Remoting & .NET

Dario Maggiorini
dario@dico.unimi.it

Outline

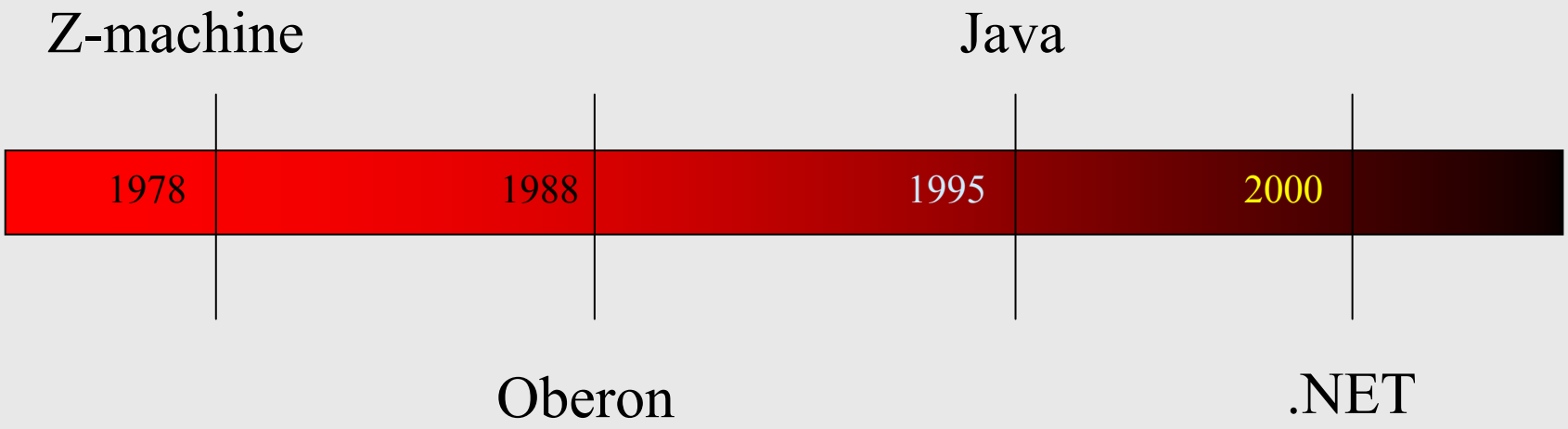
- .NET
  - C#
  - Common Language Runtime
- Remoting
  - AppDomain
  - Oggetti serializzabili
  - Proxy
  - Registrazione di un servizio
  - Utilizzo di un servizio
  - UDDI
    - Interrogazione
    - Creazione dinamica di un Proxy

.NET

.NET per sommi capi

- .NET è un runtime che prevede l'esecuzione di pseudocodice all'interno di una macchina virtuale
  - pseudocodice : MSIL
  - macchina virtuale : CLR
- .NET mette inoltre a disposizione del programmatore una libreria di classi su cui costruire i programmi

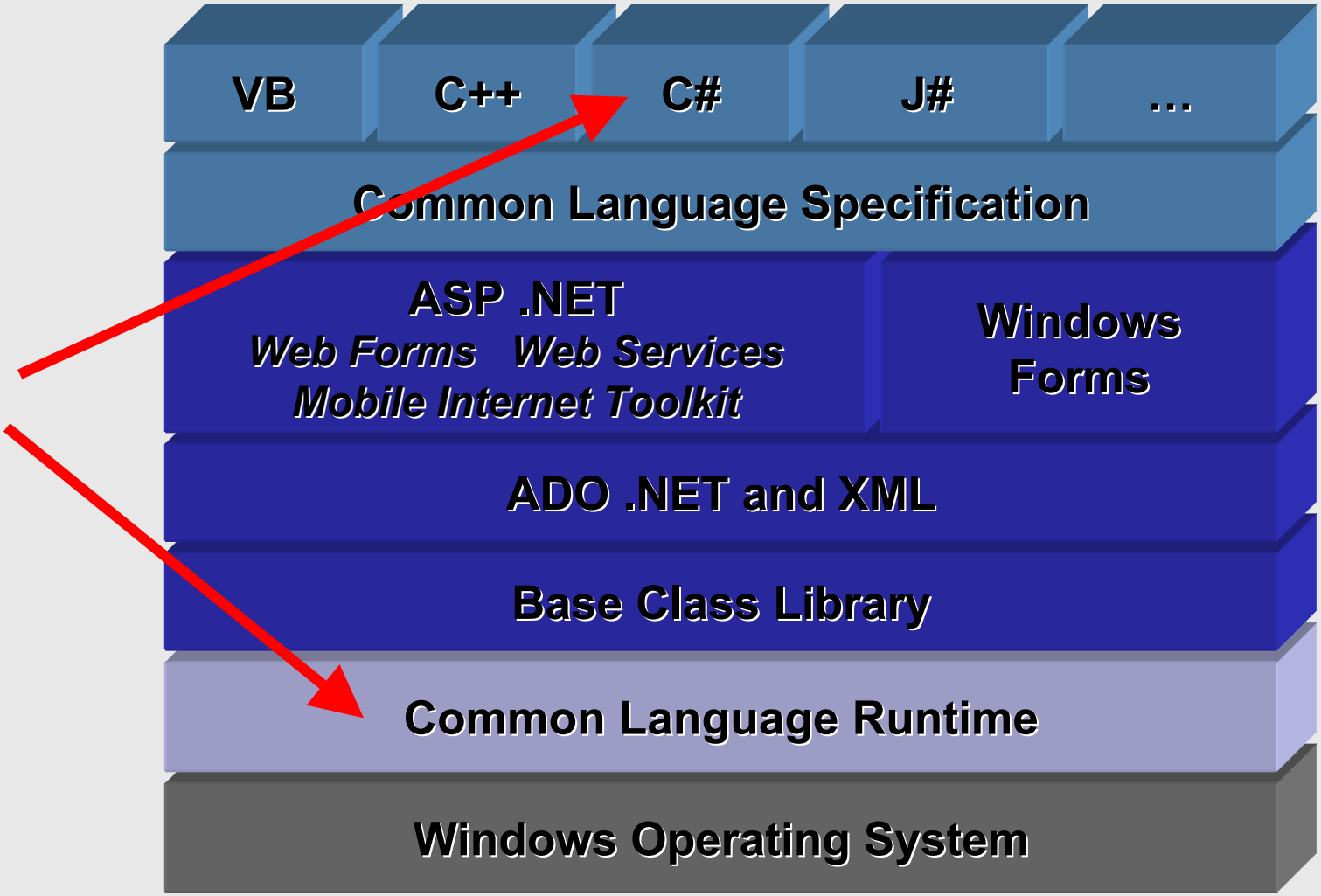
La riscoperta della ruota ?



.NET: caratteristiche

- Gli sviluppatori possono usare un qualsiasi linguaggio
- Il codice sorgente viene compilato in un linguaggio intermedio (MSIL) e posto all'interno di un "Assembly" (una DLL od un eseguibile)
- Negli assembly il codice è corredato di metadati, per poter descrivere l'assembly stesso e le classi in esso contenute (introspection)
- l'MSIL, quando eseguito, viene compilato "just in time" ed eseguito dalla macchina virtuale (Common Language Runtime o CLR)

Architettura del framework .NET



C#

yet another ...

- Sintassi simile al C
  - ha anche un pre-processore
- Gestione delle classi simile a java
  - e non solo quello ...
  - questa volta però lo “spazio delle classi” attraversa anche tutti gli altri linguaggi
- Molto più “programmer friendly” di altri linguaggi
  - Questo però dovrebbe far parte della naturale evoluzione degli ambienti di programmazione

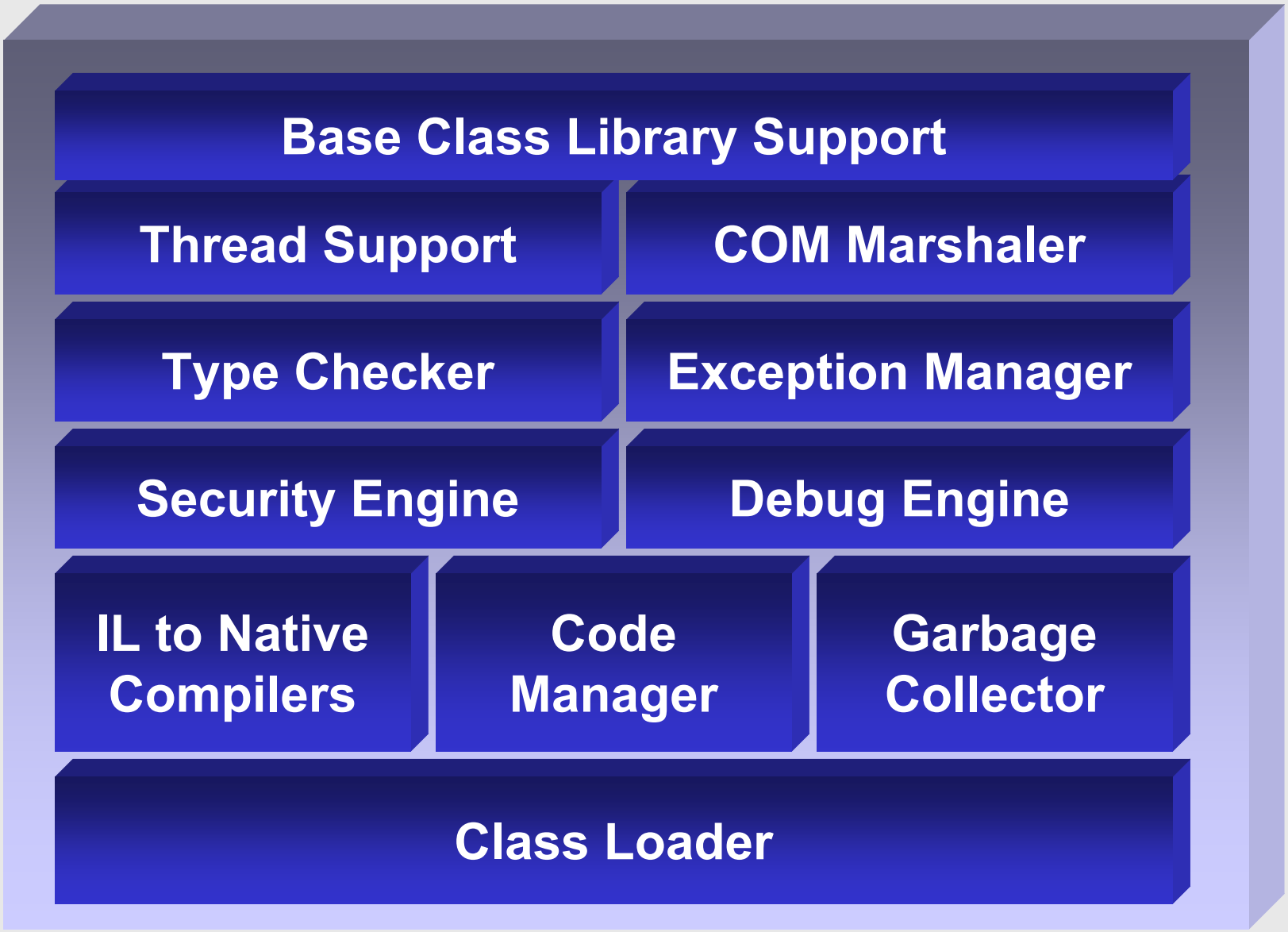
Le novità

- Finalmente, uno standard ECMA basato su una “invenzione” Microsoft
- Rispetto ad altri linguaggi di programmazione disponibili in ambiente windows è il primo totalmente “component oriented”
  - Proprietà delle classi
  - Metodi
  - Eventi
  - Attributi a Design-time e run-time

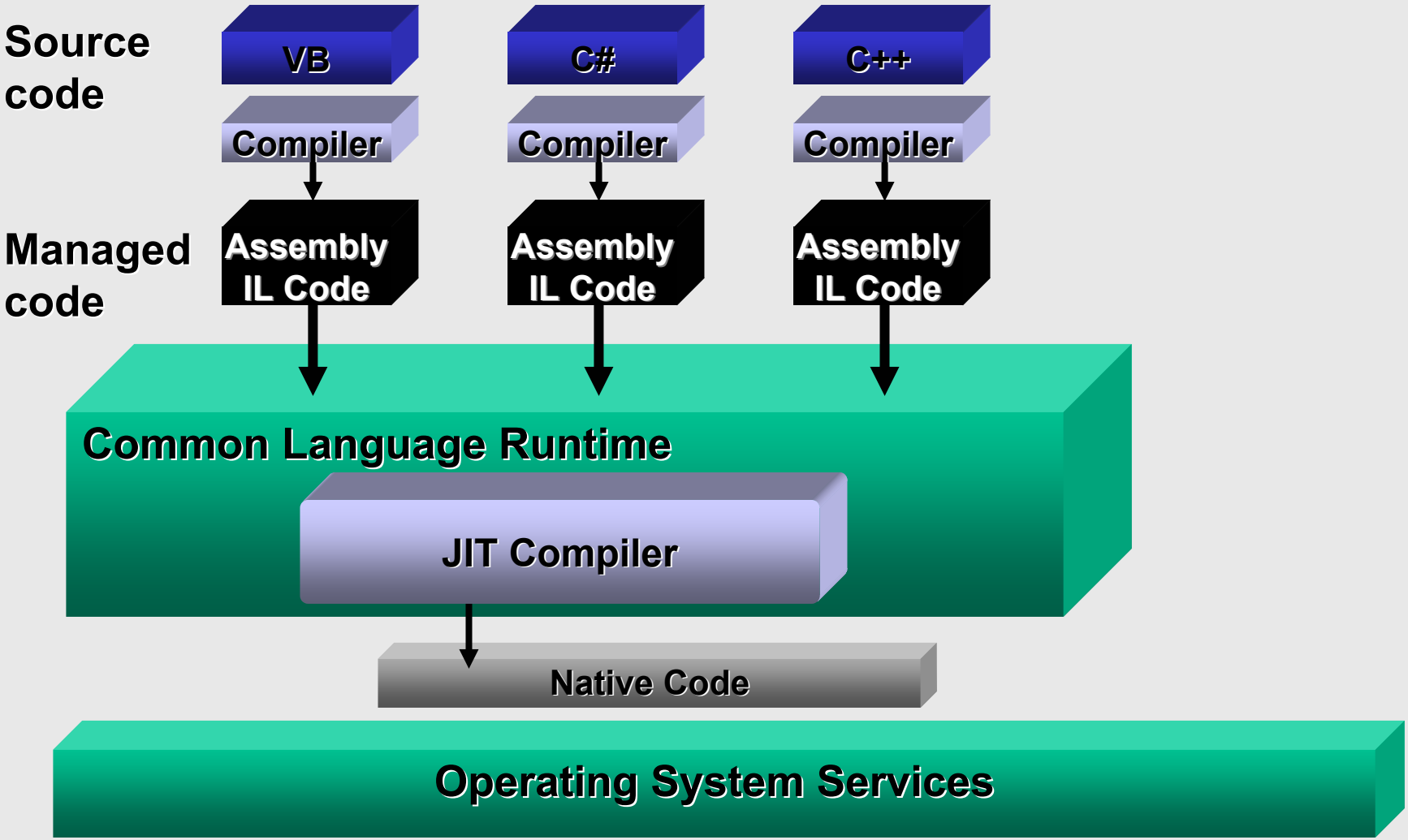
Common Language Runtime

CLR

- “Gestisce” il codice in esecuzione
  - Threading
  - Gestione della memoria
  - “*Auto-versioning, no more DLL Hell*”
- Gestione della sicurezza
  - Accesso ai singoli oggetti basato su regole
- Remotizzazione degli oggetti con SOAP


Base Class Library Support
Thread Support
COM Marshaler
Type Checker
Exception Manager
Security Engine
Debug Engine
IL to Native Compilers
Code Manager
Garbage Collector
Class Loader

.Net Execution Model



Remoting

AppDomain

Un application domain è un ambiente “isolato e protetto” dove una applicazione viene fatta eseguire

- L’isolamento di un AppDomain garantisce che:
  - L’applicazione può essere bloccata in modo indipendente dalle altre
  - Un crash di una applicazione non ha effetti su altre applicazioni
  - Un’ applicazione non possa accedere a risolse o codice in altri AppDomain

Oggetti .NET e AppDomain

- Gli oggetti .NET non possono “varcare la soglia” dell’AppDomain in cui sono stati creati
  - Non è possibile copiarli
  - Non esistono “puntatori” per referenziarli
- È però possibile richiedere che l’istanza di una classe sia accessibile dal di fuori dell’AppDomain in cui risiede
  - Che cosa daremo all’oggetto chiamante dall’esterno?
    - Una copia dell’oggetto?
    - Un riferimento (un proxy) all’oggetto originale?

Copia

Se l’oggetto può essere copiato, allora un AppDomain esterno è in grado di acquisirlo facendo un “marshal-by-value”

Il client chiamante riceverà un clone dell’oggetto originale

Questo vuol dire che da quel momento in poi i due oggetti saranno indipendenti (stati separati)

Copia

- Aggiungo l'attributo "`Serializable`"

```
[Serializable]
class myClass {
   // ...
}
```

- Implemento l'interfaccia `ISerializable`

```
class myClass : ISerializable {
   // ...
}
```

Proxy

È sufficiente far ereditare la nostra classe da `MarshalByRefObject` (MBRO)

```
[Serializable]
class myClass : MarhsalByRefObject {
  // ...
}
```

Oggetti in altri AddDomain esterni riceveranno dei proxy ad istanze di `myClass`

Come fanno dei client in altri AppDomain a specificare quale oggetto vogliono ?

Remoting

- Un server deve pubblicare un servizio (un oggetto)
  - Sono in ascolto su questa porta TCP / indirizzo HTTP
  - Ho un servizio chiamato “MathCalc”
  - Quando un client si collega a questo servizio deve essergli data una copia/un proxy di questo oggetto di tipo “Calc”
- Un client deve specificare a quale servizio intende accedere
  - Voglio collegarmi a “MathCalc” sull’host “10.0.0.1” sulla porta 80 usando il protocollo HTTP

Canali

- Un canale serve a trasportare messaggi da e verso oggetti remoti
  - Un server seleziona i canali sui quali accetta di ricevere richieste
  - Un client seleziona un canale tra quelli disponibili per comunicare con il server
- CLR ci fornisce già due canali
  - HTTP
  - TCP
- È possibile implementare canali personalizzati

Cos'è un server ?

- Si tratta di una normalissima applicazione (host) che
  - Carica le classi da un assembly
  - Configura l'infrastruttura di remoting per renderle disponibili
- È possibile scrivere degli host ad hoc
- È possibile usare direttamente IIS

Operazioni svolte da un server

- Decidere quali canali supportare e registrarli
- Decidere in che modo far soddisfare le richieste per una data classe (activation model)
- Registrare le classi

Registrazione di un canale

```
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Http;
using System.Runtime.Remoting.Channels.Tcp;

...

ChannelServices.RegisterChannel (new HttpChannel());
ChannelServices.RegisterChannel (new TcpChannel(4242));
```

Activation model

- Server based
  - WellKnownObjectMode.SingleCall
    - Ad ogni richiesta un nuovo oggetto viene istanziato, l'oggetto soddisferà solo quella particolare richiesta
  - WellKnownObjectMode.Singleton
    - Viene istanziato un oggetto che si occuperà di soddisfare tutte le richieste
- Client based
  - Client Activated Object (CAO)

Well-known objects type

- I tipi messi a disposizione dal server sono sempre noti a priori
  - lato server
    - ecco una classe
    - ecco come e quando istanziarla
    - ecco dove (indirizzo/end point) il client dovrà rivolgersi per contattare questa classe
  - lato client
    - voglio collegarmi a questo nodo di rete
    - prendo il tipo (noto) presente a questo end point

Registrazione di una classe

- Un server può registrare una classe facendo uso di

  `RegisterWellKnownServiceType`

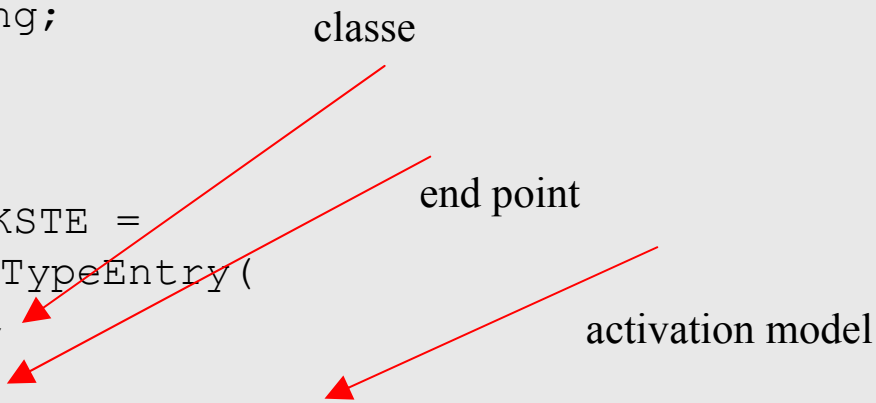
  specificando:
  - la classe
  - il nome dell'end point
  - l'activation model

```
using System.Runtime.Remoting;

...

WellKnownServiceTypeEntry WKSTE =
        new WellKnownServiceTypeEntry(
                typeof(Calc),
                "MathCalc",
                WellKnownObjectMode.SingleCall
        );

RemotingConfiguration.RegisterWellKnownServiceType(WKSTE);
```



```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Http;
using System.Runtime.Remoting.Channels.Tcp;

class RemotingHost {
   static void Main(string[] args) {
        RemotingConfiguration.ApplicationName = "MathService";
        WellKnownServiceTypeEntry WKSTE =
                new WellKnownServiceTypeEntry(
                        typeof(Calc),
                        "SharedMathCalc",
                        WellKnownObjectMode.Singleton
                )
        ;
    RemotingConfiguration.RegisterWellKnownServiceType(WKSTE);
    ChannelServices.RegisterChannel(new HttpChannel(9000));
    ChannelServices.RegisterChannel(new TcpChannel(4242));
  }
}
```

Well-known objects type

- I tipi messi a disposizione dal server sono sempre noti a priori
  - lato server
    - ecco una classe
    - ecco come e quando istanziarla
    - ecco dove (indirizzo/end point) il client dovrà rivolgersi per contattare questa classe
  - lato client
    - voglio collegarmi a questo nodo di rete
    - prendo il tipo (noto) presente a questo end point

Well known objects URL

- Gli oggetti registrati devono essere resi raggiungibili dai client
- La localizzazione di un “well known object” viene effettuata tramite un “well known object URL”

```
Protocol://ComputerName:Port/ApplicationName/ObjectUri
```

- Il nome dell’applicazione diventa una directory virtuale
- Il nome dell’oggetto dovrebbe finire con .rem o .soap
- Un canale TCP necessita sempre del numero di porta

configurazione esterna

- È possibile evitare tutte le operazioni viste prima e costruire un file di configurazione per il remoting

- Il formato del file è XML
- Il nome convenzionale è quello dell'applicazione con postfisso ".config"
- Il codice si semplifica notevolmente

MathService.exe.config

```
<configuration>
  <system.runtime.remoting>
    <application name="MathService">
      <service>
        <wellknown mode="SingleCall" type="Calc"
                   objectUri = “MathCalc" />
        <wellknown mode="Singleton" type="Calc"
                   objectUri = "SharedMathCalc" />
      </service>
      <channels>
        <channel port="9000" ref="http" />
        <channel port="4242" ref="tcp" />
      </channels>
    </application>
  </system.runtime.remoting>
</configuration>
```

```
using System;
using System.Reflection;
using System.Runtime.Remoting;

class MyHost {
  static void Main(string[] args) {
    String s = Assembly.GetExecutingAssembly().Location;
    RemotingConfiguration.Configure (s + ".config");
    Console.ReadLine();
  }
}
```

Client activated object

- È una modalità molto diversa da quella server side
- Il client può richiedere effettivamente l'istanziazione di un oggetto sul server (fornisce i parametri al costruttore)
- L'istanza persiste fintanto che:
  - Scatta un timeout
  - Il client distrugge il riferimento all'oggetto remoto

Registrazione

- Viene effettuata usando `RegisterActivatedServiceType`
- Ha bisogno di specificare solo la classe da registrare

```
using System.Runtime.Remoting;
...
ActivatedServiceTypeEntry ASTE =
        new ActivatedServiceTypeEntry(typeof(Calc));
RemotingConfiguration.RegisterActivatedServiceType(ASTE);
```

```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Http;
using System.Runtime.Remoting.Channels.Tcp;

class RemotingHost {
   static void Main(string[] args) {
        RemotingConfiguration.ApplicationName =
                                  "WiseOwlMathService";
        ActivatedServiceTypeEntry ASTE =
                 new ActivatedServiceTypeEntry(typeof(Calc));
        RemotingConfiguration.RegisterActivatedServiceType(ASTE);

        ChannelServices.RegisterChannel(new HttpChannel(9000));
        ChannelServices.RegisterChannel(new TcpChannel(4242));
        Console.ReadLine();
   }
}
```

MathService.exe.config

```
<configuration>
  <system.runtime.remoting>
    <application name="MathService">
      <service>
        <activated type=“Calc" />
      </service>
      <channels>
        <channel port="9000" ref="http" />
        <channel port="4242" ref="tcp" />
      </channels>
    </application>
  </system.runtime.remoting>
</configuration>
```

```
using System;
using System.Reflection;
using System.Runtime.Remoting;

class MyHost {
  static void Main(string[] args) {
    String s = Assembly.GetExecutingAssembly().Location;
    RemotingConfiguration.Configure (s + ".config");
    Console.ReadLine();
  }
}
```

Il codice NON CAMBIA

Operazioni svolte da un client

- Un client vuole usare un oggetto remoto
  - well known object
    - il client crea un proxy usando `Activator.GetObject`
  - Client activated object
    - il client chiede al server di istanziare un oggetto remoto usando `Activator.CreateInstance` e ne ottiene un proxy

Activator.GetObject

```
object o = Activator.GetObject ( GetType (Calc),
   "http://localhost:9000/MathService/SharedCalc")

Calc c = (Calc) o;

c.Add (42);
```

Activator.CreateInstance

- Richiede più lavoro da parte della rete
  - Il client manda il messaggio di attivazione al server
  - Il server crea l'oggetto usando il costruttore più opportuno
  - Il server incapsula l'oggetto in un ObjRef
  - Il server manda al client un messaggio con ObjRef all'interno
  - Il client crea un proxy per ObjRef

ObjRef

- Il CLR crea una istanza di un oggetto ObjRef ogni volta che ona classe viene registrata al fine di contenere tutte le informazioni utili all'infrastruttura di remoting
  - lo "strong name" della classe
  - la gerarchia di ereditarietà
  - la lista delle interfacce implementate
  - l'URI dell' end point
  - dettagli sui canali registrati

Activator.CreateInstance

```
Attribute activationAttributes =
   new UrlAttribute("http://localhost:9000/MathService");

object o = Activator.CreateInstance(
   GetType(Calc),
   null,
   activationAttributes
)

Calc c = (Calc) o;

c.Add (42);
```

MathClient.exe.config

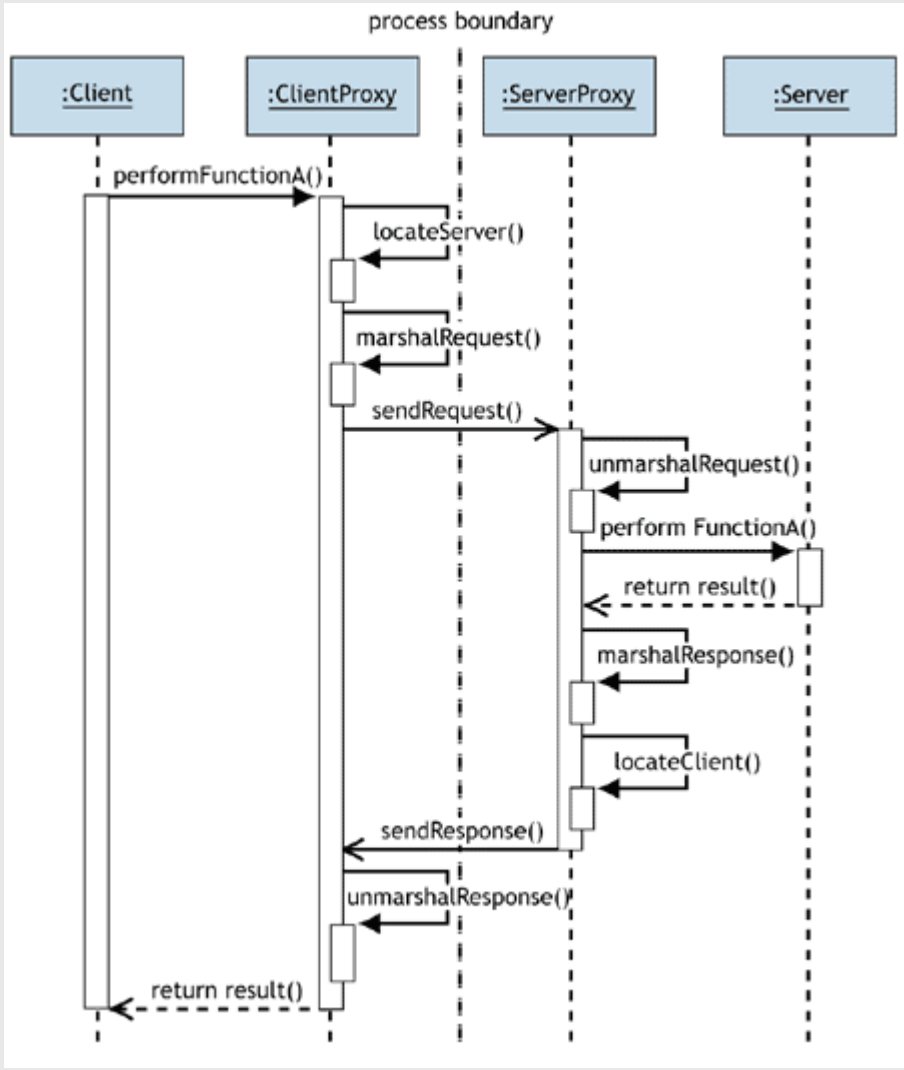
```
<configuration>
  <system.runtime.remoting>
    <application name="MathService">
      <client url = "http://localhost:9000/MathService"
                    displayName = "MathService">
        <activated type = "Calc"/>
        <wellknown type = "BadCalc"
       url="http://localhost:9000/MathService/BadCalc"/>
      </client>
    </application>
  </system.runtime.remoting>
</configuration>
```

Dietro le quinte ...

- Il client riceve un proxy quando attiva un oggetto remoto
  - Il client “pensa" che il proxy sia davvero l’oggetto
- Il Proxy è una istanza della classe `TransparentProxy`
  - Ricalca in tutto e per tutto l’oggetto remoto

- Una chiamata al proxy:
  - passa il controllo all’oggetto vero se siamo nello stesso AppDomain
  - altrimenti crea un messaggio (sotto forma di un oggetto che implementa `IMessage`) e lo passa ad una istanza di `RealProxy`
  - `RealProxy` manda il messaggio all’oggetto remoto

- `RealProxy` può essere estesa e customizzata
  - e.g. per dare load balancing
- `TransparentProxy` non può essere modificata


process boundary
:Client
:ClientProxy
:ServerProxy
:Server
performFunctionA()
locateServer()
marshalRequest()
sendRequest()
unmarshalRequest()
perform FunctionA()
return result()
marshalResponse()
locateClient()
sendResponse()
unmarshalResponse()
return result()

Well-known objects type

- I tipi messi a disposizione dal server sono sempre noti a priori
  - lato server
    - ecco una classe
    - ecco come e quando istanziarla
    - ecco dove (indirizzo/end point) il client dovrà rivolgersi per contattare questa classe
  - lato client
    - voglio collegarmi a questo nodo di rete
    - prendo il tipo (noto) presente a questo end point

UDDI registry

Universal Description Discovery and Integration

- È il corrispettivo dell'RMI registry per i web service
- Può essere interpellato per avere informazioni riguardo i servizi che vi sono registrati

Interrogazione UDDI

```
string uddihost = “http://test.uddi.microsoft.com/inquire”;

UddiConnection myConn = new UddiConnection(uddihost);
FindService fs = new FindService("*");
ServiceList servList = fs.Send(myConn);

foreach (ServiceInfo servInfo in servList.ServiceInfos) {
   // ...
}
```

Per Windows bisogna installare l’UDDI SDK

Very unknown service

- Se usiamo un UDDI registry può capitare di imbatterci in un servizio che vogliamo utilizzare ma che non sappiamo assolutamente come istanziare
  - il nostro sistema locale non sa istanziare un proxy
- Usiamo un proxy generico
- Generiamo il proxy a runtime basandoci sulla descrizione del servizio

Creazione dinamica di un Proxy

- L'interrogazione dell'UDDI registry ci ha restituito un oggetto di tipo `ServiceInfo`

Primo passo:

Da questo dobbiamo risalire all'entry point del servizio.

```
GetBindingDetail gbd = new GetBindingDetail();
gbd.BindingKeys.Add( servInfo[i].ServiceKey );
BindingDetail bd = gbd.Send(myConn);
BindingTemplate bt = bd.BindingTemplates[0];
string accessPoint = bt.AccessPoint.Text;
```

Creazione dinamica di un Proxy

- Ora, avendo l'access point possiamo andare ad interrogare la descrizione del servizio (al suo entry point) e creare dinamicamente il proxy da istanziare

WSDL

Web Services Description Language

WSDL è un formato XML usato per descrivere servizi. Nel caso dei web services abbiamo un sottoinsieme dei metadati presenti nella classe remota

```
ServiceDescription sd = ServiceDescription.Read(
                              new XmlTextReader(accessPoint)
                         );

ServiceDescriptionImporter sdi =
       new ServiceDescriptionImporter();
sdi.AddServiceDescription(sd, null, null);
sdi.ProtocolName = "Soap";
sdi.Style = ServiceDescriptionImportStyle.Client;

CodeNamespace codeNs = new CodeNamespace(“MyProxy");
sdi.Import(codeNs, null);

StringBuilder csCodeBuilder = new StringBuilder();
StringWriter sw = new StringWriter(csCodeBuilder);
```

```
CSharpCodeProvider csCodeProvider = new CSharpCodeProvider();
csCodeProvider.CreateGenerator().GenerateCodeFromNamespace(
                                                    codeNs,
                                                    sw,
                                                    null);
sw.Close();

String csCode = csCodeBuilder.ToString();
ICodeCompiler csComp = csCodeProvider.CreateCompiler();
CompilerParameters compParams = new CompilerParameters();
compParams.GenerateExecutable = false;
compParams.GenerateInMemory = true;
compParams.IncludeDebugInformation = false;
compParams.ReferencedAssemblies.Add("System.Dll");
compParams.ReferencedAssemblies.Add("System.Xml.Dll");
compParams.ReferencedAssemblies.Add("System.Web.Services.Dll");

CompilerResults compResult =
   csComp.CompileAssemblyFromSource(compParams, csCode);
```