301AA - Advanced Programming [AP-2017]

Lecturer: Andrea Corradini  
Tutor: Lillo Galletta  

Department of Computer Science, Pisa 
Academic Year 2017/18

**AP-2017-08:** 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
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
Communication Protocol Models

- Remote Procedure Call (RPC)
  - Since 1980s, pioneered by Sun
  - Testing with various communication models
- Distributed extension of MS COM (DCOM)
  - Lets COM talk to other platforms
  - Complex configuration and complicated security model
- Remote Method Invocation (RMI)
  - Communication between methods of Java classes
- Drawbacks of RPC/RMI approach?
  - Platform-specific, procedural and low-level
.NET Components

• 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
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.
- Includes a virtual machine that provides a new platform for software development.
Overview of .NET Framework (cont.)

• Includes support for XML and Simple Object Access Protocol (SOAP) to deploy web services
• Goal: support desktop, window, and web-based application services on both Windows platform on other platforms through SOAP and HTTP
• .NET simplifies and improves support for components development and deployment w.r.t. Component Object Model (COM), and Distributed COM (DCOM) technology.
• COM components can be reused. COM does not support versioning (side by side execution), which may cause version conflict (DLL Hell problem). .NET technology supports component versions, and different versions can coexist without any conflict.
Overview of .NET Framework (cont.)

• Support of distributed components by Remoting Channel technology.
• Interoperability between COM, .NET and XML web service components.
• .NET web service is a replacement of MS DCOM technology for Internet applications supported by XML, SOAP and HTTP protocols. It overcomes DCOM’s lack of support for firewall.
• The .NET framework is available in .NET Framework SDK and Visual Studio.NET IDE SDK which support writing, building, testing, and deploying of .NET applications. It supports all .NET languages such as VB.NET, VC.NET, C#, and others.
• The .NET framework consists of two main parts: Common Language Runtime (CLR) and a set of unified framework basic class libraries including ASP.NET web forms for building web applications, Windows Forms for building desktop applications, and ADO.NET for data access.
Microsoft CLI (Common Language Infrastructure): historical notes

• When Java became popular Microsoft joined the initiative
• The idea was to exploit the dynamic load features of JVM to implement a component based architecture like COM
• There were two main problems:
  – Interoperability with the existing code (COM)
  – Support for many programming languages
• They extended the JVM but Sun complained of license infringement
• Microsoft started developing its own technology
• This was based on their experience on Java, but they tried to address the two problems above
• The result was the Common Language Infrastructure (CLI)
• The core of CLI is the Common Language Runtime (CLR) which plays the same role as the JMV in Java
Common Language Infrastructure
CLR and JVM

- Secure
- Portable
- Automatic MM (GC)
- Type safety
- Dynamic loading
- Class Library
- OOP

- Mix-in inheritance

Note that the essential traits of the execution environment are similar, though there are relevant differences in the design.

CLI has been standardized (ECMA and ISO) and is a superset of Java. We will refer mainly to CLR.
A new layer to the onion

Runtime exposes a superset of OS Services through the BCL

Runtime mediates access between the application and OS

Applications are group of types interacting together

Different runtimes implements in a different way LP abstractions such types: interoperability is complex
How CLR works

- C#
- C++
- ML
- VB

Unmanaged
- x86

Managed
- CIL

- Loader
- JIT
- Managed x86

- GC
- Security
- BCL
- CLR

...
Type system

- Execution environments such as CLR and JVM are *data oriented*
- A type is the unit of code managed by the runtime: loading, code, state and permissions are defined in terms of types
- Applications are set of types that interact together
- One type exposes a static method (Main) which is the entry point of the application: it loads the needed types and creates the appropriate instances
Java type system

Base types

Object

interface T

int

T[]

Class

String

class T
Java type system

• There are base types: numbers, Object, String and Class (which is the entry-point for reflection)
• Type constructors are:
  – Array
  – Class
• The number types are unrelated to Object with respect to inheritance relation
• This applies to interfaces too, but objects that implements interfaces are always inherited from object
• Java type system is far simpler than the one of CLR
CLR type system

Object

interface \( T \)

Type

Array \( T[] \)

String

ValueType

Delegate

class \( T \)

Delegate \( T \)

Struct \( T \)

int

Enum

Enum \( T \)

Base types
CLR Type System

- Common rooted: even numbers inherits from Object
- There are more type constructors:
  - **Enum**: constants
  - **Struct**: like class but without inheritance and stack allocation
  - **Delegate**: type that describes a set of methods with common signature
- Value types (numbers and structs) inherits from object. Still are not references and aren’t stored on the heap
- The trick is that when a value type should be upcasted to Object it is *boxed* in a wrapper on the heap
- The opposite operation is called *unboxing*
Foundation of .NET framework – CLR

- **Common Language Runtime (CLR)** is a virtual machine environment sitting on the top of Windows operating system.

- CLR consists of **Common Type System (CTS)**, **Just-In-Time CIL Compiler (JIT)**, **Virtual Execution System**, plus other management services (**garbage collection**, **security management**).

- CLR is like **JVM** in Java. It is assembled in a package of assembly consisting of **MS Intermediate Language (MSIL)** code and **manifest** (Metadata about this packet).

- The **CIL** code is translated into native code by JIT compiler in CLR. IL code **is verified by CTS** first to check the validity of data type used in the code.
Foundation of .NET framework – CLR

- **Multilanguage support**: (VB, managed C++, C# etc) by Common Language CLR implementation.
- A class in one language can inherit properties and methods from related classes in other Languages.
- The CTS defines a standard set of data type and rules for creating new types.
  - Reference types
  - Value types
- The code targeting CLR and to be executed by CLR is called **.NET managed code**. All MS language compilers generate managed codes that conform to the CTS.
Foundation of .NET framework – CLR

• The **CIL** code is like Java byte code. Regardless of the source programming languages, IL codes can interact by supports of the CLR.
• The IL code can be in the format of **executable (.EXE)** or **Dynamic Link Library (.DLL)**.
• If these IL codes are generated by .NET compiler, they are called **managed code**.
• The managed code can be executed only on **.NET aware platform**. Some DLL or EXE generated by non .NET compilers (such as early version of VC++) are called **un-managed code**.
Foundation of .NET framework – CLR

VB .Net

C# .Net

C++ .Net

VB .Net Compiler

C# .Net Compiler

C++ Compiler

IL code

Deployment

Assembly in .DLL or .EXE

Class Loader and Type Verifier

JIT

Managed native code

CLR Execution Unit

CLR

Class Library
CLR

• The CLR is a high performance execution engine. It provides a code-execution environment which manages the code targeting the .NET framework.

• The code management includes management of memory, thread, security, code verification and IL compilation.
The .NET Framework Class Library

- The .NET framework class library is a collection of reusable basic classes which are well organized by namespaces. The framework class library collects all classes including Windows Foundation Classes (WFC) into a single unified set of class.
- Correspond to Java API and packages
- A namespace consists of many classes and sub-namespaces. It is deployed as a component class library itself and is organized in a component–based hierarchy.
- The .NET framework itself is built up in a component model.
The .NET Framework Class Library
The .NET Framework Class Library

• Developers can create **custom namespaces**
• A namespace can be deployed as an **assembly** of binary components.
• `using <namespace>` in C# or `import <namespace>` in VB to access classes in a namespace.
The Component Model of .NET

- CIL DLL components are replacing COM Components
- MSIL Remoting Channels EXE components are replacing DCOM Component
- Web Service components are new SOAP components intended to be cross-platform, and cross-language web based components.
- The .NET component technology is unified-language oriented. Any .NET component is in the format of pre-compiled MSIL, which can be binary plugged in by any other MSIL components or any other .NET compatible clients.
The Component Model of .NET

• A .NET component is a single **pre-compiled** and **self described CIL module** built from one or more classes or multiple modules deployed in a **DLL assembly file**.

• An assembly consists of up to four parts:

  1) **Manifest** (table of info records): name of assembly, key info version, strong name, culture info, files that make up assembly, reference depended assemblies exported info.
  2) **Metadata** of modules
  3) **CIL code** of module
  4) Resources such as image files.
The Component Model of .NET

• A module has CIL code and its metadata but without manifest. Not loadable dynamically. Building block at compile time to build up an assembly Module file. Extension: .netmodule.

• An Assembly is made up by one or many classes in a module. Assembly has a manifest file to self-describe the component itself.

• An assembly has a file extension .dll or .exe and is dynamically loadable. A .dll file is not executable just like a class file is a byte code file that is not executable.
The Component Model of .NET

• Different types of components.
  – Visual – like Visual Basic controls which can be deployed in a toolbox as an icon for “drag and drop” in a window form container
  – Non-visual – known as .NET component
The Component Model of .NET

• A .NET component can be
  – **Local** (.dll), can only be accessed locally (within same application domain), in same machine
  – **Remote (distributed)** (.exe), can be accessed remotely in same machine or different machines.

• A .NET DLL component can be deployed
  – as a **private component**, which knows the target client
  – as a **shared public component**, which does not know the target client.

• A DLL component can be plugged-in to Windows form, Web form of another DLL or application.
Using Lib.dll

NameSpace Lib

test.dll or.exe

Lib.dll

(a) At compile time process

Application Domain

Test.dll
or
Test.exe

Lib.dll

(b) At run time process
The Connection Model of .NET

- **.NET component compositions**
  - Component compositions enable the component reuse in either **aggregation compositions** or **containment compositions**.

- **Containment compositions**: if a request to the outer component needs helps from an inner component the request is forwarded to that inner component.

- The outer component does not expose the interface of the inner component.

- The client is blind of the handler of the request. The `outerM2()` delegates a request to the `innerM()` method of inner component.
.NET aggregation compositions

- In **aggregation composition** the service of **inner component** hands out its service directly to the client of **outer component**.
  - The outer component exposes the interfaces of inner component.
  - The **innerM()** method of inner component becomes part of interface to the outer component.
- A .NET component can also be composed by mixed aggregations and containments in a flat structure or nested compositions in multiple levels in depth.
.NET containment and aggregation composition
Delegates

• The .NET Delegate is a method type (a reference to a method) which is similar to function pointer in C++, but it is type-safe and secure.

• A delegate will delegate a control flow to its registered event handler when the event is raised. Example of design pattern observer (like event listener in Java).

• An instance of a Delegate can hold a static method of a class or a method of a component or a method of an object itself.

• Two types of Delegates: SingleCast or MultiCast.
A **SingleCast** Delegate can only delegate one method at a time

```csharp
Delegate int MyDelegate();

public class MyClass
{
    public int ObjMethod() { - - - }
    static public int StaticMethod() { - - - }
}

public class Drive
{
    Static public void Main()
    {
        MyClass c = new MyClass();
        int x;
        MyDelegate dlg = new MyDelegate(c.objMethod());
        x = dlg();
        dlg = new MyDelegate(MyClass.StaticMethod());
        x = dlg();
    }
}
```
SingleCast Delegate

• As seen in this example, MyDelegate is a Delegate which references any method with int return type and without any parameter.
• The signatures of objMethod and StaticMethod match the Delegate MyDelegate.
• The first dlg() invokes objMethod and the second dlg() invokes the class method StaticMethod.
MultiCast Delegate

• A **MultiCast Delegate** has a void return type and can bind multiple methods and it will invoke them in the order of registrations. Example:

```csharp
Delegate void MultiDelegate();
MultiDelegate mdlg = null;
mdlg += new MultiDelegate (Method1);
mdlg += new MultiDelegate (Method2);
```

• Registration is done by `+= Delegate` operation and unregistration is done by `-= operation`.
• The Delegate plays a role of listener, and event handler must register this listener to be able to handle the event once the event is fired.
The communication of Components by Event and Delegate

- An **event** is a message sent by an object (the **event source**) to invoke an action. The object that intercepts the event and handles it is the **event target**.

- **Event-driven communication model** between components or within same component.

- The **Delegate** class is the communication channel class between event source and event target.

- Events can be predefined such as events triggered by a Windows Form component. A developer can also define a custom event.
The communication of Components by Event and Delegate

1. Create a delegate class.
   
   ```
   Public delegate void DelegateStart();
   ```

2. Create a class containing a delegate field, Class MyClass
   
   ```
   { public event DelegateStart EventStart;
   ```

3. Define an event handler
   
   ```
   public void handleEvent(){ - - -}
   ```

4. Bind delegate event with event handler via event listener, trigger an event, invoke the event handler.
   
   ```
   Public static void Main() { MyClass EventObj = new MyClass();
   EventObj.EventStart += new DelegateStart(handleEvent);
   EventObj.EventStart();
   ```

   ```
   ```
Remoting Connectors for .NET Distributed Components

• A component or a client cannot directly access a remote component running in different application domain in same or different processes unless using Remoting channel connection.
• The marshaling makes it possible to invoke a remote method of a distributed component.
• There are two ways to marshal an object: in **MBV (Marshal by Value)** server passes a copy of object to client or in **Marshal by Reference (MBR)** client creates a proxy of a remote object.
• When a remote component must run at a remote site, MBR is the only choice.
• Similar to **RMI in Java**
The Remoting asynchronous callback is based on **Remoting Delegate**. It will not block out the client while waiting for notification from remote components.

For example, someone wants to be notified once the stock prices reaches a specified level. Instead of pooling the stock price all the time, why not let the server notify you and you can do whatever you want to do.

In some other cases, the jobs on server will take very long to complete, why not let the server notify you when the job is done.

When client makes a synchronous call to remote method of remote component, it passes a **callback method** to server to be called back late through **Remoting Delegate**.
NET Component Deployments

• A .NET component can be deployed as **private component** or **public shared component** in an assembly file. The assembly is an atomic deployment (distribution) unit in .NET Framework.

• A private component knows the component where it will be plugged-in. A public shared component does not know which component will use itself.

• It must be published (registered) in a centralized repository **Global Assembly Cache (GAC)**. A shared component supports side-by-side multiple version component execution.
Private Deployment

• A Private component must be deployed in a same directory of its client or sub-directory of where the client is. It is the simplest deployment done by copying all components to where the client is.

• The undeployment is done by simply removing all related .dll components from the directory. A private component does not support versioning control and for in-house development within a company. The following shows an example of private deployment
Public shared deployment

• The most popular reusable component deployment is to deploy a component with a strong name, to register it with GAC.

• A shared component with strong name can make itself unique by public/private key pair. A shared component registered with GAC can make itself to be shared anywhere.