Poseidon House
Castle Park
Cambridge CB3 0RD
United Kingdom

TELEPHONE: **Cambridge (01223) 515010**
INTERNATIONAL: **+44 1223 515010**
FAX: **+44 1223 359779**
E-MAIL: **apm@ansa.co.uk**

ANSA Phase III

COM - overview and analysis

Andrew Watson

Abstract

This slide set provides a short technical briefing on the salient features of Microsoft's COM, and compares it with the OMG's OMA & CORBA specifications.

It is intended for a technically-aware audience with existing knowledge of CORBA.

The slides are in colour.

APM.1407.00.01 **Draft** 20th February 1995

Request for Comments (confidential to ANSA consortium for 2 years)

Distribution:
Supersedes:
Superseded by:

COM
Overview and analysis

Andrew Watson
APM
ajw@ansa.co.uk

What does COM stand for?

- **Component Object Model**
  - Object Model and binary interface specification that underlies OLE2 Compound Document Architecture
- **Common Object Model**
  - Something subtly different, apparently created in collaboration with DEC
  - Aiming to be more of a *System Object Model*?

Where does it come from?

- **Microsoft**
  - **What though they speak of " ... the Open Process fostering broad industry participation ..."**
- **"OLE forms the basis for Microsoft's strategy to evolve the Windows family into object-based operating systems"**
- **Current manifestation is as the OLE2 DLLs**
  - **OLE2 also currently available on Macintosh**
  - **"In the future software components based on OLE will also be able to interoperate across all major versions of Unix, VMS, and even MVS ..."**
- **Perceived to be in competition with:**
  - **SOM/DSOM (underlies OpenDoc)**
  - **CORBA/OMA (*pace* efforts to formalise COM/CORBA interworking)**

What's a Compound Document Architecture?

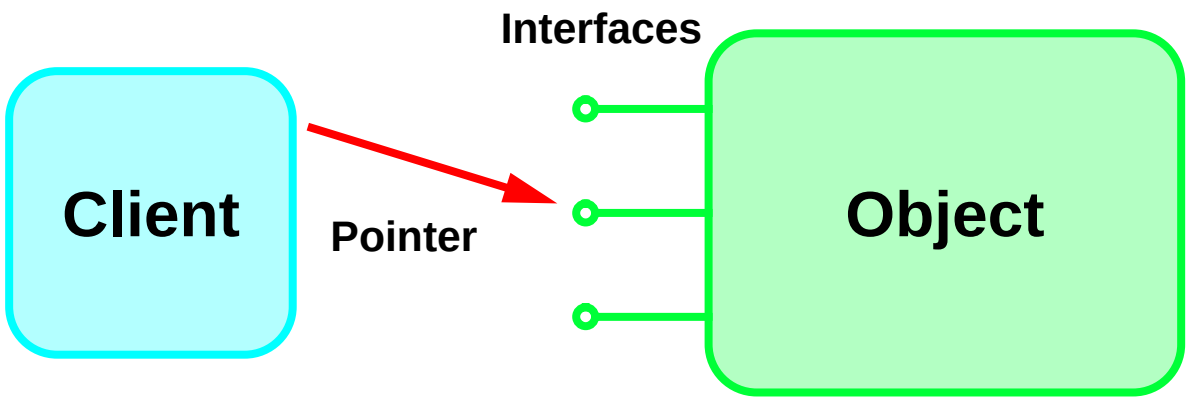
- **In the past desktop applications have stood alone**
  - **Inter-application communication restricted to file import/export, cut-and-paste**
- **Suite ("Works") applications have component document types that can be embedded in each other**
  - **e.g. place a spreadsheet or drawing frame in text column**
- **A CDA is effectively a suite application which permits adding new component document types**
  - **Inter-component communication protocol must be published**
  - **"Messages" to which particular classes of component respond must be standardised**

Technical highlights

- **Essentially a “classical” object model**
  - **Synchronous invocation (operation name + parameters) directed to single object**
- **One major variation - objects may have multiple “interfaces”**
- **Storage management via a reference counting scheme**
- **No run-time inheritance structures of any kind**
  - **“Object-based” in Wegner’s taxonomy**
- **Object creation via factory interfaces**
- **Binary specification**
- **Distribution transparency**

Interfaces

- **Interface is collection of operations**
  - **Named by UUID (“globally unique” 128 bit value)**
- **Interfaces conform to a binary specification**
  - **There is also an IDL, derived from OSF IDL**

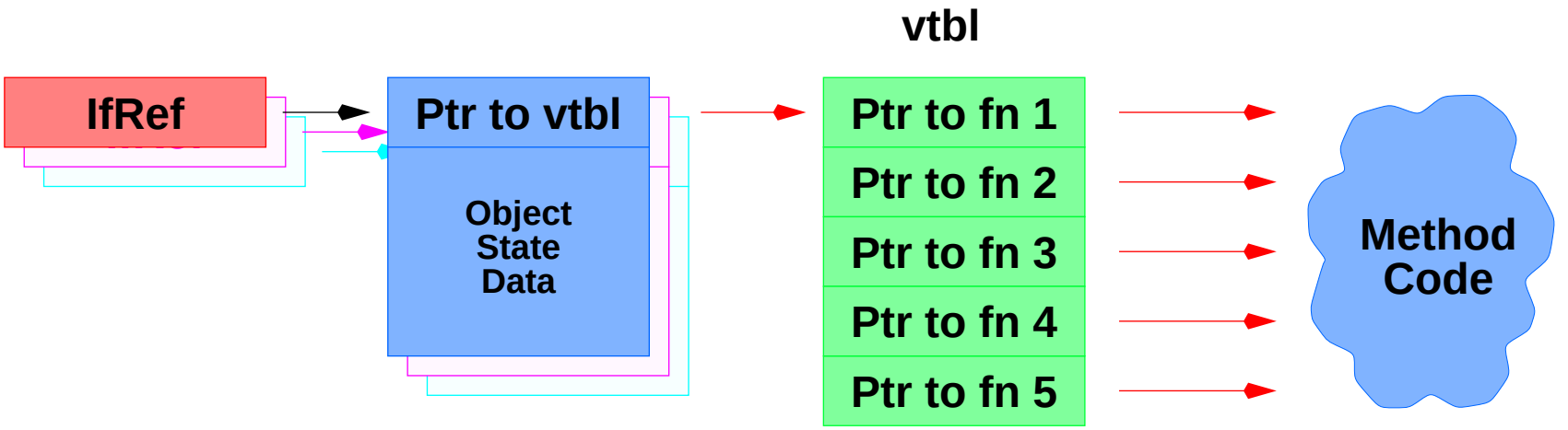


Why interfaces?

- **COM's designers concerned about version handling**
  - **"Does my Windows 3.x application work with Windows 95?"**
- **Interfaces allow new applications to present the old application's interface as well**
- **By strict convention, every interface includes an operation `QueryInterface`**
  - **When invoked with UUID of interface, returns ifref if this object supports it**
- **Also by strict convention, every object carries a standard interface IUnknown**
  - **Includes `QueryInterface`**
- **Ambition is to handle "find-and-bind" in a few dozen cycles**

Inside interfaces

- **At the binary level, an interface is a pointer to an array of function pointers**
  - **Array termed *vtbl***
- **Local case:**

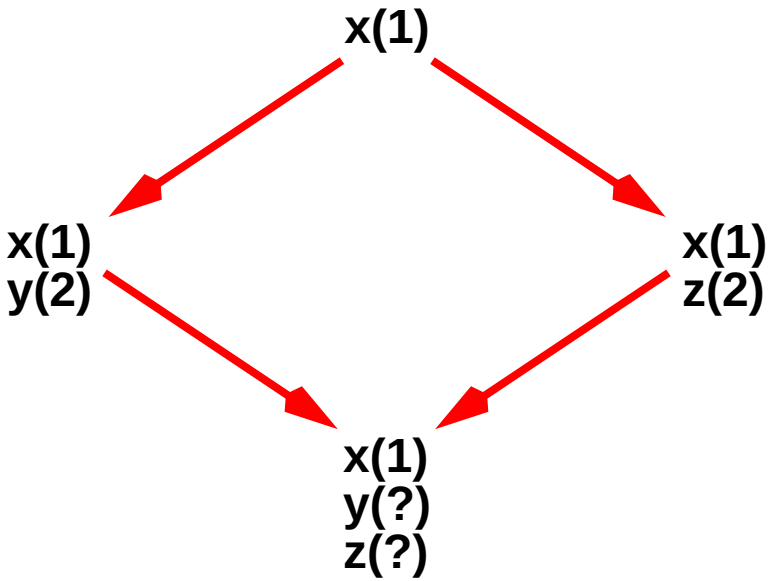


Inside Interfaces (cont.)

- **In remote case, method code is set of proxies that marshal parameters and transmits them**
  - Proxy “object state” is actually addressing information for remote object
  - COM provides DLL of marshalling methods for standard interfaces
  - Users can compile NDR stubs for their own interfaces ...
  - ... or use completely home-brewed marshalling if they wish
- **Receiving stub marshals and calls target object**
  - “Remote” object could be in another process (use shared memory)
  - ... or another machine (MS propose to use DCE RPC)
- **Truly distributed COM still lies in the future**
  - Currently only using ‘LRPC’ transmission

Interface tables

- **Interface function table is always contiguous**
- **Operations are identified at run time by offset in vtbl**
- **Hence no multiple supertypes:**



Interfaces vs subtyping

- One COM interface can be labelled with multiple UUIDs
  - `QueryInterface` can return same interface in response to many UUIDs
- Clients can treat interface as if it has fewer operations
  - i.e. use interface as if it were a supertype
- But operations looked up by offset, not by name
  - Makes ST-style polymorphism very impractical
- Intention: all attempts to “cast to supertype” done by passing UUID for required interface (type) to `QueryInterface`

Sidelight on multiple interfaces

- **Every interface shares code with other interfaces of the same “type”**
  - **State is localised by being kept with the ptr to the vtbl**
- **Why not allow one object to have multiple vtbl pointers for the same set of method code?**
  - **i.e. multiple instances of the same interface, with different state**
- **Only prevented by design of `QueryInterface`**
- **A useful technique that allows multiple clients to be distinguished by what interface they invoke**
  - **Example: abstraction of Unix file and file handles**

Invocation model

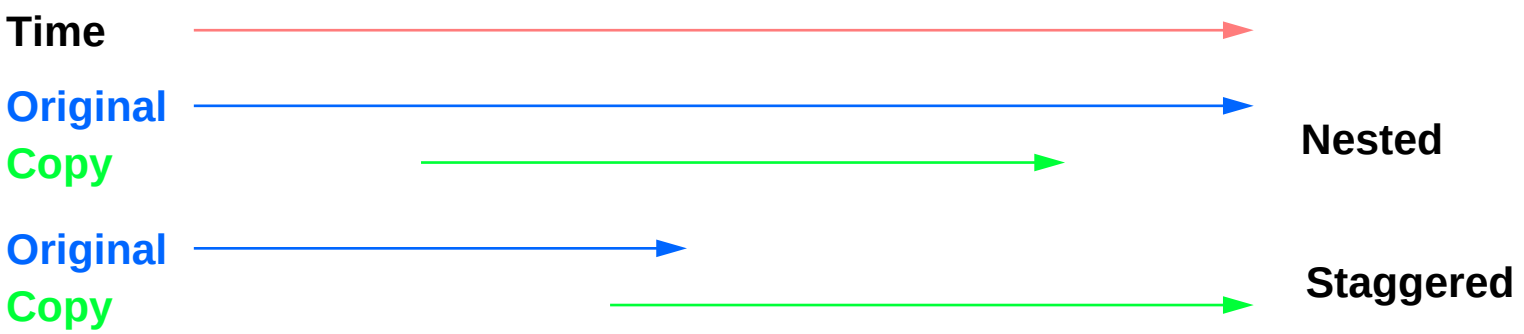
- **Synchronous, blocking call [AFAIK]**
- **Relies on availability of threads in servers and clients**
  - **Single-threaded client may block for extended periods during remote invocations**
  - **Single-threaded server could deadlock if it indirectly calls one of its own methods**
- **COM spec includes API for generating server threads**

Storage management

- Done via reference counting
- Every interface carries the following operations:

```
AddRef
Release
```

- In general, `AddRef` is called every time ifref is copied, `Release` when an ifref is destroyed
- Some recommended short-cuts



Persistent storage

- **COM defines an OO abstraction of structured, transactional storage**
  - **Could be mapped onto a variety of media**
- **Objects use instances of this interface to read/write byte sequences**
  - **Client can ask object to save its state to provided storage interface**
- **A related set of conventions for uniform data transfer**
  - **Standard “clipboard format” for information interchange**

Re-use mechanisms

- In general, two sorts of re-use
  - Re-use of common components
  - Modifying code templates to produce new sorts of components
- COM says nothing about compile time re-use via code modification
- Spec mentions two enhancements to component re-use
- Containment - hide re-used object inside container object
  - Leads to many almost-empty methods just to call contained object
- Aggregation - container passes off contained object methods as its own
  - COM has support for delegating admin. functions (`QueryInterface`, ref. counting) back to container

Miscellany

- **Multiple results via out and in/out parameters**
  - **Just like CORBA**
- **Monikers**
  - **Objects that provide location services**
  - **Will track down and re-activate object to which they refer if it has moved or been passivated**
- **IAdviseSink interface**
  - **Implements asynchronous notification service**
- **Exceptions**
  - **Handled via 32-bit result code and conventions**

Analysis

- **No security mechanisms**
  - Is an object allowed to refuse to hand out one of it's interfaces?
- **Why no multiple interfaces of same "type"?**
- **Reference counting better than nothing**
  - But create/delete put load on object
  - Weighted reference counts would have been better
- **Objects do own management functions**
  - Doesn't rule out repositories, though
- **Very little tool support**

Some Conclusions

- **It's an object model and ORB, with some Object Services and Common Facilities functions**
- **As long as it's only inside OLE, no-one will use it**
  - **How many of us write compound document parts?**
  - **Lack of programmer support (leverage concept)**
  - **Use will be indirect, via OLE automation**
- **It's designed for a homogeneous world**
  - **COM as "assembler" vs CORBA as "HLL" analogy**

COM vs OMA/CORBA

- **COM is a single-vendor binary specification**
  - **Advantages:** Binary portability
    Total compatibility
  - **Disadvantages:** Single-platform (albeit the most pervasive)
    Single-vendor
    Difficult to code for (leverage)
    Not distributed yet (call us back in October)
- **OMA/CORBA is open, source-level standard**
  - **Advantages:** Source-code portability across platforms
    Plenty of tool support for small-volume app. authors
    Part of an architecture with wider set of services
    No single-vendor lock-in
  - **Disadvantages:** No binary portability
    Specification ambiguities make portability less easy

References

Microsoft Object Technology Strategy; Component Software Strategic White Paper
098-55163, June 1994

Introduction to Common Object Model Specification
OMG Document 94-10-9