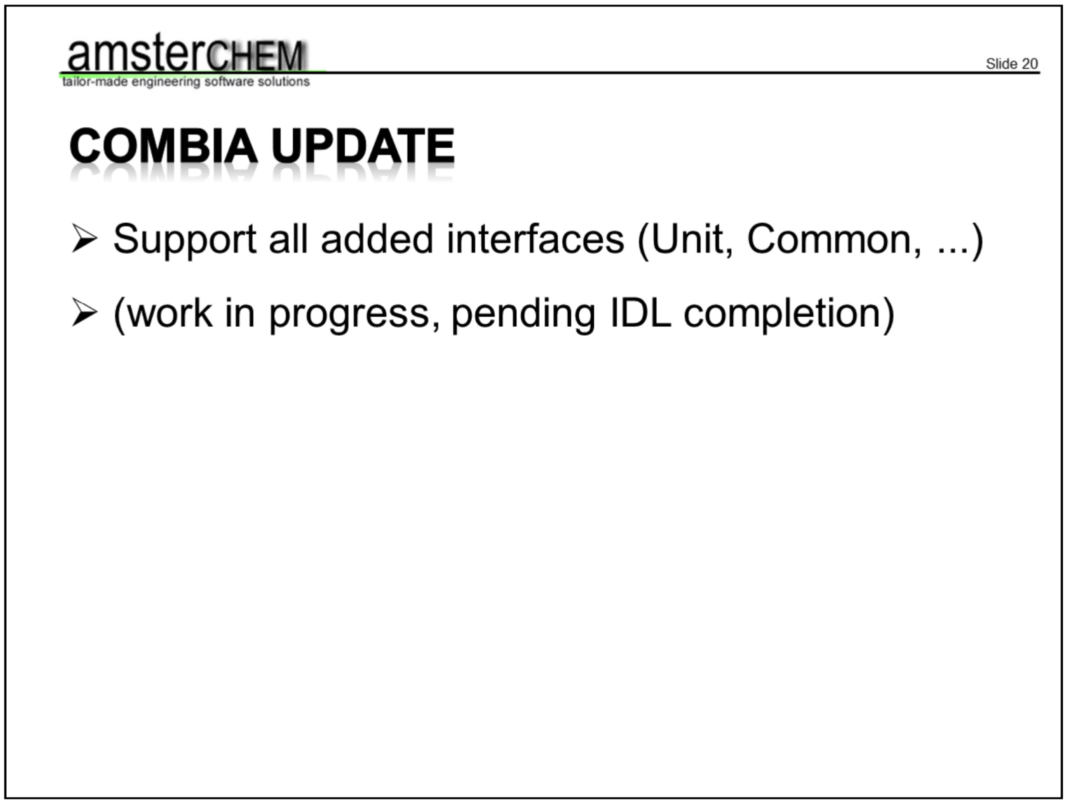

amsterCHEM
tailor-made engineering software solutions
Slide 19

## DRIVER: COBIA_CODEGEN

```
Usage COBIA_CodeGen <options>

/?                Help (this text)
/a                Generate interface adapters for first library in IDL list
/A                Add interfaces to existing class
/c classname      Class name
/C                Don't use COBIA:: namespace in output for COBIA types
/e                Generate PMC entry points
/i idlfile        Set input idl file name (multiple possible)
/I interface      Interface (multiple possible)
/l language       Set language (optional, default CPP)
/n                Create a new CAPE-OPEN class
/N namespace      Optional name space for new CAPE-OPEN class
/o outputfile     Set output file name
/p                Parse input file(s) and exit
/P                Class is creatable PMC class, only valid with /n
/r                Generate raw interfaces for first library in IDL list
/t                Template argument, only valid with /n (multiple possible)
/u                Prefer importing a namespace over fully qualified type names
/w                Generate interface wrappers for first library in IDL list
/x interface      Ignore/skip this interface (may appear multiple times)
+<tag> <value>    Specify PMC creation tag, only valid with /P
```

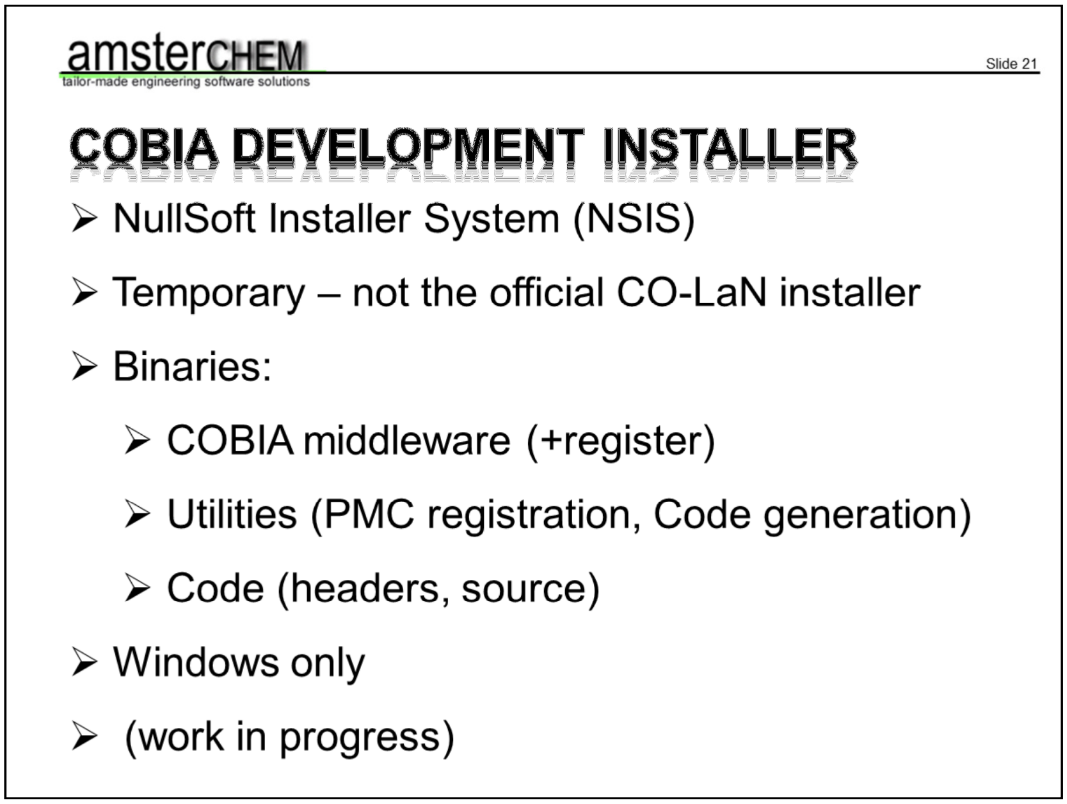
Such a wizard is at this point not provided, but there is a command line application that you can use, that has a whole bunch of command line options to tell it what to do. I am not going through all of these options now, I cannot even get them to all fit on this slide. Just type COBIA_CODEGEN and the application itself will tell you.


amsterCHEM
tailor-made engineering software solutions

## COMBIA UPDATE

- Support all added interfaces (Unit, Common, ...)
- (work in progress, pending IDL completion)

One delivery is to make sure that COM interop is in place for all the new functionality. This requires a COMBIA update, which is not yet done, mostly because the IDL is not complete yet. COMBIA will, for phase II, not support custom interfaces, as providing support for interfaces that are not known at compile time requires functionality very close to marshaling, which is part of phase 3.


amsterCHEM
tailor-made engineering software solutions

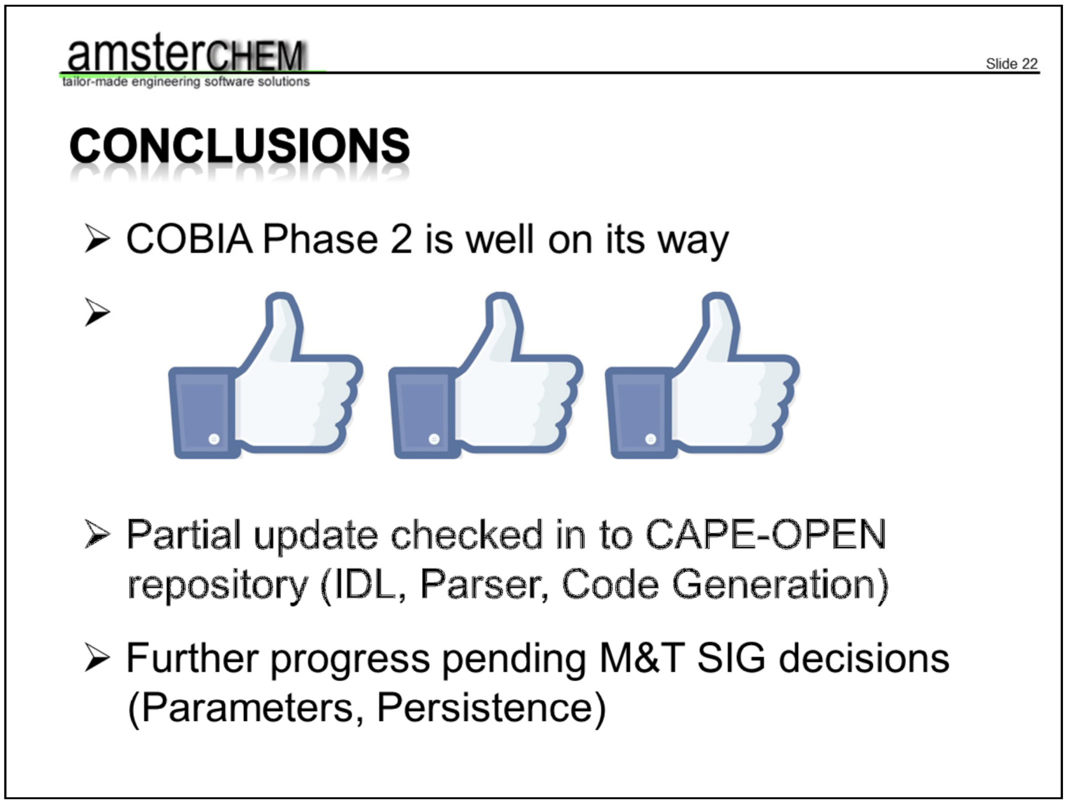
Slide 21

## COBIA DEVELOPMENT INSTALLER

- NullSoft Installer System (NSIS)
- Temporary – not the official CO-LaN installer
- Binaries:
  - COBIA middleware (+register)
  - Utilities (PMC registration, Code generation)
  - Code (headers, source)
- Windows only
- (work in progress)

The last delivery will be an installer. This will not be the official CO-LaN installer, but this installer is meant for people to get started developing, testing and contributing to COBIA prior to its release. It will be a Nullsoft installer, and it will be available on Windows only. Essentially this puts COBIA itself and the development headers in place. Optionally also the source and the test binaries.

The installer is not made yet. But that is of course not a lot of work.


amsterCHEM
tailor-made engineering software solutions

Slide 22

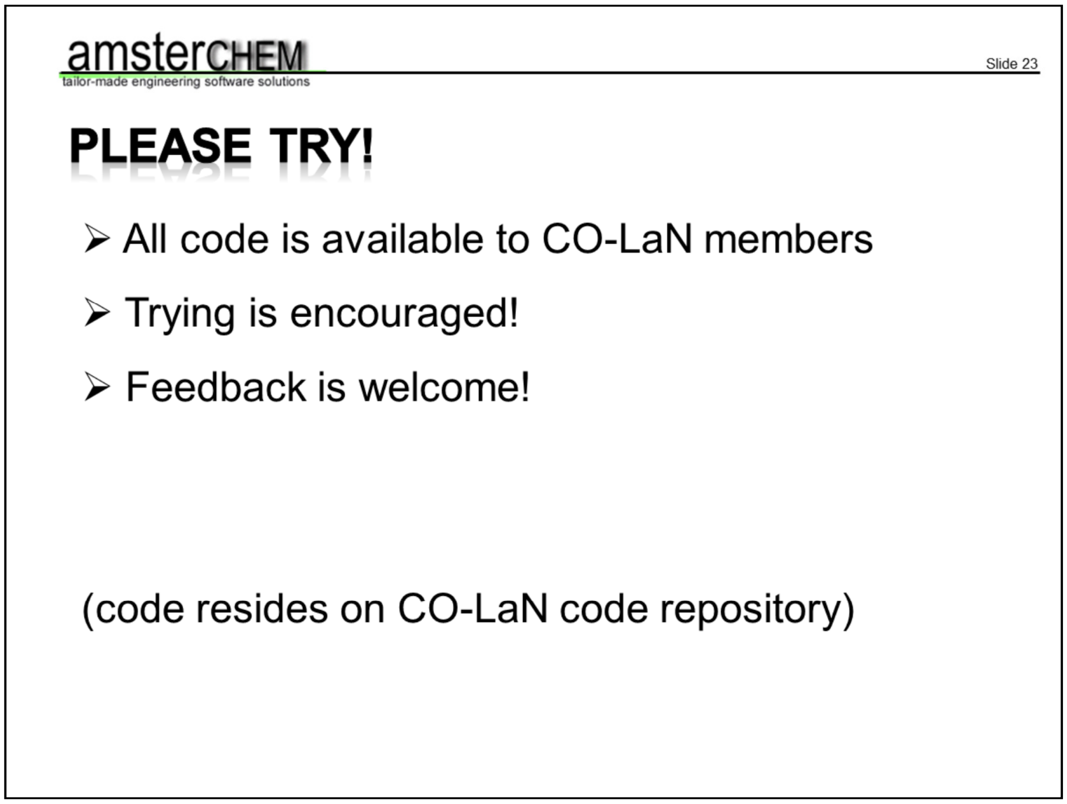
## CONCLUSIONS

- COBIA Phase 2 is well on its way
-
- Partial update checked in to CAPE-OPEN repository (IDL, Parser, Code Generation)
- Further progress pending M&T SIG decisions (Parameters, Persistence)

To conclude. COBIA phase II is well on its way – most of the coding is done. It is just a matter of agreeing on the content of the IDL.

Phase I has already shown that COBIA meets all its targets of system independent, less error prone, easier development of CAPE-OPEN components. For me it remains unchanged that COBIA gets three thumbs up, on a scale of three.

Most of the code is checked in already into the CAPE-OPEN repository, and the rest will follow as soon as the M&T SIG converges on the parameter and persistence interfaces.


amsterCHEM
tailor-made engineering software solutions

## PLEASE TRY!

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

(code resides on CO-LaN code repository)

Please try the code that is there. COBIA is not a hobby of the M&T SIG, we want COBIA to be the basis underlying all CAPE-OPEN interactions in the future. This will only work if we all agree on how it works. So try. The code is available to all CO-LaN members. If you have not yet access to the repository, please ask to set that up for you. And please let us know what you think.