CS355 Web Technologies

Dr. Ismail Hababeh

German-Jordanian University

Lecture 24

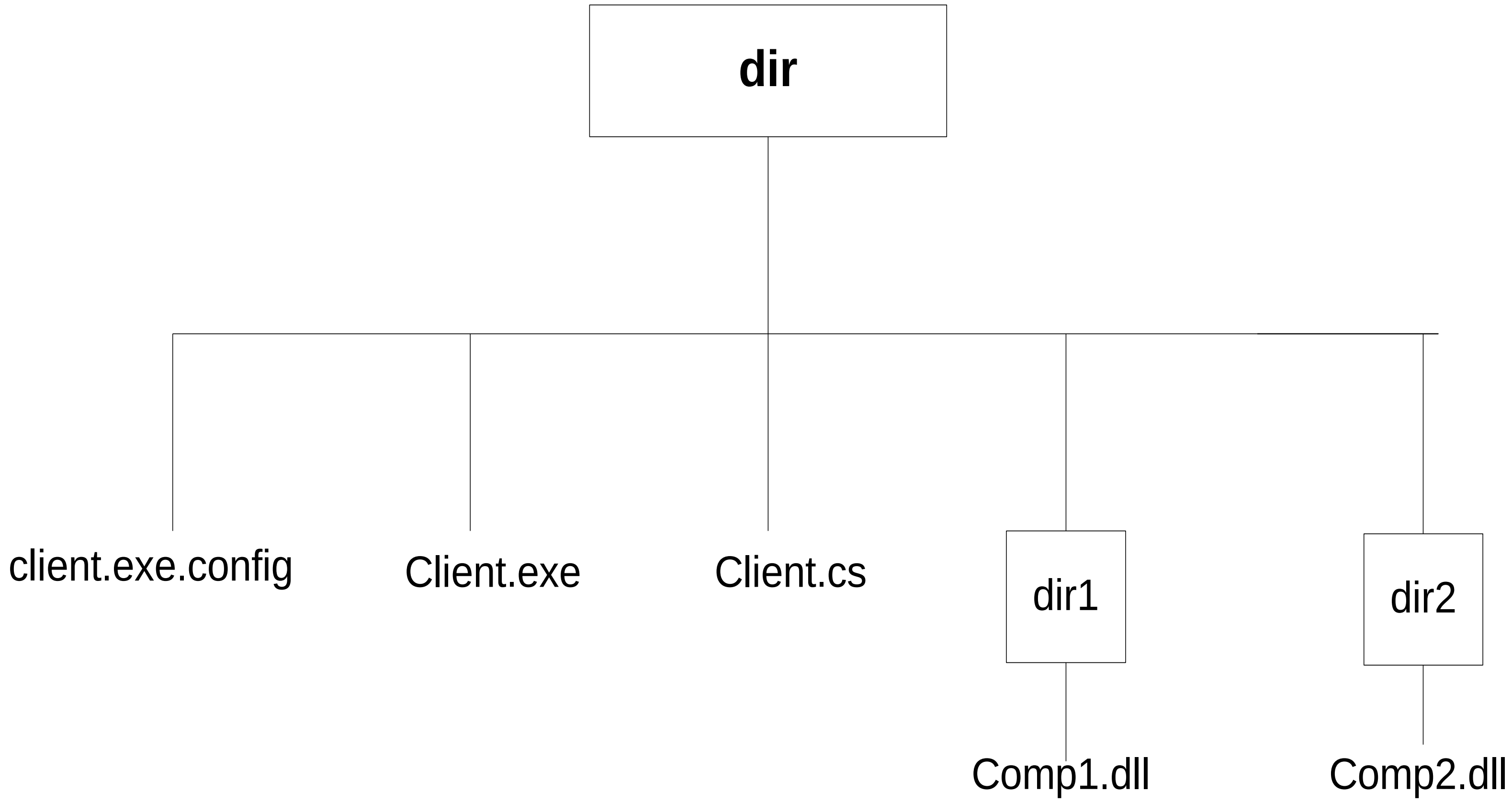
.NET Service Deployments

- A .NET service can be deployed as private or public shared in an assembly file.
  - Assembly is an atomic (basic) deployment (distribution) unit in .NET Framework.
- A private deployment knows where the service plugged-in.
- A public shared deployment doesn't know which service is used.
  - Supports multiple version service execution (versioning control)
  - Must be published (registered) in a centralized repository Global Assembly Cache (GAC).

Private Service Deployment

- A Private service must be deployed in the same directory or sub-directories of the client work. The simplest deployment done by copying all services to where the client is.
- The undeployment is done by removing all related .dll services from the client directory.
- A private service doesn't support versioning control (*records changes to a file or set of files over time so that you can recall specific versions later*).
- A private service is in-house development within a company.

Private Service Deployment - Example



Public Shared Service Deployment

- Deployed with a strong name (sn.exe)
- Registered with Global Assembly Cache GAC that can make it shared anywhere.
- Uniquely Identified by public/private key pair.

Creating .NET Public Shared Service

- Step 1: Create a public/private key pair by strong name sn.exe utility:

  > sn – k mykey.snk

  - The public key is for verification against private key which is signed as a cryptographic signature stored in the service assembly.
  - The public key is stored in a manifest of the assembly.
  - When a client references the service, the public key token is stored in the client's assembly.

Creating .NET Public Shared Service

Step2: Embed the following lines into the source code of the service

using System.Reflection; // view the attribute information.

[assembly:AssemblyKeyFile (“mykey.snk”)] // create a key pair (consisting of a public key and a private key).

[assembly:AssemblyDelaySign (false)] // a technique for signing assemblies outside of the build process.

[assembly://AssemblyVersion (“1,2,3,4”)]

// Version information for an assembly consists of the following four values:

// Major Version, Minor Version, Build Number, Revision

Creating .NET Public Shared Service

- Step3: Register the shared service in the Global Assembly Cache GAC

```
>gacutil /i:myservice.dll
```

- Step4: Using shared service
  - The client must make a reference to the shared service to be reused.

```
>csc /t:exe/r:myservice.dll/out:myapp.exe myapp.cs
```

Using Public Shared Service - Example

- In order to use the shared service, the client source code must “use namespaces” where namespace is available in the assembly.

Example: A public shared service deployment of the TempConv

```
using System;
using System.Reflection;
[assembly:AssemblyVersion("1.0.0.0")]
[assembly:AssemblyKeyFile("originator.key")]
namespace TempConv
{
public class TempConv
{
    public TempConv()
    {
    }
```

Using Public Shared Service - Example

```
public double cToF(double c)
            {
                        return (c*9/5.0+32);
            }
public double fToC(double f)
            {
                        return (f-32)*5/9.0;
            }
    }
}
```

Using Public Shared Service - Example

- The following command signs the signature with the service immediately without a delay.

```
>csc /t:library myservice.cs
```

- The next command line stores a public key token in the client service.

```
>sn –R myservice.dll mykey.sn
```

Using Public Shared Service - Example

- If the signature delay is needed, client must make a reference to the shared service to be reused. we can sign the signature late by

```
>csc  /r:myservice.dll /out:myapp.exe myapp.cs
```

- The signature is verified when the service is registered with GAC (in step #3) to ensure that the service is not altered since the assembly was built.

- At run time the public key token in the client manifest is verified against the public key which is part of service identity. If they match, then it indicates that this is the right service wanted.

Web Services Protocols

- Simple Object Access Protocol SOAP is an XML based message exchange protocol specified by the World Wide Web Consortium (W3C) specification.
- SOAP is also a lightweight protocol working in a distributed heterogeneous environment.
- SOAP = HTTP + XML since SOAP message is an XML-document follows a specific XML Schema.
- SOAP is used to specify the format of a request and response of the web service to get and send message via HTTP port by HTTP POST method.

Simple Object Access Protocol (SOAP)

- Every SOAP message has a required envelope and a message body.
  - SOAP envelope identifies an XML document as a SOAP message.
  - A SOAP envelope element must have one sub-element which is the body element.

Simple Object Access Protocol (SOAP)

- SOAP envelope may also have an optional sub element header. An element header gives more metadata about the message.

- A Body element of a SOAP message can be:
  - Data itself
  - Name of method be invoked
  - Arguments of the method be invoked
  - SOAP request message
  - Return results from a SOAP response message

Using SOAP - Example

- The SOAP request message asks a web service to convert a temperature zero Celsius degree to Fahrenheit. The name of the invoked method is “toFahrenheit”.

```
<?xml version="1.0" encoding="UTF-8"?>

<SOAP-ENV:Envelope

   SOAP-ENV:encodingStyle=

   "http://schemas.xmlsoap.org/soap/encoding/"

    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelop/”

   xmlns:xsd=http://www.w3.org/2001/XMLSchema //XML schema definition

 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:SOAP-

   ENC="http://schemas.xmlsoap.org/soap/encoding/">
```

xsi:type allows an XML instance to associate element type information directly rather than through an XSD

```
    <SOAP-ENV:Body>
        <toFahrenheit>
            <celsius xsi:type="xsd:string">0</celsius>
        </toFahrenheit>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

- This is the Body element of this SOAP request, an XSD string type, data “0” is passed in as an argument to the remote method “toFahrenheit” (the operation) of this web service.

- The SOAP response message of the converted temperature (0 Celsius) to Fahrenheit degree.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
    xmlns:SOAP-
    ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <SOAP-ENV:Body>
      <toFahrenheitResponse
         SOAP-ENV:encodingStyle=
          "http://schemas.xmlsoap.org/soap/encoding/">
      <toFahrenheitResult xsi:type=
         "xsd:string">32</toFahrenheitResult>
     </toFahrenheitResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```