COM and ATL

An Introduction

彭卫国

德国 海德堡

wei-guo.peng@sap.com
COM and ATL

- **Common Object Model**
  - Published specification (Open standard)
  - Component technology
  - Integrated into Win32 OS

- **Active Template Library**
  - Tool to create COM servers in C++
  - Based on C++ template
  - Generate common C++ codes
Brief History of COM

- 1980: application linking running on the same desktop, lightweight protocol for data "Pez"
- 1989: OLE (Compound Document)
- 1996: ActiveX Objects not simply static, COM (MSMQ, MTS)
- 1997: COM+
COM: Common Object Model

DCOM: Distributed Common Object Model

OLE: Object Link and Embedding

ActiveX and OLE control

- Is a COM object
- Is UI – single-threaded
- Properties
Why COM?

- Language independent
  - Binary standard
  - Support C++, VB, Java, JavaScript, VBScript
- Robust versioning
  - Multiple interfaces
- Location transparency
  - GUID (globally unique identifier)
  - DLL (in process), EXE (out of process, distributed)
- Object orientation
  - Encapsulation, inheritance, polymorphism
COM and OOP

- COM is a binary standard, NOT source code standard
  - COM can be implemented in different languages
  - COM can be accessed using different languages
- OOP can use the libraries implemented only in the same language
COM Server and Client

- COM implementer – server
  - Mostly created with C++
  - Use ATL

- COM consumer – client
  - All kinds of languages C++, VB, ..
  - Internet (ActiveX control for client and COM on the server)
COM Interface

- Defines the methods, which can be called by COM clients
- Should not change after published
- Can be implemented differently
- COM interface is realized with Vtable
- Properties are supported with method calls
Interface Implementation

Component Object

Client Variable → VTBL pointer

private object data

VTBL

pointer to function
pointer to function
pointer to function

function1(pObj, arg1, arg2...)
{
    ...........
}

© W2W GmbH 2000 – COM and ATL (Dr. Wei-Guo Peng) / 9
interface IUnknown
{
    virtual HRESULT QueryInterface(IID& iid, void** ppvObj) = 0;

    virtual ULONG AddRef() = 0;

    virtual ULONG Release() = 0;
}

// Call QueryInterface on the component object PhoneBook, asking for a pointer
// to the Ilookup interface identified by a unique interface ID.

hRes = pPhoneBook->QueryInterface( IID_ILOOKUP, &pLookup);

if( SUCCEEDED( hRes ) ) {
    pLookup->LookupByName("Daffy Duck", &szNumber);
    // use Ilookup interface pointer
    pLookup->Release();
    // finished using the IPhoneBook interface pointer
}

else {
// Failed to acquire Ilookup interface pointer.
}
Versioning & Multiple Interfaces

![Diagram showing an object with interfaces A, B, and C]
Access a COM Object
Call a COM Object in C++

CoInitialize(); //Load COM DLL/EXE
CreateInstance(); //Create a COM object
QueryInterface(); //Fetch the interface
Release(); //Release an interface pointer
AddRef(); // Add an interface pointer
Release();
CoUninitialize(); //Unload COM
Dim CalcPi As PISVRLib.CalcPi
Set CalcPi = New PISVRLib.CalcPi
CalcPi.Digits = 5
Pi = CalcPi.CalcPi
Call a COM Object in JavaScript

<html>
<object classid = "clsid:C41A8166-614E-4EFC-AA16-73B88A2C9A7C"
       id = "obj" hight = "100%" width = "100%">
</object>
<body>
  <script language="JavaScript">
    obj.Digits = 5;
    var pi = obj.CalcPi();
    document.write("Pi: " + pi);
  </script>
</body>
</html>
In-process server
- DLL
- In the same process of calling COM client

Local server
- EXE
- Has own process

Service
- EXE
- Will be started as service (SCM)
In- and Out-of- Process

Client Process

- In-process Component Object
  - In-Process Server

Client Application

COM

Interprocess Communication

- LRPC
- Cross-process with lightweight RPC

- RPC
- Cross-network with true RPC

Local Server Process

- Local Object
- Remote Object

Remote Computer

- Remote Server Process
- Remote Object
COM: Reuse

- Containment
- Aggregation
COM Containment/Delegation

IUnknown knows A, B, and C

External Interfaces
A
B
C

Outer Object

Outer Object uses Inner Object’s C implementation as any client.

Unknown controls Inner Object lifetime

Inner Object: Contained inside Outer Object
COM Aggregation

Outer Object

Unknown controls Inner Object lifetime

Inner Object delegated unknown calls to Outer Object

Inner Object’s C exposed directly from Outer Object

Unknown knows A, B, and C

External Interfaces

A

B

C

© W2W GmbH 2000 – COM and ATL (Dr. Wei-Guo Peng) / 21
## COM Threading Model

<table>
<thead>
<tr>
<th>Single</th>
<th>COM will protect access to global, static, or instance data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appartment</td>
<td>COM will protect access to instance data only. Global and static data must be protected by COM programmer</td>
</tr>
<tr>
<td>Free</td>
<td>Multithreading model. You have to synchronize access to instance, global and static data.</td>
</tr>
<tr>
<td>Both</td>
<td>Adopt the threading model of the client.</td>
</tr>
</tbody>
</table>
COM: Object Lifetime

- Managed by client
- Language dependent
- Based on „Reference Count“ of COM object
- In C++
  - AddRef() / QueryInterface()
  - Release()
  - Interface pointer leak!
COM+

- COM+ = MTS
- Run time environment for COM
  - Distributed transaction
  - Integrated security (role based)
  - Threading pooling
  - Configuration and administration
- Support Multitier programming model
- Attribute-based programming
C++ Template

- Standard in C++ ANSI
- Generic way for reuse by allowing parameterized types
- Function template & class template
## Template and inheritance

<table>
<thead>
<tr>
<th>Inheritance</th>
<th>Template</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertical code reuse</td>
<td>Horizontal code reuse</td>
</tr>
<tr>
<td>Dynamic binding</td>
<td>Static binding</td>
</tr>
<tr>
<td>Large and slow</td>
<td>Small and fast</td>
</tr>
<tr>
<td>ATL</td>
<td>MFC</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>Creation of small &amp; COM-based components</td>
<td>Easy development of large, Windows-based applications</td>
</tr>
<tr>
<td>Depends on C runtime library &amp; atl.dll ~ 58KB</td>
<td>Depends on MFC DLL (mfc42.dll ~ 1MB)</td>
</tr>
<tr>
<td>No visual aspect</td>
<td>Lots of visual functions</td>
</tr>
<tr>
<td>Server component</td>
<td>Thick client</td>
</tr>
<tr>
<td>Difficult to understand</td>
<td>Easy to understand</td>
</tr>
</tbody>
</table>
### ATL: Features

<table>
<thead>
<tr>
<th>Class Wrappers</th>
<th>VARIANT, BSTR, HWND</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Basic COM interfaces</strong></td>
<td>IClassFactory, Idispatch, IPersistXxx, IConnectionPointContainer, IEnumXxx</td>
</tr>
<tr>
<td><strong>Classes for Managing COM</strong></td>
<td>Exposing class objects, self-registration, server lifetime management</td>
</tr>
<tr>
<td><strong>Wizards</strong></td>
<td>App Wizard, Object Wizard</td>
</tr>
</tbody>
</table>
Why ATL

- Class wrappers around data types
  - VARIANT, BSTR, HWND
- Handles the common work to create COM
  - Housing (DLL/EXE)
  - Class factory
  - IUnknown interface
  - Smart pointer
  - Smart types
- Wizard
  - AppWizard
  - Object Wizard
import "oaidl.idl";
import "ocidl.idl";

[object,
 uuid(03445A24-AB40-4965-B82C-E59371255E40),
 dual,
 helpstring("IHelloObj Interface"),
]
interface IHelloObj : IDispatch
{
    [id(1), helpstring("method Hello")]
    HRESULT Hello([out, retval]
    BSTR* msh);
}

[uuid(7BE967E5-B532-4E9C-8742-6B812B37252B),
 version(1.0),
 helpstring("HelloCOM 1.0 Type Library")
]
library HELLOCOMLib
{
    importlib("stdole32.tlb");
    importlib("stdole2.tlb");
    [
        uuid(4BF67CA6-0793-45F9-A7E0-E39E72326D5F),
        helpstring("HelloObj Class")
    ]
coclass HelloObj
{
    [default] interface IHelloObj;
};

© W2W GmbH 2000 – COM and ATL (Dr. Wei-Guo Peng) / 31
Vtable and Automation

- Vtable interface – IUnknown
  - Binding at compile time
  - Strongly typed language (C++, Java)
- Automation interface – IDispatch
  - Binding at runtime
  - No-type language (VB, scripts)
- ATL support dual interfaces
Automatic Marshaling

Client Process

- Client Interface Pointer
- Proxy

Server Process

- Component Instance
- Stub

Different machines
Different processes
Different Threads
ATL Smart Types

- **Strings**
  - OLECHAR: wchar_t
  - BSTR: fixed length of OLECHAR, _bstr_t
  - A2BSTR, W2A

- **VARIANT**
  - _variant_t

- Smart pointer
Error Handling in ATL

- ISupportErrorInfo
- CComCoClass::Error
- In ATL just return Error()
- With smart pointer, try-catch can be used
Wizards

- **IDL Wizard**
  - Method/property
- **Object Wizard**
  - ATL object class
  - Other classes
- **Class Wizard**
  - Member function
  - Implement an interface
  - Connection point
ATL Support of COM Server

- Register and Unregister
- Exposing class objects
- Manage server’s lifetime
Other support by ATL

- Connection point & Event
- Collections and Enumeration
- ActiveX Control
- Persistence
Conclusion

- COM is a key technology for Windows. It is small and fast
- COM is especially useful for back-end server side development
- Think of using COM+/MTS for large projects based on COM
- Use ATL to write COM
- Use smart pointer (avoid Iptr leak!)
- COM/ATL is not easy to use as JavaBeans. It requires experienced programmers