Introduction to .NET Framework

Compiled By:

Madan Kadariya

NCIT

.NET – What Is It?

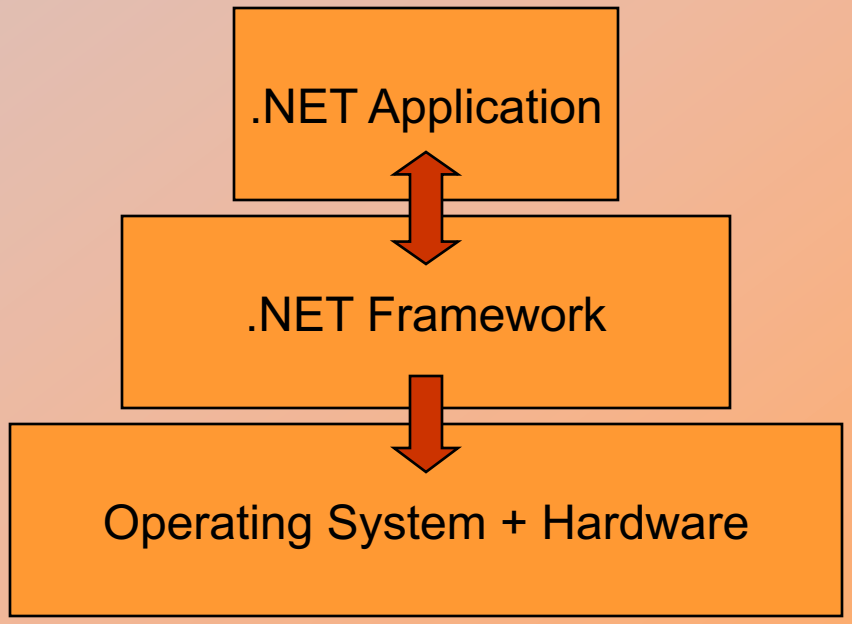
- Software platform
- Language neutral
- In other words:

  .NET is not a language (Runtime and a library for writing and executing written programs in any compliant language)

What Is .NET

- .Net is a new framework for developing web-based, network programming and windows-based applications within the Microsoft environment.
- The framework offers a fundamental shift in Microsoft strategy: it moves application development from client-centric to server-centric.

.NET – What Is It?



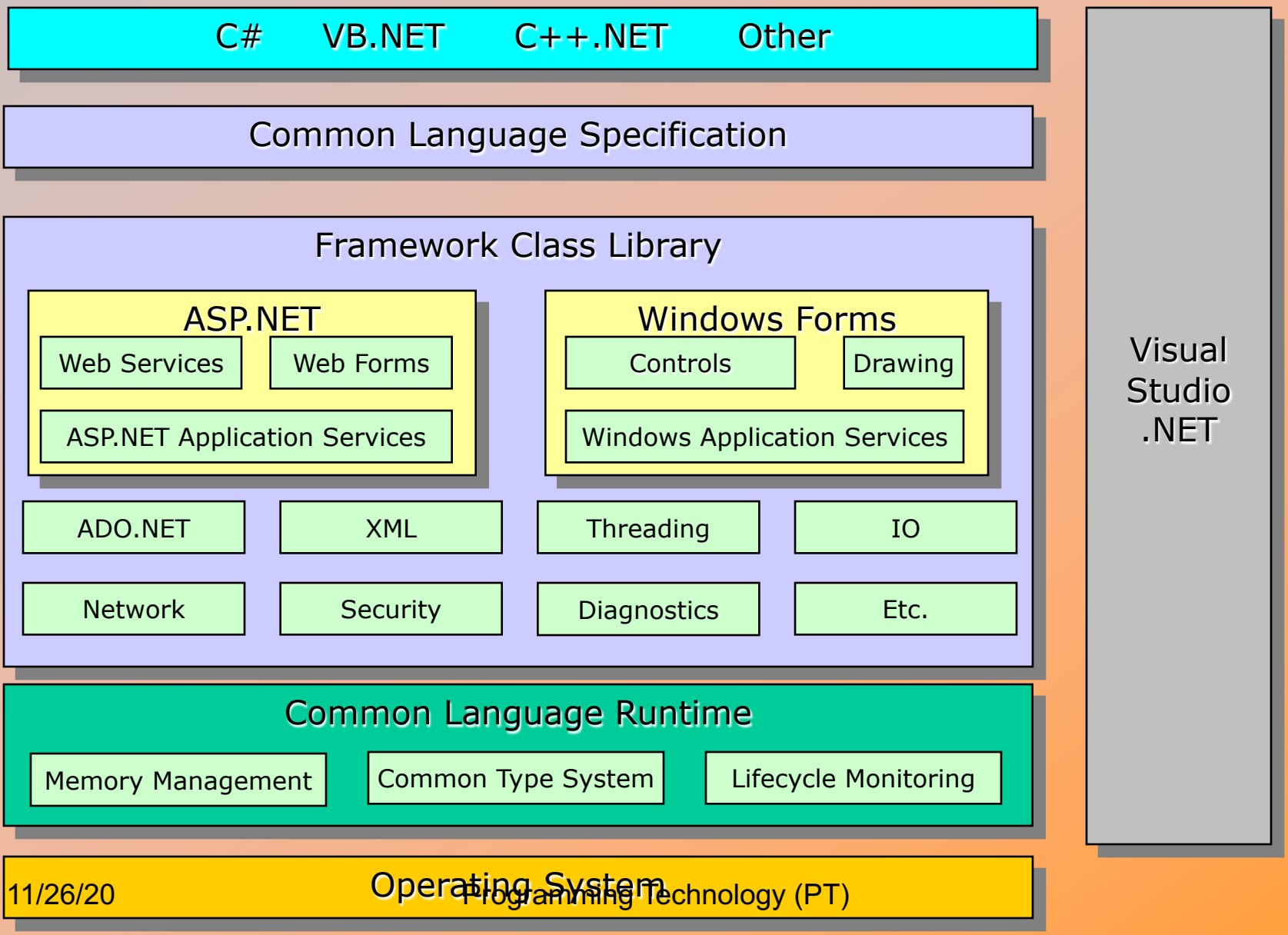
History

- Development began in 1998
- Beta 1 released Oct, 2000
- Beta 2 released July, 2001
- Finalized in Dec, shipping in Feb 2002
- Vista ships with .NET Framework 3.0 (Runtime).
- Latest Framework: .NET Framework 4.7.1

.NET Overview

- Three main elements:
  - The Framework (CLR, FCL, ASP, WinForms)
  - The Products (Windows, Visual Studio, Office)
  - The Services (My Services)
- Framework Goals
  - Improved reliability and integrated security.
  - Simplified development and deployment.
  - Unified API, multi-language support.
- XML is the .NET “Meta-Language”.
- All MS server products now .NET-enabled.

.NET Framework, Language and Tools



Common Language Runtime

- A runtime provides services to executing programs
  - Standard C library, MFC, VB Runtime, JVM
- CLR provided by .NET manages the execution of code and provides useful services
  - Memory management, type system, etc.
  - Services exposed through programming languages
    - C# exposes more features of the CLR than other languages (e.g. VB.NET

.NET Framework Class Library

- Framework – you can call it and it can call you
- Large class library
  - Over 2500 classes
  - Major components
    - Base Class: Networking, security, I/O, files, etc.
    - Data and XML Classes
    - Web Services/UI
    - Windows UI

Framework Libraries

- Web Services
  - Expose application functionalities across the Internet, in the same way as a class expose services to other classes.
  - Each Web service can function as an independent entity, and can cooperate with one another.
  - Data described by XML.
- ASP.NET
  - Replacement for the Active Server Technology.
  - Web Forms provide an easy way to write interactive Web applications, much in the same way as “normal” Windows applications.

Framework Libraries

- Provides facilities to generate Windows GUI-based client applications easily
- Form-oriented
- Standard GUI components
  - buttons, textboxes, menus, scrollbars, etc.
- Event-handling

Common Language Specification

- CLS is a set of rules that specifies features that all languages should support
  - Goal: have the .NET framework support multiple languages
  - CLS is an agreement among language designers and class library designers about the features and usage conventions that can be relied upon
    - Example: public names should not rely on case for uniqueness since some languages are not case sensitive
    - This does not mean all languages are not case sensitive above the CLR!

Some .NET Languages

- C#
- COBOL
- Eiffel
- Fortran
- Mercury
- Pascal
- Python
- SML

Perl
Smalltalk
VB.NET
VC++.NET
J#.NET
Scheme
....

More are planned or under development

Common Language Runtime (CLR)

- CLR works like a virtual machine in executing all languages.
- All .NET languages must obey the rules and standards imposed by CLR. Examples:
  - Object declaration, creation and use
  - Data types,language libraries
  - Error and exception handling
  - Interactive Development Environment (IDE)

Common Language Runtime

- Development
  - Mixed language applications
    - Common Language Specification (CLS)
    - Common Type System (CTS)
    - Standard class framework
    - Automatic memory management
  - Consistent error handling and safer execution
  - Potentially multi-platform
- Deployment
  - Removal of registration dependency
  - Safety – fewer versioning problems

Common Language Runtime

- CTS is a rich type system built into the CLR
  - Implements various types (int, double, etc)
  - And operations on those types
- CLS is a set of specifications that language and library designers need to follow
  - This will ensure interoperability between languages

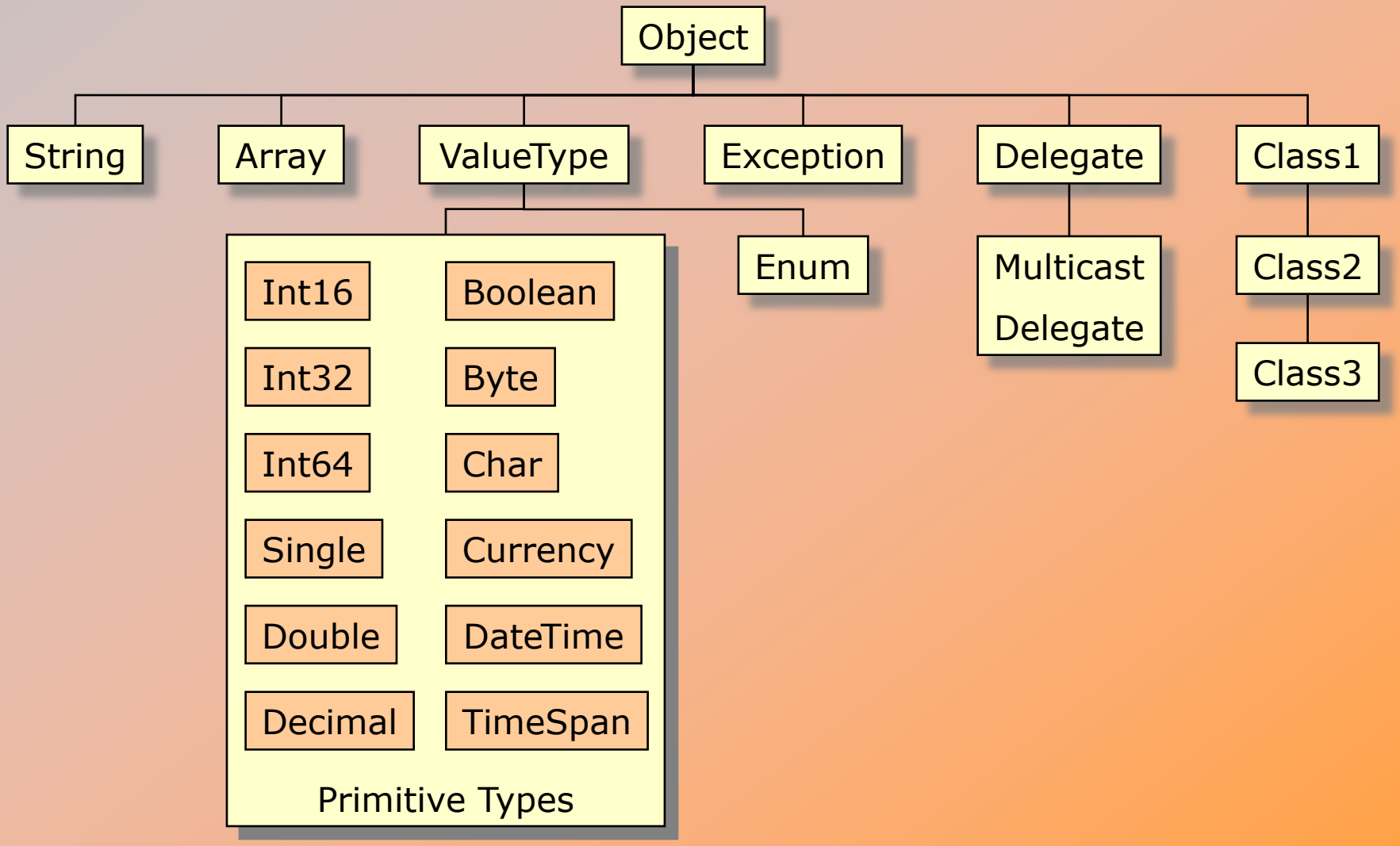
The Common Type System

- At the core of the Framework is a universal type system called the .NET Common Type System (CTS).
- Everything is an object - but efficient
  - Boxing and Unboxing
- All types fall into two categories - Value types and Reference types.
  - Value types contain actual data (cannot be null). Stored on the stack. Always initialized.
  - Three kinds of value types: Primitives, structures, and enumerations.
- Language compilers map keywords to the primitive types. For example, a C# “int” is mapped to System.Int32.

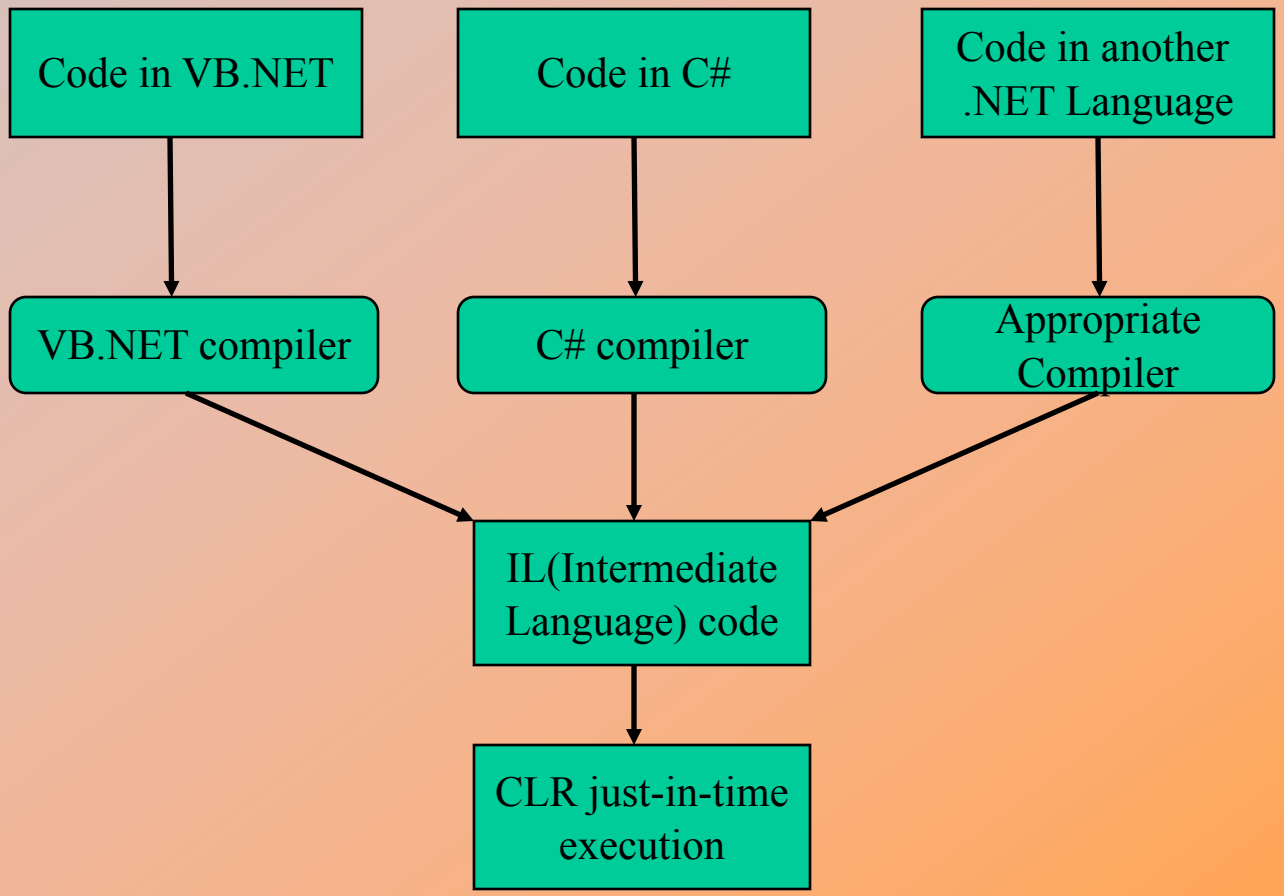
The Common Type System

- Reference types are type-safe object pointers. Allocated in the "managed heap"
- Four kinds of reference types: Classes, arrays, delegates, and interfaces.
  - When instances of value types go out of scope, they are instantly destroyed and memory is reclaimed.
  - When instances of reference types go out of scope, they are garbage collected.
- Boxing = converting an instance of a value type to a reference type. Usually done implicitly through parameter passing or variable assignments.
- UnBoxing = casting a reference type back into a value type variable.

The Common Type System



Compilation in .NET



Intermediate Language (IL)

- .NET languages are not compiled to machine code. They are compiled to an Intermediate Language (IL).
- CLR accepts the IL code and recompiles it to machine code. The recompilation is just-in-time (JIT) meaning it is done as soon as a function or subroutine is called.
- The JIT code stays in memory for subsequent calls. In cases where there is not enough memory it is discarded thus making JIT process interpretive.

Languages

- Languages provided by MS
  - VB, C++, C#, J#, JScript
- Third-parties are building
  - APL, COBOL, Pascal, Eiffel, Haskell, ML, Oberon, Perl, Python, Scheme, Smalltalk…

Packaging: Modules, Types, Assemblies, and the Manifest

- A “module” refers to a managed binary, such as an EXE or DLL.
- Modules contain definitions of managed types, such as classes, interfaces, structures, and enumerations.
- An assembly can be defined as one or more modules that make up a unit of functionality. Assemblies also can “contain” other files that make up an application, such as bitmaps and resource files.
- An assembly is the the fundamental unit of deployment, version control, activation scoping, and security permissions.

Packaging: Modules, Types, Assemblies, and the Manifest

- An assembly is a set of boundaries:
  - A security boundary - the unit to which permissions are requested and granted.
  - A type boundary - the scope of an assembly uniquely qualifies the types contained within.
  - A reference scope boundary - specifies the types that are exposed outside the assembly.
  - A version boundary - all types in an assembly are versioned together as a unit.
    - Avoid multiple version problem for DLL's

Packaging: Modules, Types, Assemblies, and the Manifest

- An assembly contains a "manifest", which is a catalog of component metadata containing:
  - Assembly name.
  - Version (major, minor, revision, build).
  - Assembly file list - all files "contained" in the assembly.
  - Type references - mapping the managed types included in the assembly with the files that contain them.
  - Scope - private or shared.
  - Referenced assemblies.
- In many cases, an assembly consists of a single EXE or DLL - containing the module's MSIL, the component metadata, and the assembly manifest. In other cases, the assembly may consist of many DLLs, with the manifest in its own file.
- No MSIL code can ever be executed unless there is a manifest associated with it.

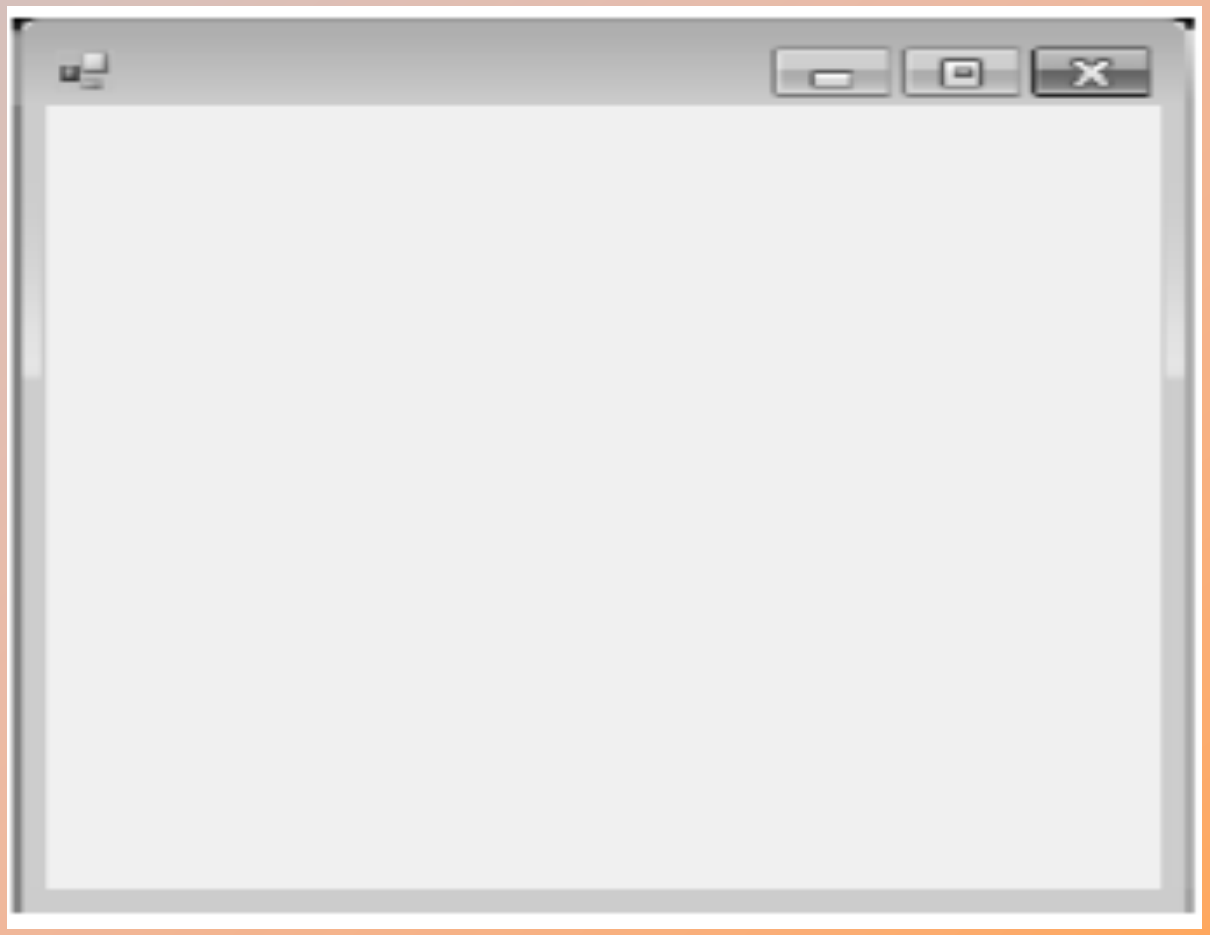
Windows Forms

- Framework for Building Rich Clients
  - RAD (Rapid Application Development)
  - Rich set of controls
  - Data aware
  - ActiveX® Support
  - Licensing
  - Accessibility
  - Printing support
  - Unicode support
  - UI inheritance

Windows Form Example:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
// The minimum required windows forms namespaces.
using System.Windows.Forms;
namespace SimpleWinFormsApp
{
// This is the application object.
class Program
{
static void Main(string[] args)
{
Application.Run(new MainWindow());
}
}
// This is the main window.
class MainWindow : Form {}
}
```

Output



Windows Presentation Foundation (WPF)

- WPF, or Windows Presentation Foundation, is a graphical system for rendering user interfaces.
- It provides great flexibility in how we can layout and interact with our applications.
- We can use C# or any other CLR language to communicate with user interface elements and develop application logic.
- The advantages of WPF for our application are its rich data binding and visualization support and its design flexibility and styling.
- WPF enables us to create an application that is more usable to our audience.

Windows Presentation Foundation (WPF)

- Suppose we want the user to select a car model for purchase.
- The standard way of displaying this choice is to display a drop-down list of car model names from which users can choose.
- There is a fundamental usability problem with this common solution:
  - Users are given only a single piece of information from which to base their decision- the text that is used to represent the item in the list.
  - For car fanatic this may not be an issue, but other users need more than just a model name to make an educated decision on the car they wish to purchase.
  - This is where WPF and its data visualization capabilities come into play.

XAML(Extensible Application Markup Language)

- XAML forms the foundation of WPF.
- XAML is similar to HTML in the sense that interface elements are defined using a tag-based syntax.
- XAML is XML-based and as such it must be well formed, meaning all opening tags require closing tags, and all elements and attributes contained in the document must validate strictly against the specified schemas.
- By default, when creating a WPF application in Visual Studio 2010, the following schemas are represented in generated XAML files:

http://schemas.microsoft.com/winfx/2006/xaml/presentation: This schema represents the default Windows Presentation Framework namespace.

XAML(Extensible Application Markup Language)

- *http://schemas.microsoft.com/winfx/2006/xaml*: This schema represents a set of classes that map to CLR objects.
- At runtime when a XAML element is processed, the default constructor for its underlying class is called, and the object is instantiated; its properties and events are set based on the attribute values specified in XAML.
- When creating a WPF application from visual studio 2010, an Application template creates two XAML files along with their respective code-behind files: ***App.xaml (App.xaml.cs)*** and ***MainWindow.xaml (MainWindow.xaml.cs).***

XAML(Extensible Application Markup Language)

- ***App.xaml*** represents the entry-point of the application.
- This is where application- wide (globally scoped) resources and the startup window are defined.
- Resources can be anything — data templates, arrays of strings, or brushes used to color the background of text boxes etc.

<Application x:Class="MyFirstWPFApplication.App"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
StartupUri="MainWindow.xaml">
<Application.Resources>
</Application.Resources>
</Application>

XAML

- A Form in WinForms is referred to as a Window in WPF.
- Anything placed on a WinForms Form is called a control, whereas items placed on a WPF Window are referred to as UIElements.
- Panels are WPF UIElements used for layout.
- A Control in WPF is a UIElement that can receive focus and respond to user events.
- A Content control in WPF can contain only a single item, which can in turn be other UIElements.
- The WPF Window class is a specialized Content control.

XAML

- Instead of depending on screen resolution, WPF measures UI Elements in Device Independent Units (DIUs) that are based on the system DPI.
- This enables a consistent look between many different hardware configurations.
- If controls could speak, the conversation might go something like this: **Layout Engine**: “Control, how much space would you like to have?”

<This is the Measure Stage>

Control: “I would like 50 DIUs for height, 100 DIUs for width, and a margin of 3 DIUs in the containing Grid cell.”

Layout Engine asks all other controls and layout containers.

Layout Engine: “Sorry, you can have only 40 DIUs for height, but I can grant the rest of your requests.”

<This is the Arrange Stage>

Arranging elements with layout panel

1. **Stack Panel:** Stack Panels place UIElements in -wait for it-stacks. Items are placed in either a vertical pile (the default), like a stack of DVDs, or a horizontal arrangement, like books on a shelf.
2. **Wrap Panel:** The Wrap Panel automatically wraps overflow content onto the next line(s).
3. **Dock Panel:** The Dock Panel uses attached properties to "dock" child UIElements.
4. **Canvas Panel:** Allows elements to be positioned absolutely using fixed coordinates. This layout container is the most similar to traditional Windows Forms, but it doesn't provide anchoring or docking features.
5. **Uniform Grid:** Places elements in an invisible table but forces all cells to have the same size.
6. **Grid :** Arranges elements in rows and columns according to an invisible table.

Windows Communication Foundation (WCF)

- Windows Communication Foundation is just that the foundation for communication between Windows computers.
- ASMX was really designed to make public services such as, for instance, adding an API to a simple Web application, WCF is a complete distributed computing platform for Windows.
- In the early days of .NET, there was a technology called .NET Remoting that replaced DCOM.
- DCOM was Distributed COM, or the common accepted way to communicate between distributed components.
- Remoting replaced it when .NET came out.

Windows Communication Foundation (WCF)

- WCF is a set of .NET technologies (Web services ,.Net Remoting and enterprises services) for building and running connected systems.
- WCF provides secure, reliable, and transacted messaging along with interoperability
- WCF applications can be developed in any language which can target the .NET runtime
- System.ServiceModel is the assembly that contains core functionality for WCF.
- A service is a Common Language Runtime (CLR) type that encapsulates business functionality and exposes a set of methods that can be accessed by remote clients.

When you should use it

- When your business logic has to interact with a variety of client applications.
- When client apps, which are going to use your service, may be written in Java or .Net.
- You are targeting a distributed computing architecture.

Advantages of WCF

Known Advantages:

- **Makes UI programming & distributed programming very easy.**

Reduce complexity by allowing us to focus on single programming model rather than learn multiple programming models.

- **Helps us talk to various applications written in various languages with ease which means more probable revenues as various applications can start using your core services with ease.**

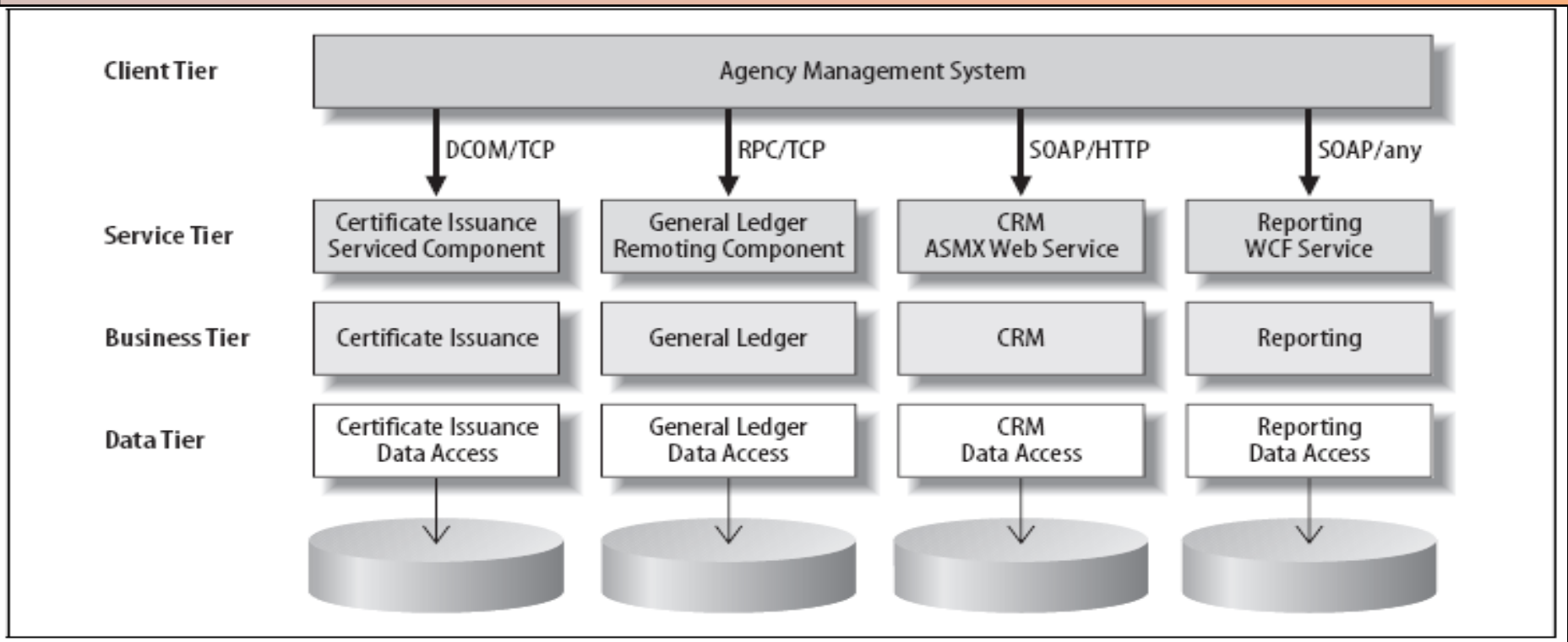
- **Helps us forget about interoperability between various underlying web service technologies in the past, present and future.**

- **With WCF, a single service can be defined and exposed over multiple endpoints to support multiple protocols at the same time.**

Discussion about SOA & WCF

1. What is SOA

SOA is the practice of sequestering the *core business functions* into independent services that don't change frequently. These services are glorified functions that are called by one or more presentation programs. The presentation programs are volatile bits of software that present data to, and accept data from, various users.
SOA is nothing more than separating changeable elements from unchangeable elements.



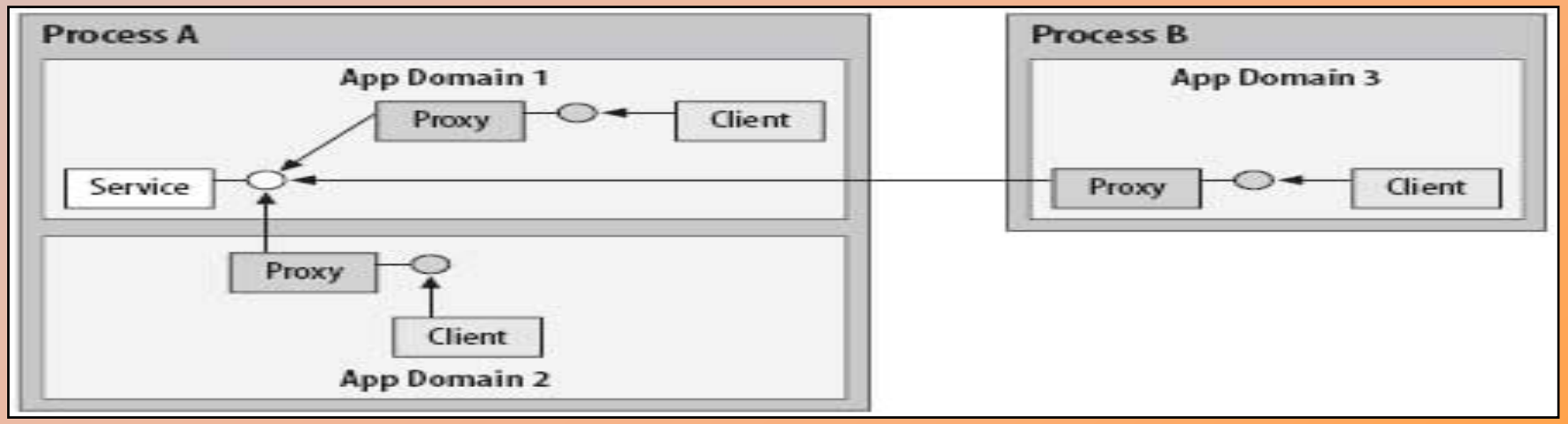
. Service boundaries implemented with different technologies

1. Services in WCF

a) Services' Execution Boundaries

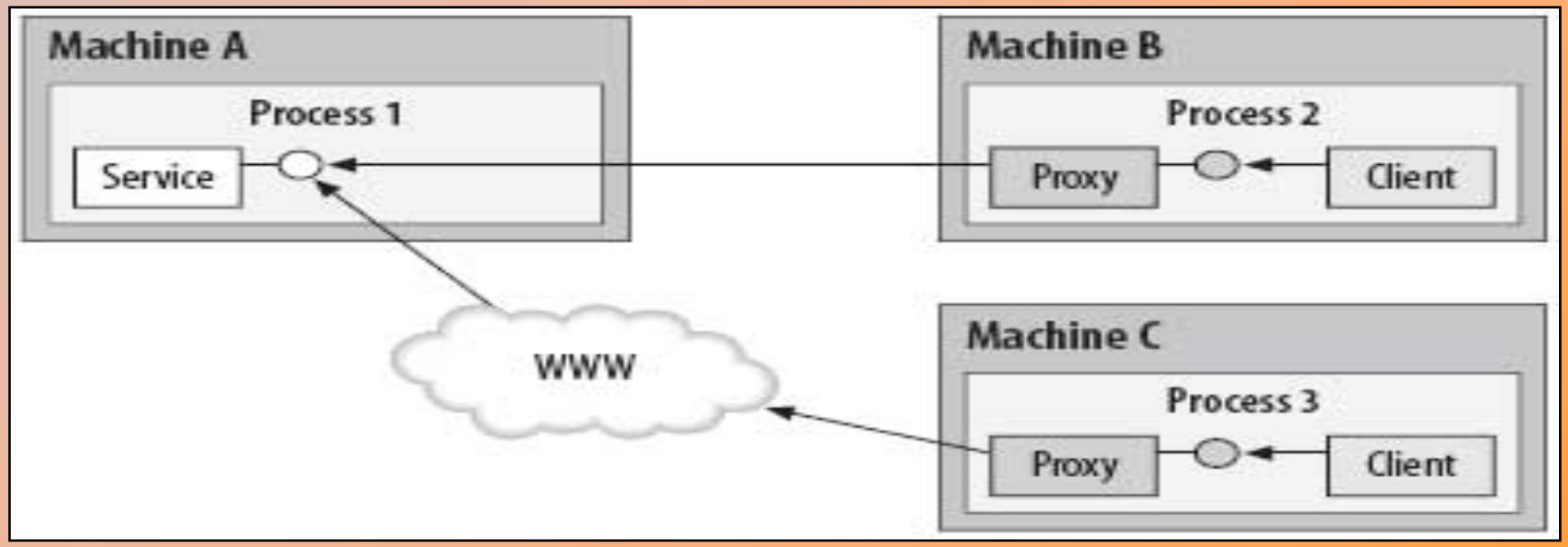
WCF allows the client to communicate with the service across all execution boundaries.

On the same machine , the client can consume services in the same app domain, across app domains in the same process, or across processes.



b) WCF & Location Transparency

WCF takes the remote programming model of instantiating and using a proxy and uses it even in the most local case. Because all interactions are done via a proxy, requiring the same configuration and hosting ,WCF maintains the same programming model for the local and remote cases; thus it not only enables you to switch locations without affecting the client, but also significantly simplifies the application programming model.



2. Addresses

Description:

In WCF, every service is associated with a unique address. The address provides two important elements: the location of the service and the transport protocol or transport schema used to communicate with the service. The location portion of the address indicates the name of the target machine, site, or network; a communication port, pipe, or queue; and an optional specific path or URI.

a) TCP Addresses

Ex. net.tcp://localhost:8002/MyService.

b) HTTP Addresses

Ex. http://localhost:8001

c) IPC Addresses

Ex. net.pipe://localhost/MyPipe

d) MSMQ Addresses

Ex. net.msmq://localhost/MyService

e) Peer Network Addresses

Peer network addresses use net.p2p for transport, to indicate the use of the Windows peer network transport. You must specify the peer network name as well as a unique path and port.

3. Contracts

Description:

In WCF, all services expose contracts. The contract is a platform-neutral and standard way of describing what the service does. WCF defines four types of contracts.

a) Service Contracts

Describe which operations the client can perform on the service.

b) Data Contracts

Define which data types are passed to and from the service. WCF defines implicit contracts for built-in types such as int and string, but you can easily define explicit opt-in data contracts for custom types

c) Fault Contracts

Define which errors are raised by the service, and how the service handles and propagates errors to its clients

d) Message Contracts

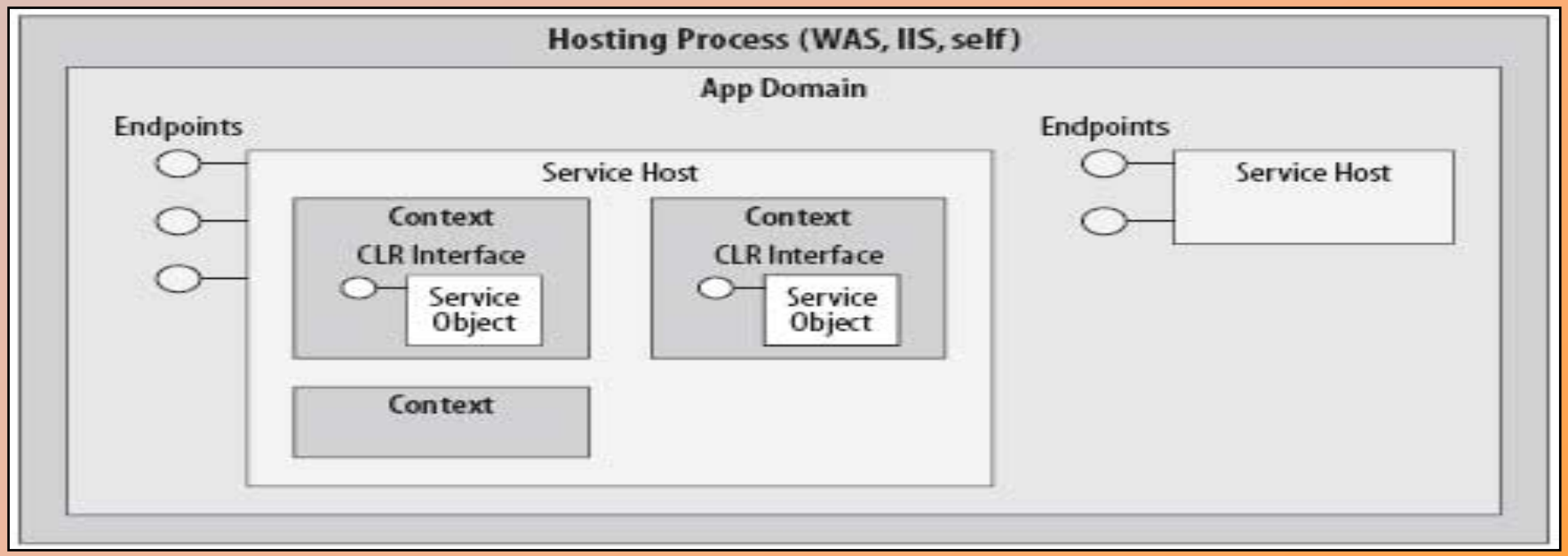
Allow the service to interact directly with messages.

4. Hosting

Description:

Every WCF service must be hosted in a Windows process called the host process. A single host process can host multiple services, and the same service type can be hosted in multiple host processes. The host can be provided by IIS, by the Widows Activation Service (WAS) on Windows Vista, or by the developer as part of the application.

So each .NET host process can have many app domains. Each app domain can have zero or more service host instances.



a) IIS Hosting

Hosting in IIS is very similar to hosting a classic ASMX web service. You need to create a virtual directory under IIS and supply a .svc file. The .svc file functions similar to an .asmx file, and is used to identify the service code behind the file and class.

b) Using Visual Studio

You can use Visual Studio to generate a boilerplate IIS-hosted service.

c) Web.Config file

The web site config file (Web.Config) must list the types you want to expose as services. You need to use fully qualified type names, including the assembly name.
<system.serviceModel> <services> <service name = "MyNamespace.MyService"> ... </service> </services> </system.serviceModel>

d) Self Hosting

Self-hosting is the name for the technique used when the developer is responsible for providing and managing the life cycle of the host process. Self-hosting is used both in the case of wanting a process (or machine) boundary between the client and the service, and when using the service in-proc—that is, in the same process as the client.

5. Bindings

A single service can support multiple bindings on separate addresses.

a) Standard Bindings

WCF defines nine standard bindings:

- **Basic Binding**
- **TCP Binding**
- **P2P Binding**
- **IPC Binding**
- **WS Binding**
- **Federated WS Binding**
- **Duplex WS Binding**
- **MSMQ Binding**
- **MSMQ Integration Binding**

A binding describes the protocols supported by a particular endpoint, specifically, the following:

- The transport protocol, which can be TCP, named pipes, HTTP, or MSMQ
- The message encoding format, which determines whether messages are serialized as binary or XML, for example
- Other messaging protocols related to security and reliability protocols, plus any other custom protocols that affect the contents of the serialized message

Name	Transport	Encoding	Interoperable
BasicHttpBinding	HTTP/HTTPS	Text, MTOM	Yes
NetTcpBinding	TCP	Binary	No
NetPeerTcpBinding	P2P	Binary	No
NetNamedPipeBinding	IPC	Binary	No
WSHttpBinding	HTTP/HTTPS	Text, MTOM	Yes
WSFederationHttpBinding	HTTP/HTTPS	Text, MTOM	Yes
WSDualHttpBinding	HTTP	Text, MTOM	Yes
NetMsmqBinding	MSMQ	Binary	No
MsmqIntegrationBinding	MSMQ	Binary	Yes

6. End Points

- **Every service is associated with an address that defines where the service is, a binding that defines how to communicate with the service, and a contract that defines what the service does. The endpoint is the fusion of the address, contract, and binding (ABC)**
  - Address **:- The address is obviously the location of the service, such as 'net.pipe://localhost/LocalTimeService'**
  - **B**inding**:- The binding specifies security options, encoding options, and transport options.**
  - Contract**:- the contract is the actual interface that the service implements.**
- **Every endpoint must have all three elements, and the host exposes the endpoint.**

Example :-

<endpoint name ="LocalTimeService“
address="net.pipe://localhost/LocalTimeService" binding="netNamedPipeBinding“
contract="ILocalTime" />

```
<system.serviceModel>
  <services>
    <service name = "MyService">
      <endpoint
        address = "net.tcp://localhost:8000/MyService/"
        bindingConfiguration = "TransactionalTCP"
        binding = "netTcpBinding"
        contract = "IMyContract"
      />
      <endpoint
        address = "net.tcp://localhost:8001/MyService/"
        bindingConfiguration = "TransactionalTCP"
        binding = "netTcpBinding"
        contract = "IMyOtherContract"
      />
    </service>
  </services>
  <bindings>
    <netTcpBinding>
      <binding name = "TransactionalTCP"
        transactionFlow = "true"
      />
    </netTcpBinding>
  </bindings>
</system.serviceModel>
```

Summary

- The .NET Framework
  - Dramatically simplifies development and deployment
  - Provides robust and secure execution environment
  - Supports multiple programming languages
- C# does not exist in isolation but has a close connection with the .NET framework
- .NET
  - CLR a relatively new, Java-like platform, but multi-language
  - Src→MSIL→JIT→Native Code
  - .NET framework includes many class libraries

Summary

- .NET Framework is a code execution platform – the environment which .NET programs run
- .NET Framework consists of two primary parts: Common Language Runtime and .NET Class Libraries
- The CLS (Common Language Specification) allows different languages to interact seamlessly.
- The CTS (Common Type System) allows all languages to share base data types.

Summary

- .NET languages are compiled to MSIL by their respective compilers
- MSIL code is compiled to machine code by the JIT compiler
- All .NET languages have equal access to the FCL (Framework Class Library) which is a rich set of classes for developing software
- Base Class Library is set of basic classes: Collections, I/O, Networking, Security, etc.
- ADO.NET provides .NET applications with access to relational databases

Summary

- .NET has great XML support including: DOM, XSLT, XPath, and XSchema
- Windows Forms provides GUI interface for the .NET applications
- ASP.NET allows creating web interface to .NET applications
- Web Services expose functionality from web sites and make it remotely accessible through standard XML-based protocols
- Visual Studio .NET is powerful development IDE for all .NET languages and technologies