COM & COM+
(Component Object Model)

Bazsó-Dombi András,
Group 251.
What is COM? – Low level

• Objects created independently need to be used in other applications
• Create an object, add some special “attributes” to make it COM-compatible, store it in a DLL.
• In the using application, add a reference to the DLL.
• Use the external object transparently, just as if it would have been written within the same application.
• Some special object services that facilitate the use of COM objects and increase productivity.
What is COM? – High level

• Microsoft’s object interoperability model (architecture)
• Supports (obviously) component-based systems
• Limited to MS Windows systems (↔ CORBA)
• Original COM limited to a single machine, DCOM, COM+ distributed
• Object transparency (objects created in different programming languages, stored anywhere, used as if they were part of the using application)
The evolution of COM

- Clipboard 1987
- OLE 1992
- COM 1995
- Distributed COM 1996
- Microsoft Distributed Transaction Coordinator 1996
- Microsoft Transaction Server 1997
- Microsoft Message Queue Server 1997
- COM+ 1999
The evolution of COM

1. The Clipboard
   - The first form of inter-application communication under Windows
   - Problem: if something changed, it had to be re-pasted

2. DDE (Dynamic Data Exchange)
   - Message-passing between applications existing in early Windows versions
   - Problem: complicated protocol to follow
The evolution of COM

3. OLE (Object Linking and Embedding)
   - Easily create complex documents with linked/embedded external objects
   - Auto-update of objects when changed

4. OLE 2
   - Linked/embedded objects seen as small software components
   - Built around the idea of component based-software
The evolution of COM

5. COM

• The “glue” between independent software components that allows them to interact easily

• Component-based philosophy
  • Reduce component complexity
  • Facilitate reuse
  • Facilitate maintenance
  • Component creation transparency
  • Component location transparency
COM+

- 2 major parts:
  - An architecture for building software components as defined by the original COM
  - An integrated suite of component services with an associated runtime environment for managing the objects
    - This is meant to offer implementation of component/object-related tasks commonly used by applications, reducing development time, increasing productivity
COM+ component services

• Automate the tasks commonly associated with the use of components, by offering built-in
  – Authorization/authentication checking (security)
  – Transaction management
  – Network operations (location transparency)
  – Threading/concurrency management
  – Integrity check and failure detection
  – etc.
COM+ component services

• Components using these services, executing within the associated runtime environment are called *configured components* (--> see Control Panel/Administrative Tools/Component Services) and have a lot of advantages

• Components not using these services are called *non-configured components* and have the benefit of eliminating run-time overhead of the execution architecture.
COM+ component services

1. Just-In-Time Activation (JITA)
   - When a client attempts to instantiate a COM object, he receives a reference to a context object implemented by COM+, not a reference to the COM object itself.
   - The actual object will be instantiated only when a method call is issued.
   - This way objects are created only when necessary, reducing system load, allowing for greater system scalability (more clients using the COM+ objects).
COM+ component services

1. JITA (continuation)
   - The opposite of JITA is early deactivation. COM+ can deactivate objects even if clients still maintain references to them (to free system resources, when objects are not used for a longer time). When clients start referencing the object again, it is reactivated without transparently, the client won’t even notice.
COM+ component services

2. Object pooling
   - To allow for even greater scalability, the COM+ runtime environment maintains a pool of active objects. When the client releases an object, it is not destroyed, but kept in the pool and recycled when another client requests an object of the same type.
   - Especially useful when client applications use COM+ objects frequently, but for short times (the common case)
COM+ component services

3. Load balancing
   - Client workload is distributed among multiple servers in the network
   - When a client requests an object, a load balancing router chooses the server to create it on. The client receives the reference to the object on that server, using it directly in the future.
   - If the server goes down, the reference to the object is automatically replaced with one on another, running server
COM+ component services

4. In-Memory DataBase (IMDB)
   - Transaction-based, database-style cache, that allows extremely fast access to local data.
   - Clients see it as a normal database system through high-level data access components, such as ADO.
   - Implemented as an OLE DB provider
   - The databases are automatically generated by COM+ and/or loaded/saved to persistent storage devices
COM+ component services

5. Queued components
   - Implemented using the MicroSoft Message Queue (MSMQ) server.
   - A queue records method calls for a given object even if it is offline and executes them when the object becomes available.
COM+ component services

6. Transactions

– COM+ objects can participate in distributed transactions

– A transaction is initiated when a critical resource is to be used (such as a database, but not only)

– The mechanism relies on Microsoft Distributed Transaction coordinator (DTC), originally developed for MS SQL server, today integrated into Windows

– Uses X/Open DTP XA two-phase commit protocol standard
COM+ component services

7. Role-based security
   - Authentication is role-based (the administrator can create roles for components (even per-interface roles) and assign users to them, define their access level, etc.)
   - If the declarative security is not fine-grained enough, components can include programmatic security checks (also role-based)
COM+ component services

8. Events

- COM+ components can deliver messages to other COM+ components through events, without knowing their identity.
- The event service model is an external one, event senders and receivers implement as little event handling as possible, the administrator can configure components from outside to send/receive events.
- Receivers only need to implement a particular interface.
COM+ structure

- Load balancing
- In-memory database
- Object pooling
- Queued components
- Event model
- Better administration

---

**Microsoft Transaction Server**
- Transaction services
- Resource pooling
- Role-based security
- Administration
- Just-in-time activation

**Distributed COM**
- Remoting architecture
- Distributed component services

**COM**
- Interface-based programming
- Basic component facilities
COM interfaces

- COM objects expose their functionality through interfaces
- Clients are aware of interfaces contained in a COM component and create COM objects by referring to these interfaces
- Interfaces are identified by GUIDs.
- The most common COM interfaces are `IUnknown` (implemented by all COM classes) (`00000000-0000-0000-C000-000000000046`) and `IClassFactory` (`00000001-0000-0000-C000-000000000046`)
COM interfaces

- COM interfaces are language-neutral. The COM classes can be written in any language but the interfaces are described in an IDL (Interface Definition Language), which is similar to C++

- Example:

```plaintext
import "unknwn.idl";
[ object, uuid(10000001-0000-0000-0000-000000000001) ]
interface ISum : IUnknown{
HRESULT Sum([in] int x, [in] int y, [out, retval] int* retval);
};
```
Type libraries

- Type libraries are the binary correspondent of IDL files (compiled IDLs)
- Most programming environments have built-in support for type libraries
- COM+ interfaces for type libraries:
  - Creating: ICreateTypeLib, ICreateTypeInfo
  - Reading: ITypeLib, ITypeInfo
Type Libraries

• Can be embedded into component DLL files by adding them into a resource in the project

• Need to be registered under HKEY_CLASSES_ROOT\TypeLib using LoadTypeLibEx or RegisterTypeLib (C++) (can be done not only programmatically)
## Type translation table

<table>
<thead>
<tr>
<th>IDL Type</th>
<th>C++</th>
<th>Visual Basic</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>signed char</td>
<td>signed char</td>
<td></td>
<td>byte</td>
</tr>
<tr>
<td>unsigned char</td>
<td>unsigned char</td>
<td>Byte</td>
<td></td>
</tr>
<tr>
<td>wchar_t</td>
<td>wchar_t</td>
<td>Integer</td>
<td>char</td>
</tr>
<tr>
<td>signed short</td>
<td>short</td>
<td>Integer</td>
<td>short</td>
</tr>
<tr>
<td>unsigned short</td>
<td>unsigned short</td>
<td></td>
<td></td>
</tr>
<tr>
<td>signed int</td>
<td>int</td>
<td>Long</td>
<td>int</td>
</tr>
<tr>
<td>unsigned int</td>
<td>unsigned int</td>
<td></td>
<td></td>
</tr>
<tr>
<td>signed hyper</td>
<td>__int64</td>
<td></td>
<td>long</td>
</tr>
<tr>
<td>float</td>
<td>float</td>
<td>Single</td>
<td>float</td>
</tr>
</tbody>
</table>
# Type translation table

<table>
<thead>
<tr>
<th>IDL Type</th>
<th>C++</th>
<th>Visual Basic</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>double</td>
<td>double</td>
<td>Double</td>
<td>double</td>
</tr>
<tr>
<td>BSTR</td>
<td>BSTR</td>
<td>String</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>boolean</td>
<td>bool</td>
<td>Boolean</td>
<td>boolean</td>
</tr>
<tr>
<td>VARIANT</td>
<td>VARIANT</td>
<td>Variant</td>
<td>com.ms.com.Variant</td>
</tr>
<tr>
<td>DATE</td>
<td>DATE</td>
<td>Date</td>
<td>double</td>
</tr>
<tr>
<td>CY</td>
<td>CY</td>
<td>Currency</td>
<td>long</td>
</tr>
<tr>
<td>SAFEARRAY</td>
<td>SAFEARRAY</td>
<td>[] (a standard Visual Basic array)</td>
<td>com.ms.com.SafeArray</td>
</tr>
<tr>
<td>IUnknown*</td>
<td>IUnknown*</td>
<td>IUnknown</td>
<td>com.ms.com.IUnknown</td>
</tr>
<tr>
<td>IDispatch*</td>
<td>IDispatch*</td>
<td>Object</td>
<td>java.lang.Object</td>
</tr>
</tbody>
</table>
The Active Template Library (ATL)

• Helps building COM+ components in C++
• Composed of C++ template classes that support the most common COM+ programming tasks, such as implementing the `IUnknown`, `IClassFactory`, and `IDispatch` interfaces, as well as dealing with component registration issues.
• Part of MFC&T (Microsoft Foundation Classes and Templates)
• Partially relies on STL (Standard Template Library)
Apartment Types

• Multithreading was introduced into COM pretty late, a way had to be found to make components thread-safe

• Apartments are the basic unit of thread-safety in COM+

• An apartment is a set of threading rules shared by a group of objects
Apartment Types

- 3 types:
  - STAs (Single-Threaded Apartments)
  - NAs (Neutral Apartments)
  - MTAs (Multi-Threaded Apartments)

- In STAs COM+ ensures thread safety by automatically synchronizing using Windows messages

- In NAs and MTAs thread safety must be ensured by the components itself
Hello World COM+ in C++
Step 1: Create a classic COM component

1. In Visual Studio .NET create a new C++ ATL project
   – Project settings:
     • Not attributed
     • COM 1.0 support not enabled

2. Add class wizard/Simple ATL object
   – Short name: Message
   – CoClass: Hello
Hello World COM+ in C++

Step 2: Create a COM+ application

1. **Open**
   Control Panel/Administrative Tools/Component Services/Computers/My Computer/COM+ Applications

2. **Right-click/New/Application**
   - Create empty application
     - Type: library application (in-process)
     - Name: Hello COM+

3. **Add the COM component to the COM+ application:**
   1. Right-click the Components folder
   2. New Component
   3. Select the DLL file of the COM component
Hello World COM+ in C++

Step 3: Create a C++ client

1. Create a C++ project (any type)
2. Add:

```cpp
#import "Hello.dll" no_namespace named_guids

::CoInitialize(NULL);
HRESULT hres = S_OK;
IMessage* pMessage = NULL;
hres = ::CoCreateInstance(CLSID_Hello,NULL,CLSCTX_ALL,
IID_IMessage,(void**)&pMessage);
hres = pMessage->ShowMessage( );
pMessage->Release( );
::CoUninitialize( );
```
Hello World COM+ in C++

Step 4: Create a Visual Basic client

1. Create a new Windows application project
2. Add the Hello.tlb type library to the project references
3. Add the code:

```vbnet
Dim obj As Hello
Set obj = New Hello
obj.ShowMessage
set obj = Nothing
```
Hello World COM+ in C++

Step 5: Create a C# client

1. Create a new project of any kind
2. Add the Hello.dll or Hello.tlb to the references
3. Add the code:

```csharp
HelloLib.Hello hello = new HelloLib.Hello;
hello.hello();
```
Sources

1. Guy Eddon, Henry Eddon:  
   *Inside Microsoft COM+ Base Services*  
   Microsoft Press, 1999

2. Juval Löuy:  
   *COM and .NET Component Services*  
   O’Reilly