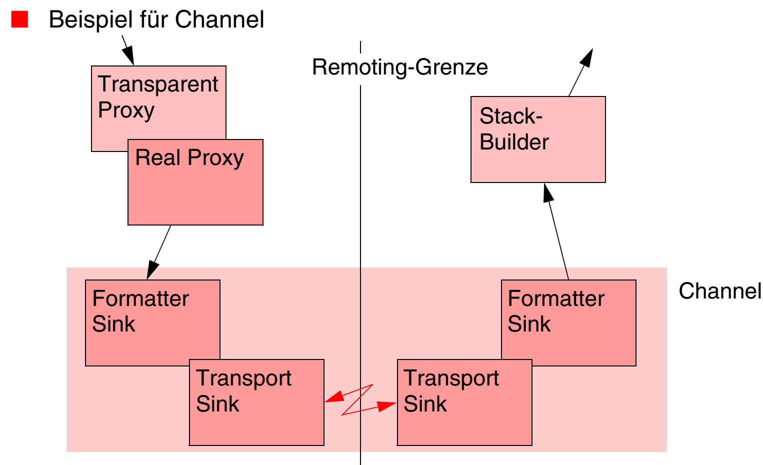
3.1 Architektur von .Net-Remoting (3)

- Beispiel für Channel



3.1 Architektur von .Net-Remoting (4)

- Einsatz verschiedener RPC-Protokolle
  - ◆ verschiedene Formatter-Sinks
    - SOAP für Web-Services
    - Formatter für .Net-Remoting-Protocol
    - (Formatter für GIOP)
- Einsatz verschiedener Transportprotokolle
  - ◆ verschiedene Transport-Sinks
    - TCP/IP für .Net-Remoting-Protocol (oder für IIOP)
    - HTTP für Web-Services
- Weitere Sinks in der Kette
  - ◆ z.B. für Transaktionen, Gruppenkommunikation etc.

3.2 Objektreferenz

- Darstellung von entfernten Objektreferenzen
  - ◆ „ObjRef“-Objekt
  - ◆ enthält
    - Typbezeichner für entferntes Objekt
    - Assembly-Informationen
    - Basisklassen und Schnittstellen
    - Objekt-URL zur Aktivierung
    - Channel-Informationen (welche Sinks müssen enthalten sein)
  - ◆ ObjRef-Objekt wird By-value übergeben
    - daraus kann erzeugt werden:
      – Proxies (per Reflection aus Typbezeichner)
      – Sink-Objekte aus Channel-Informationen und Kontext-Properties

4 Erzeugung entfernter Objekte

- Beispiel (in **C#**): Hello-Objekt

```
class Hello : MarshallByRefObject
{
    public string sayHello()
    {
        return "Hallo";
    }
}
```

- Zwei unterscheidbare Vorgehensweisen
  - ◆ Objekterzeugung durch Server: Server-activated objects
  - ◆ Objekterzeugung durch Client: Client-activated objects

4 Erzeugung entfernter Objekte (2)

- Server-activated objects (SAO)
  - Erzeugung der Objektinstanz (bei Bedarf) im Server
  - alternative Verfahren
    - Single-call object
    - Singleton object
    - Published object
- Client-activated objects (CAO)
  - Client erzeugt Objekte
  - alternative Verfahren
    - direkte Erzeugung
    - Erzeugung über Fabrik (Factory)

4.1 Single-Call Object

- Eigene Objektinstanz für jeden Aufruf
  - entspricht Stateless-Session-Bean
  - Objektinstanz ohne speicherbaren Zustand
- Server-Seite
  - Erzeugung des Channels (hier: HTTP/SOAP-Kanal)

```
ChannelServices.RegisterChannel( new HttpChannel(4711) );
```

    - alternativ: TCP-Channel mit binärem RPC-Protokoll
  - Registrierung des Objekttyps

```
RemotingConfiguration.RegisterWellKnownServiceType(
        typeof(Hello), "SayHello.soap",
        WellKnownObjectMode.SingleCall );
```


4.1 Single-Call Object (2)

- Client-Zugriff
  - Ermitteln der Referenz durch Aktivierung
    - Beispiel

```
sayer= (Hello) Activator.GetObject( typeof(Hello),
            "http://localhost:4711/SayHello.soap" );
```

    - Referenzerzeugung abhängig vom registrierten Objekttyp
- Lifecycle-Management
  - automatische Erzeugung einer Instanz für alle Aufrufe
  - nebenläufige Aufrufe möglich
    - Threading-Einstellung im jeweiligen Kontext

4.2 Singleton Object

- Genau eine Objektinstanz für alle Aufrufe
  - reiner Dienst
  - Zustand möglich
- Server-Seite
  - Erzeugung des Channels
  - Registrierung des Objekttyps

```
RemotingConfiguration.RegisterWellKnownServiceType(
        typeof(Hello), "SayHello.soap",
        WellKnownObjectMode.Singleton );
```

- Client-Zugriff
  - identisch zu Single-call object

4.3 Published Object

- Genaue eine Objektinstanz
  - reiner Dienst (wie Singleton Object)
  - aber: Instanz wird vorab erzeugt und registriert
- Server-Seite
  - Erzeugung des Channels
  - Erzeugung und Registrierung des Singleton

```
Hello sayer= new Hello();
RemotingServices.Marshal( sayer, "SayHello.soap" );
```

- Client-Zugriff
  - identisch zu Singleton object

Reproduktion oder Verwendung dieser Unterlage bedarf in jedem Fall der Zustimmung des Autors.

4.4 Direkte Objekterzeugung

- Client kann beliebige Objektinstanzen eines Typs erzeugen und nutzen
- Server-Seite
  - Erzeugung eines Channels
  - Registrierung eines Typs

```
RemotingConfiguration.ApplicationName= "MySayer";
RemotingConfiguration.RegisterActivatedServiceType(
        typeof(Hello) );
```

- Client-Zugriff
  - Erzeugung eines Channels
  - Registrierung eines Client-Typs

```
RemotingConfiguration.RegisterActivatedClientType(
    typeof(Hello), "http://localhost:4711/MySayer" );
...
M= new Hello(); // erzeugt entferntes Objekt
```

Reproduktion oder Verwendung dieser Unterlage bedarf in jedem Fall der Zustimmung des Autors.

4.5 Erzeugung über Fabrik

- Factory-Pattern (Entwurfsmuster der Fabrik)
  - Registrierung eines Singleton (die Fabrik)
  - Aufruf einer `create()`-Methode an der Fabrik
    - Erzeugung eines neuen Objekts im Server
    - Rückgabe der Referenz an Aufrufer
    - automatische Zuweisung einer Kommunikationsadresse durch .Net-Remoting

Reproduktion oder Verwendung dieser Unterlage bedarf in jedem Fall der Zustimmung des Autors.

4.6 Konfiguration

- Konfigurationsdatei statt explizitem Erzeugen und Registrieren
  - XML-Datei (vgl. Deployment-descriptor)
- Beispiel: Singleton object im Server

```
<configuration>
    <system.runtime.remoting>
        <application>
            <channels>
                <channel ref="http" port="4711"/>
            </channels>
            <service>
                <wellknown mode="Singleton"
                    type="Hello, Server"
                    objectURI="SayHello.soap"/>
            </service>
        </application>
    </system.runtime.remoting>
</configuration>
```

Reproduktion oder Verwendung dieser Unterlage bedarf in jedem Fall der Zustimmung des Autors.

4.6 Konfiguration (2)

- Laden der Konfiguration im Server
  - Angabe des Dateinamens

```
Remoting.Configuration.Configure( "HelloServer.exe.config" );
```

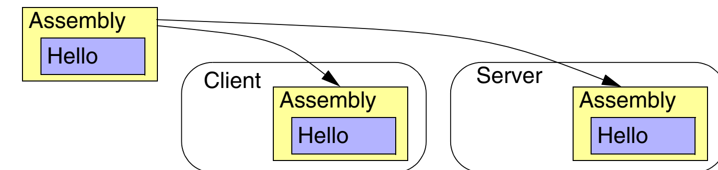
- Ähnliche Konfiguration im Client
- Weitere Konfigurationsmöglichkeiten
  - Wechsel des Channels (z.B. von HTTP auf TCP)
  - eigene Channel-Implementierungen
    - Angabe der Implementierungsklassen für Transport- und Formatter-Sinks
  - Angaben zum Lifecycle-Management

5 Assembly-Struktur

- Verteilter Objektzugriff erfordert:
  - gemeinsam genutzte Typinformationen in allen beteiligten Knoten

5.1 Gemeinsam genutzte Klasse

- Shared assembly für Objektklasse
  - über das Netz zugreifbar (Webserver, URL)
  - Objektklasse ist sowohl Client- als auch Server-Knoten bekannt

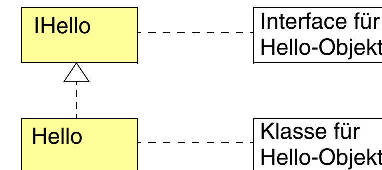


5.1 Gemeinsam genutzte Klasse

- ★ Vorteil
  - einfach
- ▲ Nachteile
  - Client kann Objekt auch direkt instanziieren
  - Benutzer auf Client-Computer können Code Disassemblieren

5.2 Gemeinsam genutztes Interface

- Objekt bekommt Interface für entfernten Zugriff (vgl. RMI)



  - Shared assembly für Interface, private Assembly für Klasse (nur Server)
- ★ Vorteil
  - Trennung möglich
- ▲ Nachteil
  - Referenz nicht weitergebbar an Dritte
    - Cast auf Interface lässt sich dort nicht durchführen (?)