301AA - Advanced Programming

Lecturer: Andrea Corradini

andrea@di.unipi.it
http://pages.di.unipi.it/corradini/

AP-10: Components: the Microsoft way
Overview

- The Microsoft approach to components
- DDE, OLE, COM, ActiveX, ...
- The .NET framework
- Common Language Runtime
- .NET components
- Composition by aggregation and containment
- Communication by Events and Delegates

Distributed Component Technologies

The goal:
- **Integration of services** for applications on various platforms
- **Interoperability**: let disparate systems communicate and share data seamlessly

Approaches:
- Microsoft: DDE, COM, OLE, OCX, DCOM and ActiveX
- Sun: JavaBeans, Enterprise JavaBeans, J2EE
- CORBA (Common Object Request Broker Architecture)
- Mozilla: XPCOM (Gecko functionality as components)
- SOAP (using XML)
The Microsoft Approach

• Continuous re-engineering of existing applications
• Component technology introduced gradually taking advantage of previous success, like
  – Visual Basic controls
  – Object linking and embedding (OLE)
  – Active X, ASP
• Solutions mainly adopted on MS platforms
• Review from older approaches to .NET + CLR
COM: Component Object Model

- Underlying most MS component technologies (before .NET)
- Made available on other platforms, but with little success
- COM does not prescribe language, structure or implementation of an application
- COM only specifies an object model and programming requirements that enable COM components to interact
- COM is a binary standard for interfaces
- Only requirement: code is generated in a language that can create structures of pointers and, either explicitly or implicitly, call functions through pointers.
- Immediate for some languages (C++, SmallTalk) but possible for many others (C, Java, VBscript,...)
COM interfaces and components

• **Invocation specification**: when an operation (method) of the interface is invoked, a pointer to the interface itself is passed as additional argument (like `self` or `this`)
  – The pointer can be used to access instance variables
• **COM component** may implement any number of interfaces.
• The entire implementation can be defined in a single class, but it does not have to be.
• A component can contain many objects of different classes that collectively provide the implementation of the interfaces provided by the component.

- A COM interface is a pointer to an interface node, which is a pointer to a table of function pointers (also called `vtable`)
- Note the double indirection
A COM component with 3 interfaces and 2 objects

- Object 1 implements Interfaces A and B,
- Object 2 implements Interface C
- Interfaces must be mutually reachable
- Possible according to COM specification, rare in practice

Figure 15.2 A COM object with multiple interfaces.
COM Interfaces

- Identity determined by **Globally unique identifiers (GUID)** (128 bits) or (non-unique) name
- **IUnknown**: root of interface hierarchy, includes:
  - QueryInterface
  - AddRef and Release (for Garbage Collection via Reference Counting)
- **QueryInterface** (GUID -> Interface reference/error) allows to know if an interface is implemented by the component
- “Invocations to **QueryInterface** argument **IUnknown** on the same component must return the same address”
- Thus **IUnknown** used to get the “identity” of a component

```c
[ uuid(00000000-0000-0000-C000-000000000046) ] interface IUnknown {
  HRESULT QueryInterface ([in] const IID iid, [out, iid_is(iid)] IUnknown *iid);
  unsigned long AddRef () ;
  unsigned long Release () ;
}
```
COM component reuse: Containement

- COM does not support for implementation inheritance
- Reuse supported through Containement & Aggregation
- **Containement**: an outer objects holds an exclusive reference to an inner object
- Requests to outer can be forwarded to inner, simply invoking one of its methods
COM component reuse: Aggregation

- Containment adds overhead for calling and returning from methods: could cause a performance issue
- With **aggregation**, a reference to the interface of Inner is passed to the client.
- **Outer** cannot intercept / modify / filter invocations to Inner
- Problem: The client should not be aware of the fact that **Inner** is serving instead of **Outer** (transparency)

- This can be achieved (only) with collaboration of the Inner object: calls to **QueryInterface** are forwarded to the **IUnknown** interface of Outer

![Diagram of COM component reuse: Aggregation](image)
COM inheritance, polymorphism versioning

• Single inheritance among interface possible but rarely used (eg \texttt{IUnknown}, \texttt{IDispatch} and few others)

• But due to the \texttt{QueryInterface} mechanism impossible to know if an interface has more methods

• Polymorphism given by support to \textit{sets of interfaces} for components:
  – The type of a component is the set of GUID of its interfaces
  – A subtype is a superset of interfaces

• COM does not support interface versioning
Creating COM objects

• An application can request a COM component at runtime, based on its class
• Class identifiers are GUIDs (called CLSIDs)
• Procedural static API for creating objects:
  – CoCreateInstance(CLASID, IID)
• Exploits a registry to identify a (local or remote) COM server which provides a Factories for COM Interfaces

Figure 15.9 COM server with two coclasses, each with a factory.
The next 8 slides (till .Net Framework excluded) can be skipped
Example from Microsoft environment (80’s)

- Excel-generated pie chart embedded in a Word document displayed in a PowerPoint presentation
- Different applications need to share data or procedures
DDE (Dynamic Data Exchange)

• A little history: starting with evolution of Microsoft approach:
  – Windows gave PCs a more accessible computing environment
  – Problem: lack of consistency between different programs
  – What if spreadsheet and word processor need to share data?
• Early solution was integrating suites into large programs:
  – e.g., Microsoft Works – Pros and cons of suite approach?

• Microsoft comes out with Dynamic Data Exchange (DDE), circa 1989
  – Lets different Windows programs share data through links
  – Suppose some spreadsheet data were linked into word processor
  – When you changed data in spreadsheet, the new data would appear in word processor
  – Limitation: you couldn’t update the data in the word processor; you had to invoke the spreadsheet to update the date there
  – Worse, links were fragile and would break if you moved data files around in file system
OLE (circa 1991)

- **Object Linking and Embedding**
  - Linking is essentially DDE, using reference semantics
  - Embedding lets users copy a snapshot of data into word processor and save it there
  - Linking is cheaper when data files are large
  - Embedding supports **compound documents** (“document-centric” computing)

- A way for Windows to create documents containing objects from other programs.
  - E.g. place a chart from Excel and a slide from PowerPoint into a Word document
  - Components containers can be re-used by many applications
  - But components do not make data independent of application programs, and OLE is a platform-specific solution.
OLE Technology (circa 1993)

• A set of APIs to create and display a (compound) document
  – Now possible to share code as well as data

• Component Object Model (COM)
  – COM protocols let components connect to origination program:
    – E.g. word processor can tell spreadsheet, “the user just clicked on the spreadsheet, so start yourself up, look for data here, and let me know when you’re done.”

• COM now includes OLE as part of a larger concept
  – OLE becomes a set of standard COM interfaces

• Embedded documents retain all their original properties
  – If the user decides to edit the embedded data, Windows activates the originating application and loads the embedded document
OLE Extensions (OCX)

• With Windows 95 came a new standard:
  – **OCX** (OLE Custom eXtension component)
  – A piece of code, smaller than application program, but with its own user interface
  – Let users bundle OCX controls to form customized applications
  – E.g., combine spell checker and synonym provider component to make a new program
  – *Is this beginning to sound like object-oriented programming?*
ActiveX (circa 1996)

• Microsoft retools OLE and COM as ActiveX
  – ActiveX applies to a whole set of COM-based technologies

• ActiveX control is Microsoft's answer to the Java technology from Sun
  – An ActiveX control is roughly equivalent to a Java applet, but is known as an ActiveX control

• Writing a program to run in the ActiveX environment creates a self-sufficient program that can run anywhere in ActiveX network

• This component is known as an ActiveX control, and is often used to attach a program to a web page
ActiveX - implementation

• An ActiveX control can be created using one of several languages or development tools, including C++ and Visual Basic, or with scripting tools such as VBScript.

• Network OLE for rudimentary support of distributed applications

• ActiveX controls originally were Windows only
  – Other vendors later provided Mac and Unix/Linux support for ActiveX

• Security issues: ActiveX controls have full file access (no sandbox)
  – Can be signed for authentication
THE .NET FRAMEWORK
Summary

• The **.NET framework** and **.NET components**
• Types of .NET components, connections of components, and deployments
• Local and distributed components
• **Aggregation** and **containment** compositions
• **Synchronous** and **asynchronous** method invocations
• **Delegates** and Event-based communication
The .NET Framework

- Introduced by Microsoft in 2000.
- Platform for rapid and easier building, deploying, and running secured .NET software components.
- Support for rapid development of XML web services and applications.
- Highly productive, component-based, multi-language environment for integrating existing applications with internet.
- Emphasis on interoperability.
The .NET Framework consists of:

- **The Common Language Specification (CLS)**
  It contains guidelines, that language should follow so that they can communicate with other .NET languages. It is also responsible for Type matching.

- **The Framework Base Class Libraries (BCL)**
  A consistent, object-oriented library of prepackaged functionalities and Applications.

- **The Common Language Runtime (CLR)**
  A language-neutral development & execution environment that provides common runtime for application execution.
.NET Framework structure
(http://www.dotnet101.com/articles/art014_dotnet.asp)
Common Language Specification

CLS performs the following functions:

• Establishes a framework that helps enable cross-language integration, type safety, and high performance code execution

• Provides an object-oriented model that supports the complete implementation of many programming languages

• Defines rules that languages must follow, which helps ensure that objects written in different languages can interact with each other
.NET Framework Base Class Library

• The Class Library is a comprehensive, object-oriented collection of reusable types

• These class library can be used to develop applications that include:
  – Traditional command-line applications
  – Graphical user interface (GUI) applications
  – Applications based on the latest innovations provided by ASP.NET
    • Web Forms
    • XML Web services
Common Language Runtime (CLR)

• CLR ensures:
  – A common runtime environment for all .NET languages
  – Uses Common Type System (strict-type & code-verification)
  – Memory allocation and garbage collection
  – Intermediate Language (MSIL) to native code compiler.
  – Security and interoperability of the code with other languages

• Over 36 languages supported today
  – C#, VB, Jscript, Visual C++ from Microsoft
  – Perl, Python, Smalltalk, Cobol, Haskell, Mercury, Eiffel, Oberon, Oz, Pascal, APL, CAML, Scheme, etc.