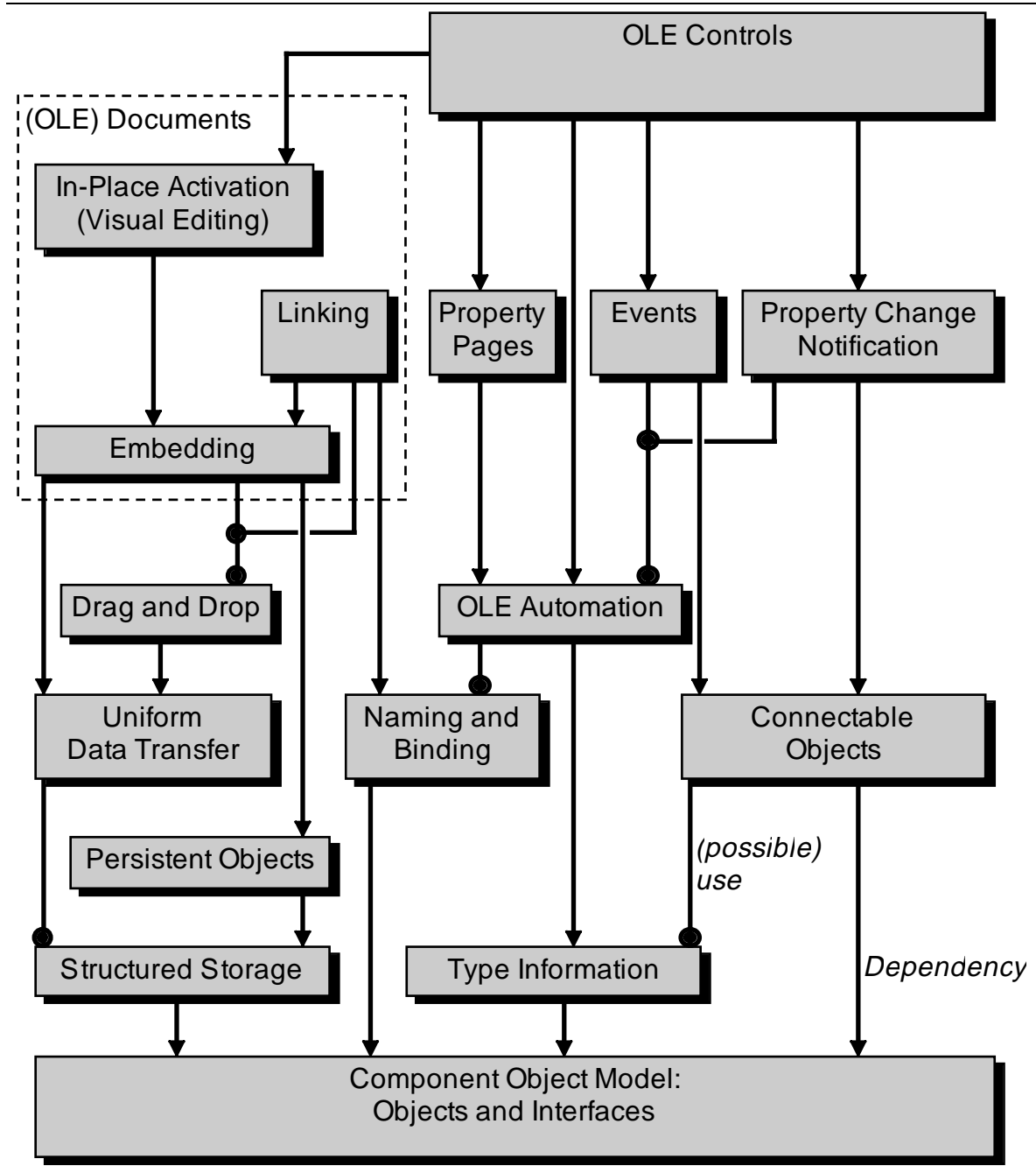


# COM, DCOM, ActiveX

- Components a la Microsoft
  - Compare with OpenDoc
- Trying to solve the same problem
  - Building programs using separately deliverable components
- Technology buzzwords
  - COM: Common Object Model
  - DCOM: Distributed COM
  - OLE: Object Linking and Embedding
  - ActiveX
- Microsofts usage
  - Building blocks of WinXX & applications
- Relation with OpenDoc
  - COM: bottom-up additions of solutions, ad-hoc
  - OpenDoc: 'top-down', clean development of distributed components

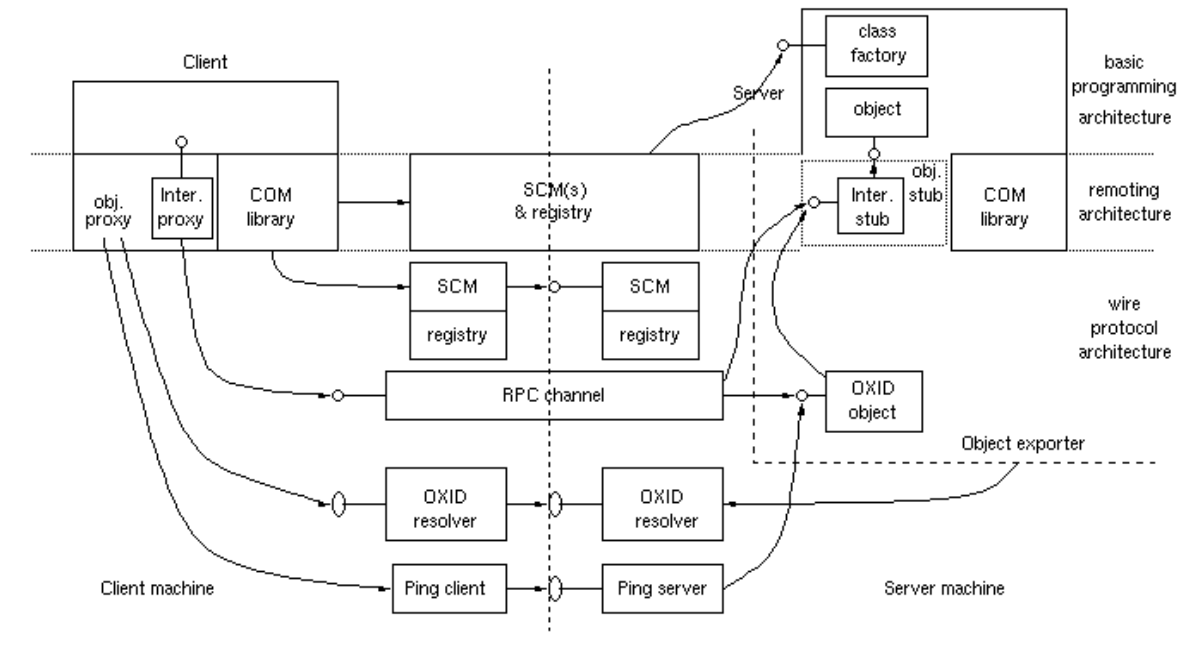
# Overview

- The big picture



# (D)COM Architecture

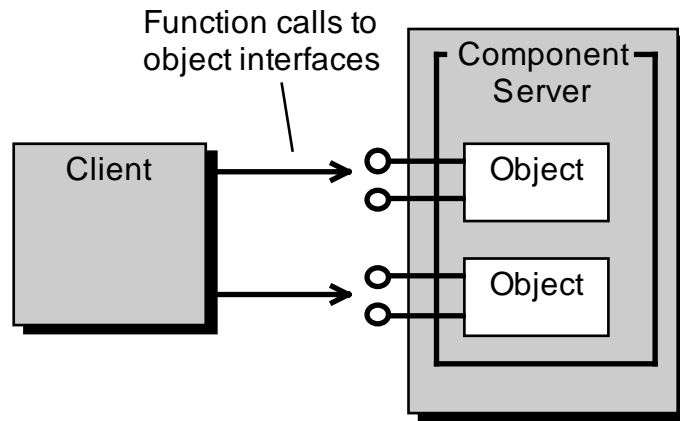
- Overview



- **COM/DCOM**
  - Compare with CORBA
- **The rest (OLE, ...)**
  - Compare with OpenDoc

# Component

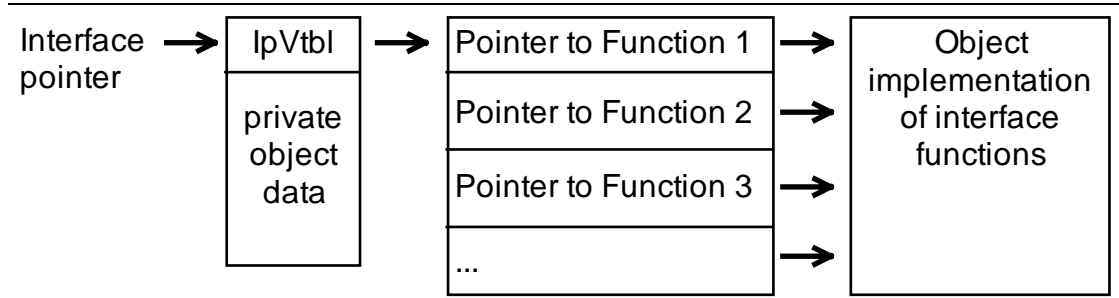
- What is a component (in this context)?



- Client code accesses functionality
  - Functionality is offered via interface
  - Basically: client/server
- Interface
    - Set of functions

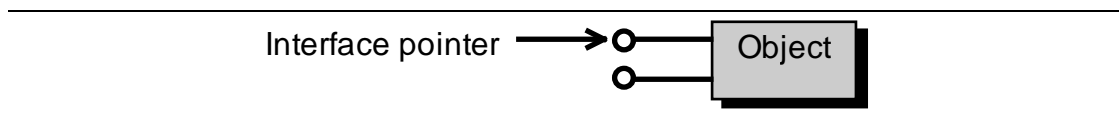
# Interface

- Binary standard and usage



- Client uses interface pointer
- Points to function table + object data
- Basically: C++ implementation

- Client sees:



- But

- Interface pointer is **not** the object (no identity!!)
- Interface pointer is 'channel' to object

- But what is then identified?

- Interface has 128 bit globally unique identifier (IID: Interface id)
- Class: group of interfaces, also has id (CLSID: Class id)

# Interface

- Object supports multiple interfaces
  - E.g. `IDataObject`, `IPersistent`, I...
  - Convention: interface name starts with 'I'
- Object always supports a 'minimal' interface
  - `IUnknown`
- `IUnknown` offers
  - `QueryInterface` for finding out if another interface is supported
  - `AddRef`, `Release` for reference counting

# Interface

- Written in an IDL (Interface Description Language)

```
// uuid and definition of IGrid1
[ object,
  uuid(3CFDB283-CCC5-11D0-BA0B-00A0C90DF8BC),
  helpstring("IGrid1 Interface"),
  pointer_default(unique)
]
interface IGrid1 : IUnknown {
  import "unknwn.idl";
  HRESULT get([in] SHORT n, [in] SHORT m, [out] LONG *value);
  HRESULT set([in] SHORT n, [in] SHORT m, [in] LONG value);
};
```

- Grid of values example

- Interface is given unique id (via uuid)

- Calculated by external tool UUIDGEN
- Based on time + location

- Immutable

- Once defined, never changed
- Version management by adding new interfaces (gives cluttering ...)

- Result of function

- Return code
- Exceptions have to be arranged otherwise

# Interface

- Interface: abstract class

```
[ ...  
]  
interface IGrid2 : IUnknown {  
    import "unknwn.idl";  
    HRESULT reset([in] LONG value);  
};
```

- A second one

- Can be combined into concrete class

```
// uuid and definition of type library  
[ uuid(3CFDB281-CCC5-11D0-BA0B-00A0C90DF8BC),  
  version(1.0),  
  helpstring("grid 1.0 Type Library")  
]  
library GRIDLib  
{  
    importlib("stdole32.tlb");  
    // uuid and definition of class  
    [ uuid(3CFDB287-CCC5-11D0-BA0B-00A0C90DF8BC),  
      helpstring("Grid Class")  
    ]  
    // multiple interfaces  
    coclass CGrid  
    { [default] interface IGrid1;  
      interface IGrid2;  
    };  
};
```

- A la multiple inheritance

# Server implementation

- A method implementation

```
STDMETHODIMP CGrid::get(IN SHORT n, IN SHORT m, OUT LONG*  
                        value)  
{  
    *value = m_a[n][m];  
    return S_OK;  
}
```

- But also IUnknown's interface

```
STDMETHODIMP CGrid::QueryInterface(REFIID riid, void** ppv) {  
    if (riid == IID_IUnknown || riid == IID_IGrid1)  
        *ppv = (IGrid1*) this;  
    else if (riid == IID_IGrid2) *ppv = (IGrid2*) this;  
    else { *ppv = NULL; return E_NOINTERFACE; }  
    AddRef(); return S_OK;  
}
```

- Each object must be able to answer "do you implement this interface?"

- With a factory for creating

```
STDMETHODIMP  
CClassFactory::CreateInstance(LPUNKNOWN p, REFIID riid, void**  
                             ppv) {  
    IGrid1* punk = (IGrid1*) new CGrid(100, 100);  
    HRESULT hr = punk->QueryInterface(riid, ppv);  
    punk->Release();  
    return hr;  
}
```

# Server implementation

- Starting the server

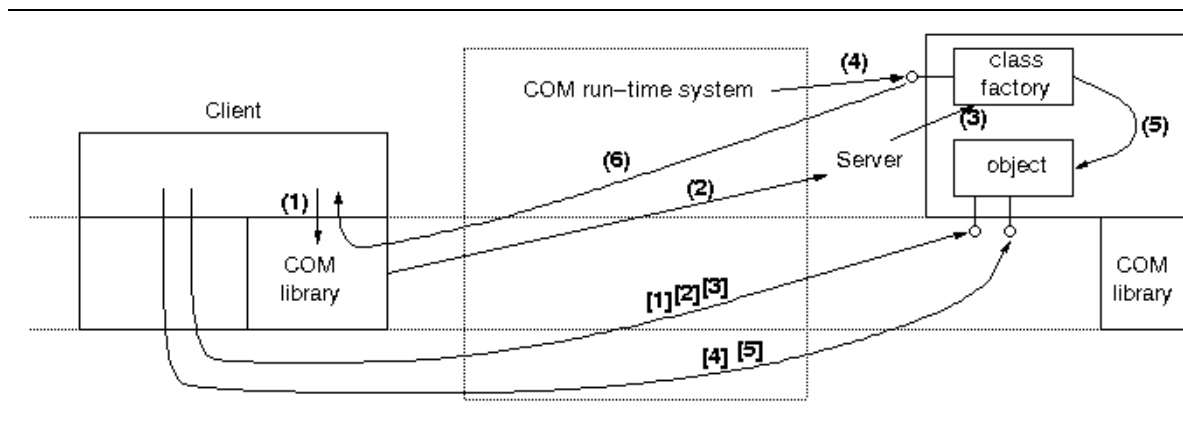
```
HANDLE hevtDone;
void main()
{
    // Event used to signal this main thread
    hevtDone = CreateEvent(NULL, FALSE, FALSE, NULL);
    hr = CoInitializeEx(NULL, COINIT_MULTITHREADED);
    CClassFactory* pcf = new CClassFactory;
    hr = CoRegisterClassObject(CLSID_CGrid, pcf,
        CLSCTX_SERVER, REGCLS_MULTIPLEUSE , &dwRegister);
    // Wait until the event is set by CGrid::~CGrid()
    WaitForSingleObject(hevtDone, INFINITE);
    CloseHandle(hevtDone);
    CoUninitialize();
}
```

# Client implementation

- The usage

```
void main(int argc, char**argv)
{
    IGrid1 *pIGrid1;
    IGrid2 *pIGrid2;
    LONG value;
    CoInitialize(NULL); // initialize COM
    CoCreateInstance(CLSID_CGrid, NULL, CLSCTX_SERVER,
                    IID_IGrid1, (void**) &pIGrid1);
    pIGrid1->get(0, 0, &value);
    pIGrid1->QueryInterface(IID_IGrid2, (void**) &pIGrid2);
    pIGrid1->Release();
    pIGrid2->reset(value+1);
    pIGrid2->Release();
    CoUninitialize();
}
```

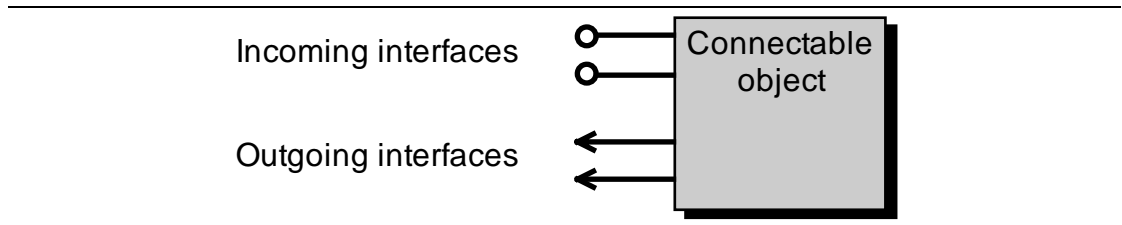
- An overview



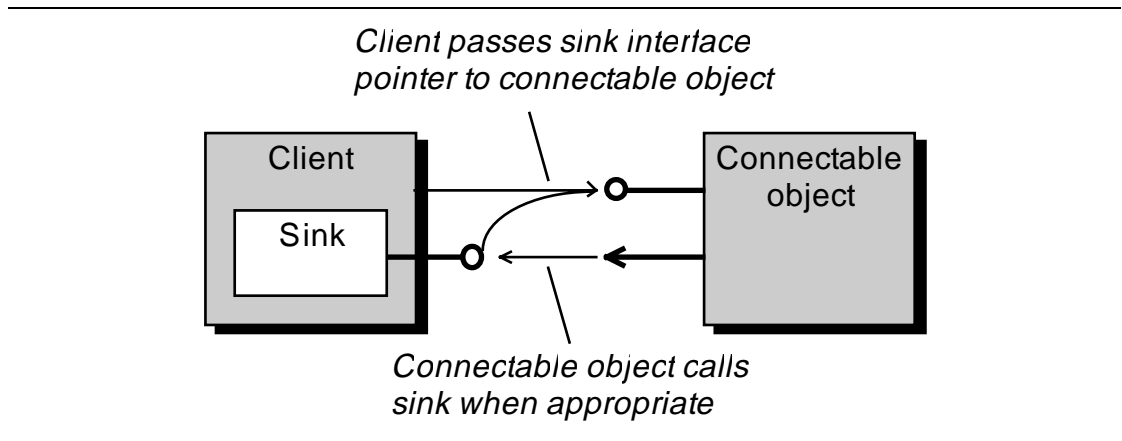
- Finding the executable/library for a class
  - Using the WinXX registry

# Connectable objects

- Kinda Event service, observer pattern



- Incoming interfaces
  - The 'normal', called as methods
- Outgoing interfaces

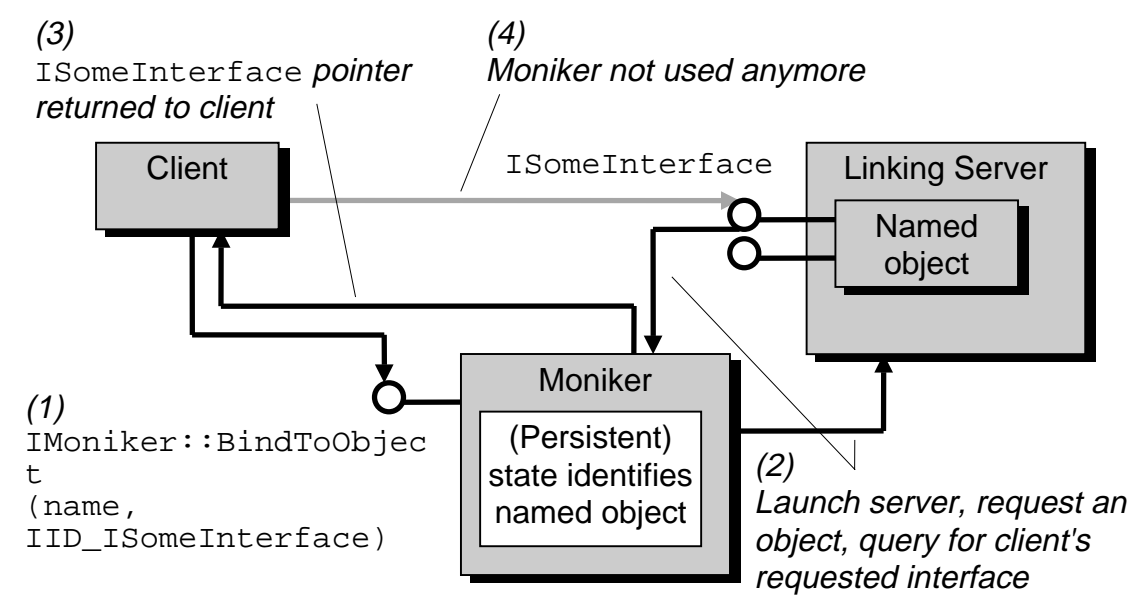


- Implemented elsewhere
  - But specified as 'needed' in connectable object
- Connectable object has to implement

```
interface IConnectionPointContainer : IUnknown
{
    HRESULT EnumConnectionPoints( IEnumConnectionPoints ** ) ;
    HRESULT FindConnectionPoint( REFIID, IConnectionPoint ** ) ;
}
```

# Naming

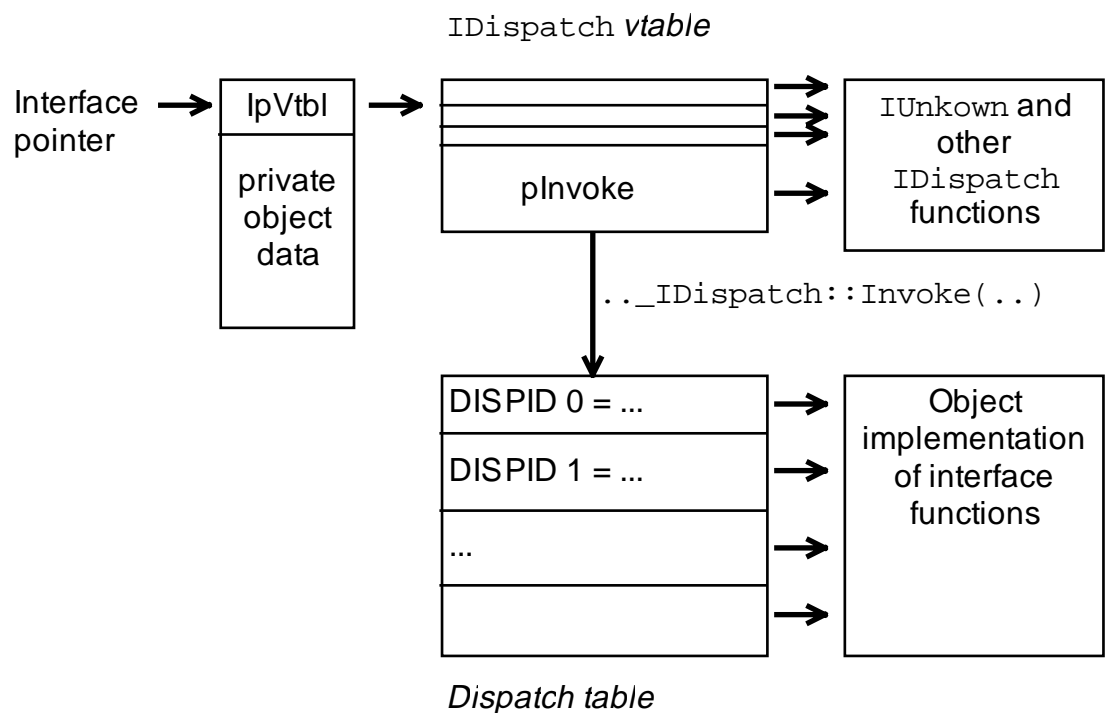
- Using intermediate components
  - Moniker
  - Kinda pointer, but more general
  - But: not a naming service
- Indirection



- Moniker variants
  - File: manages a pathname
  - Pointer: non-persistent indirection
  - And others, e.g. for composites

# Automation

- Kinda scripting/macro's
  - Controlling controllable components
- Controllable component
  - Offers interface IDispatch
  - Compare with CORBA's dynamic invocation
- Using extra indirection



- And
  - Standardized structures for parameter/result

# Compound documents

- OLE: Object Linking and Embedding
  - Containers containing containable components
- Used in almost any Microsoft application
- Structured storage
- Persistency
- Linking
- Drag and drop
- In-place editing

# Other features

- Type libraries
  - TLB is Abstract Syntax Tree of IDL
  - Interface available for querying TLB
- Marshalling (with RPC)
  - Customization allowed
  - Normally done with IDL compiler

# Other issues

- Inheritance
  - Replaced by aggregation, containment
  - Is it object-oriented?
- Distribution
  - On top of (longer existing) DCE, RPC package
  - When object cannot be found locally, it will be searched on other machines in the network.
  - Through a utility remote objects (servers) can be registered
  - In registry info about remote objects
  - At runtime proxies will be used.

# Visual J++ en COM

- Java Type Library Wizard generates interface and class files from TLB (Type Library)

```
import comserver.*;    // import comserver
- of

import comserver.IComBeeper;
import comserver.CComBeeper;

IComBeeper testBeep = (IComBeeper)new
CComBeeper();

int myTone = testBeep.getSound();

testBeep.putSound(64);

testBeep.Beep();
```

# COM Server in Java

- use the JavaTLB tool to generate .CLASS files based on the type library.
- create a .JAVA source file containing the Java implementation of the class.

```
import com.ms.com.*;
import mytroupe.*;

public class MyPerformer implements
    ICanSing, ICanDance
{
    public void sing() throws ComException {}
    public void hum() throws ComException {}
    public void dance()
        throws ComException {}
    public void shuffle()
        throws ComException {}
}
```

- COM clients can read the .TLB file to determine what services your class exposes