.NET REMOTING

OVERVIEW OF MICROSOFTS .NET REMOTING TECHNOLOGY

Peter R. Egli
INDIGOO.COM

Contents

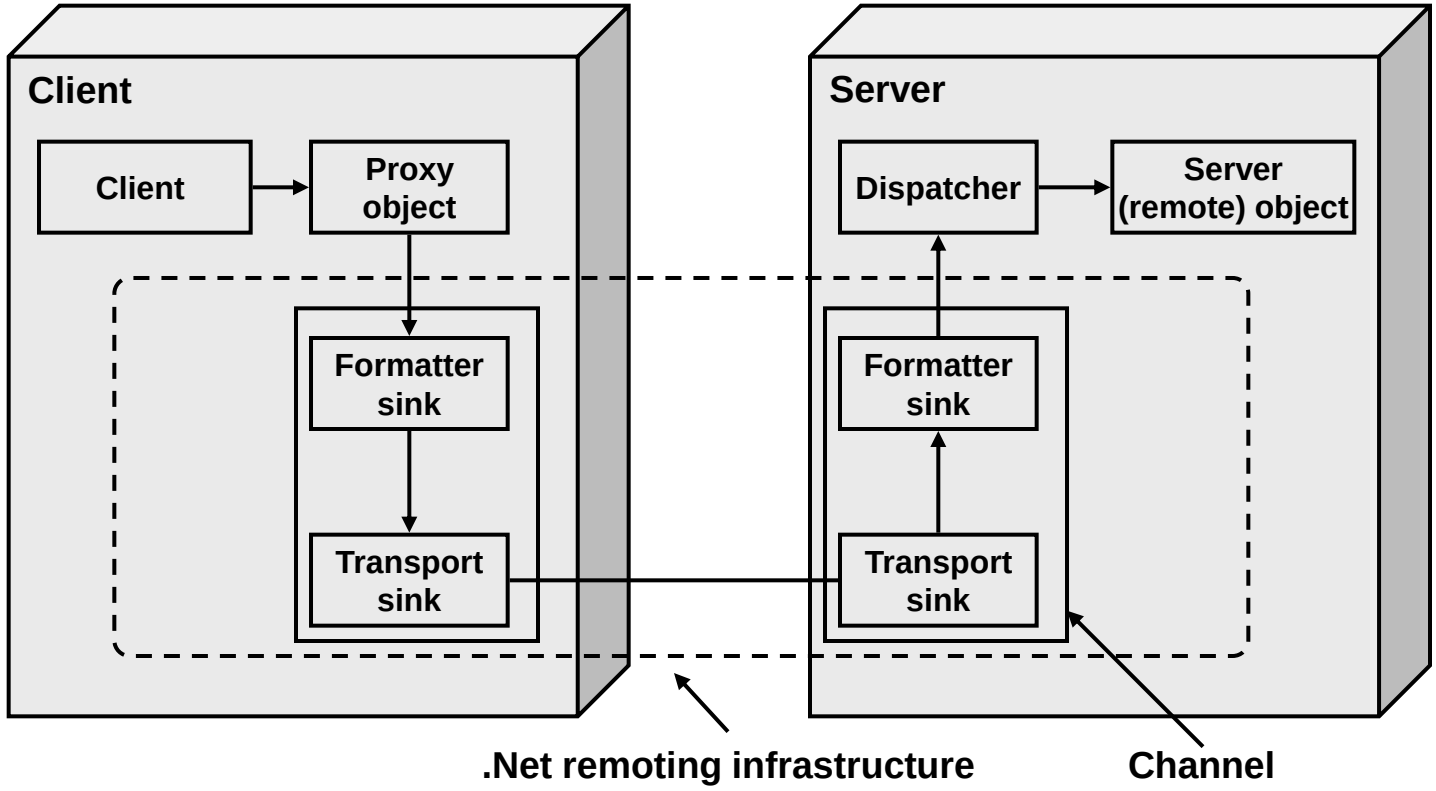
1. .Net Remoting architecture

Proxy: Client-side stub object that connects to the (remote) server object.

Channel: Transport channel for objects, defined by host + port + endpoint (= remote object service).

Dispatcher: Part of the .Net remoting infrastructure; dispatches method call to the server object.

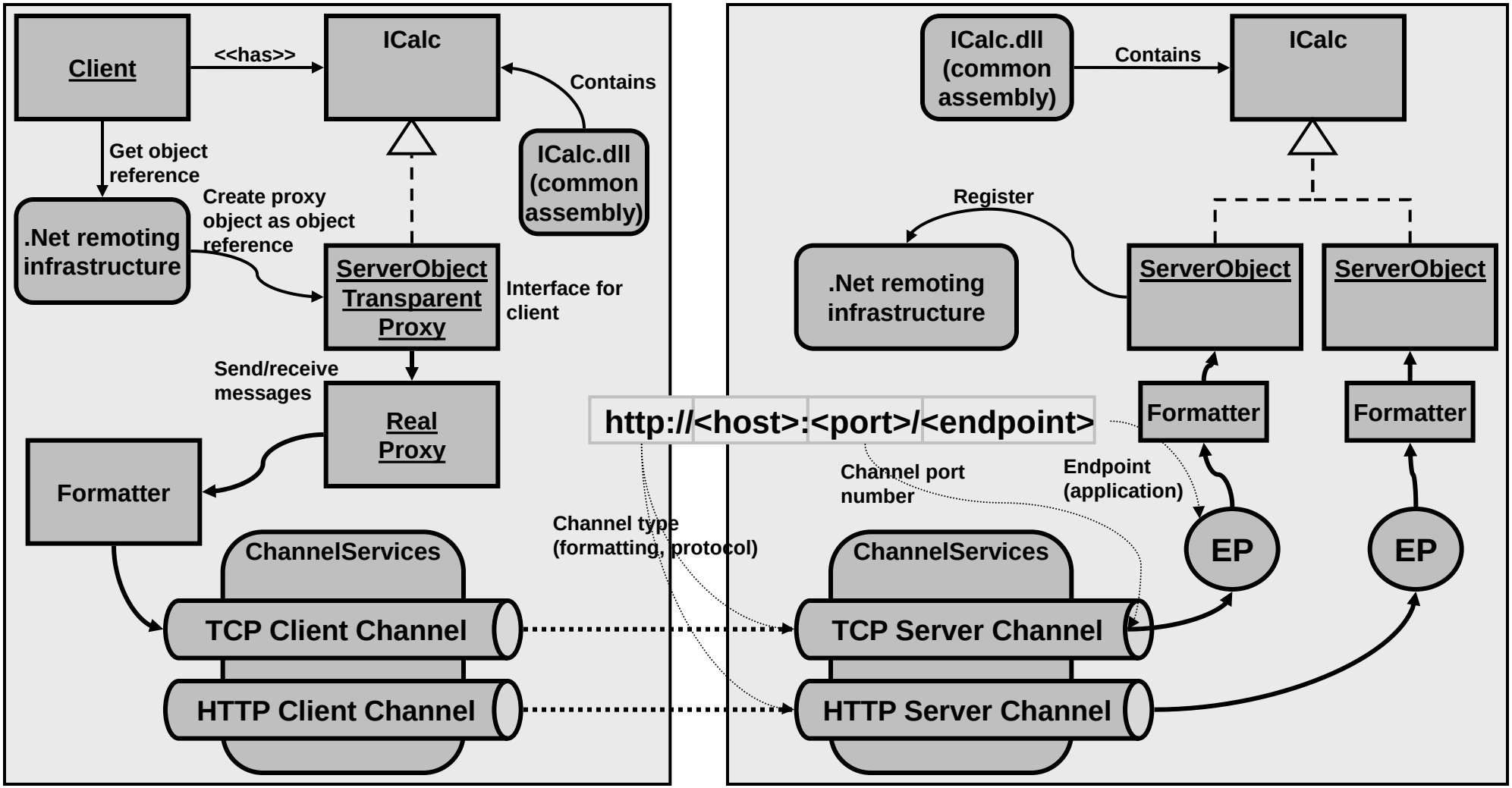
Formatter and Transport sink see below.



2. .Net Remoting concepts

Channel: Comprises a server port number and a formatting (=protocol such as HTTP or TCP)

Endpoint: Specifies the application that receives the calls (requests)



3. Remotable and nonremotable types (1/2)

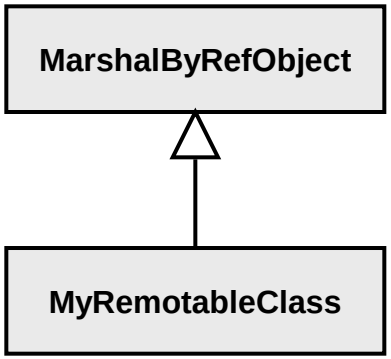
Nonremotable types:

Objects that neither derive from MarshalByRefObject nor are serializable.
Examples: File handles, sockets, window handles (in general objects that can be used only in the local context / application domain).

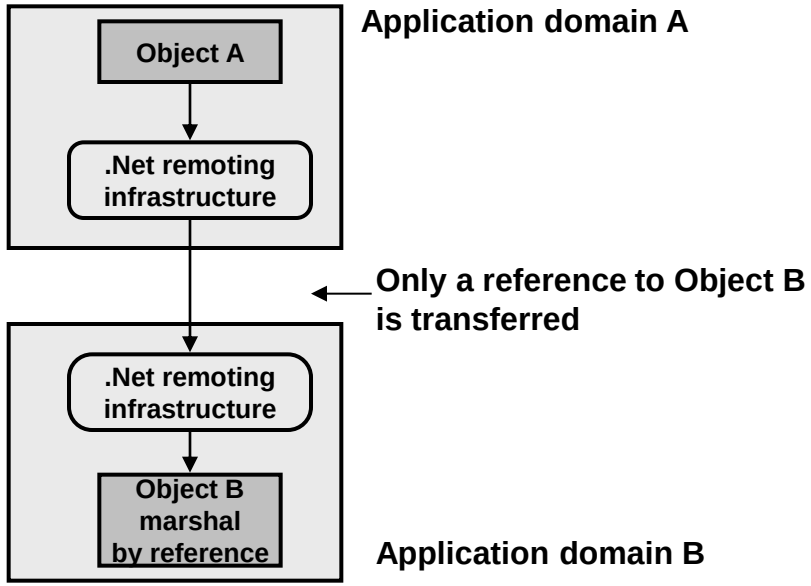
Remotable types:

1. Reference type objects:

Objects that derive from MarshalByRefObject are remotable.
The remote objects are marshalled by reference.
The client obtains a local reference object (proxy) to the remote server object.



A remote object becomes remotable by reference simply by deriving from the .Net class MarshalByRefObject

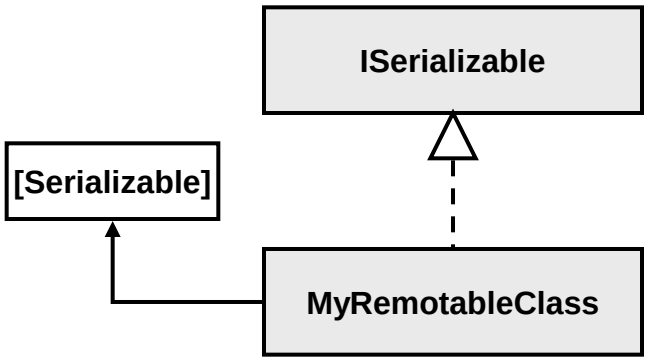


3. Remotable and nonremotable types (2/2)

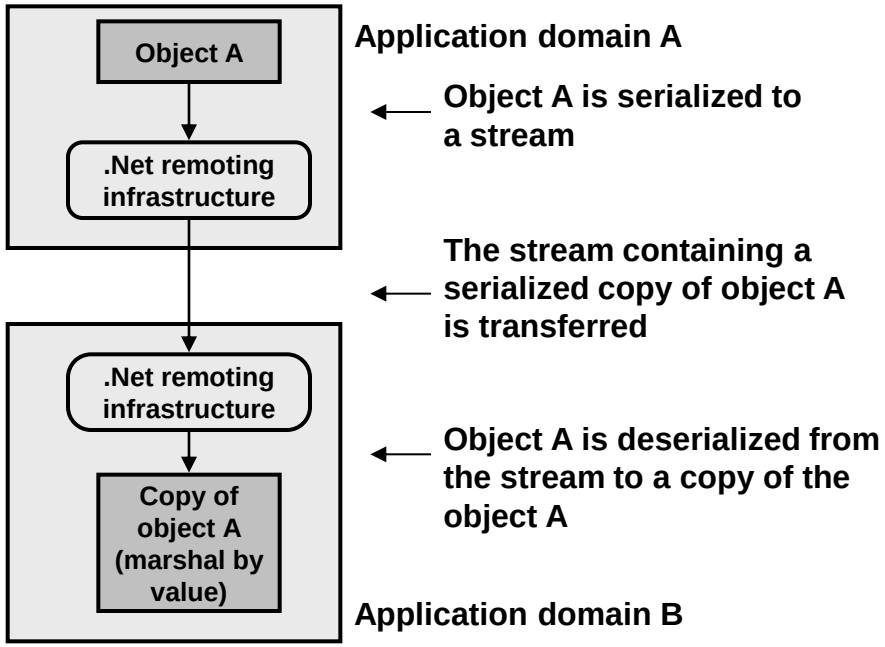
2. Value objects:

Objects that are serializable can be transferred into a different application domain. Serializable objects must be „tagged“ with the attribute [Serializable]. Additionally serializable objects may implement the ISerializable interface and provide a custom serialization (e.g. add some logging info to the serialized object stream).

Marshal by value:



The attribute Serializable is attached to MyRemoteClass marking it serializable (all members of the class need to be serializable as well). Optionally MyRemotableClass may implement the ISerializable interface allowing custom serialization.



4. .Net Remoting Server Object Activation Types (1/3)

Activation = creation and initialization of objects.

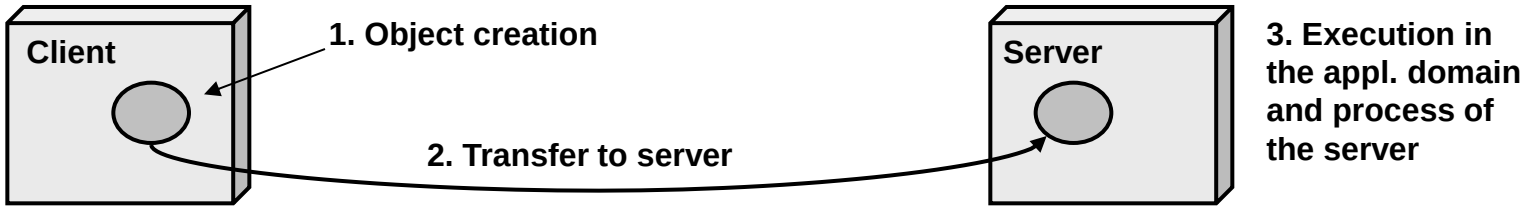
Activation of marshal by value types (Serializable):

Value type objects are activated through the de-serialization on the server side.

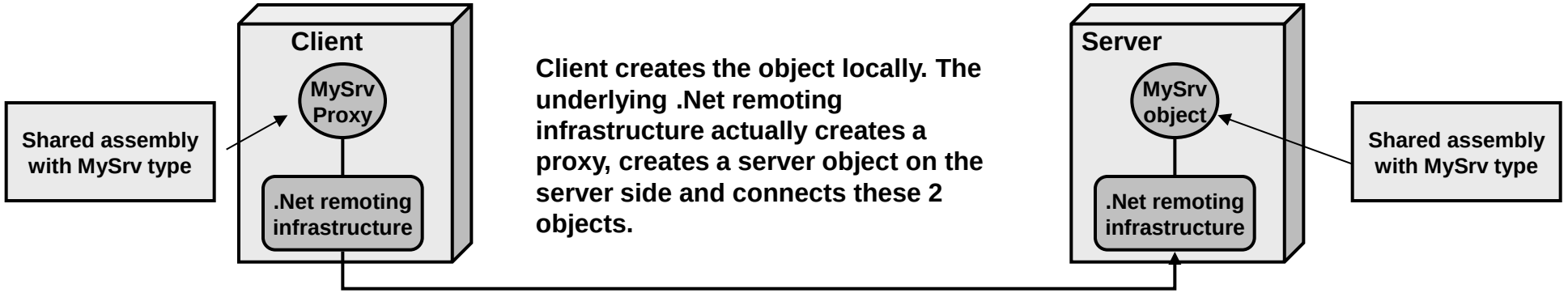
Activation of MarshalByRefObject types:

a. Client activated object (CAO):

- **Object is activated by the client, transferred to the server and the called method executed on the server side.**
- **Server object may retain state information between successive calls (stateful session).**



- **How it actually works:**



4. .Net Remoting Server Object Activation Types (2/3)

b. SAO - Server Activated Object (1/2):

- SAO call semantics is stateless (no session semantics between client and server object possible).
  - → Called „well-known“ types
  - → Published as an URI
  - → Server activates the objects and the client „connects“ to these.
  - → 2 types of server-activated objects

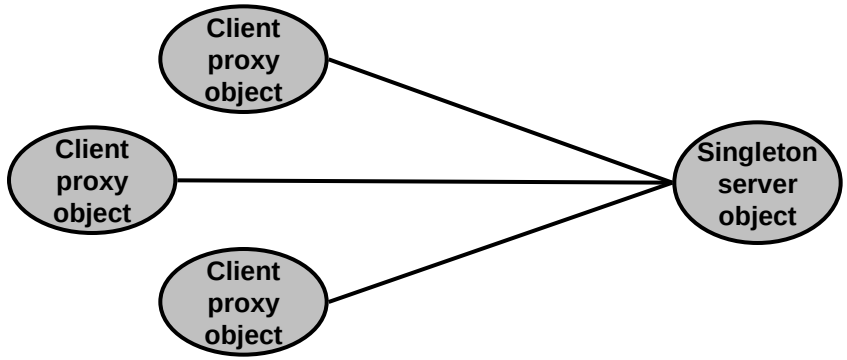
Singleton objects:

→ 1 global instance for all clients and for all remote object calls

→ Created when the first client accesses the server object.

→ Server registration as singleton:

```
RemotingConfiguration.RegisterWellKnownServiceType(
        typeof( SomeType ), "SomeURI", WellKnownObjectMode.Singleton );
```



4. .Net Remoting Server Object Activation Types (3/3)

b. SAO - Server Activated Object (2/2):

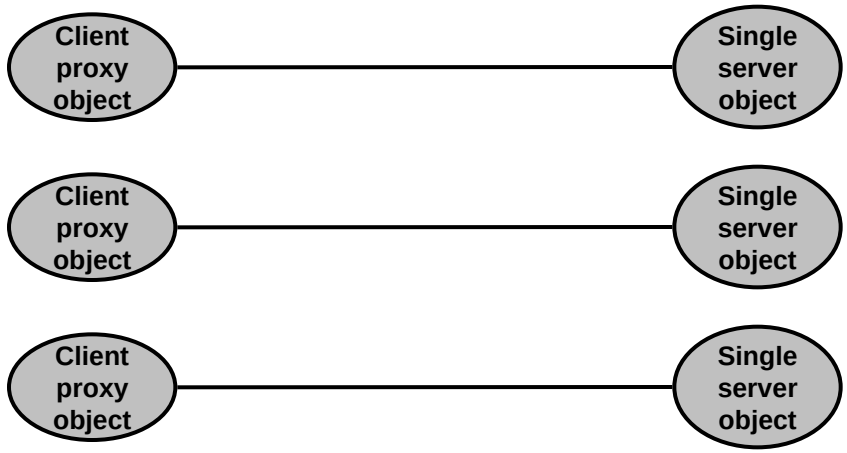
Single-call objects:

→ Individual object for each client method call.

→ Every method call is executed on a new server object instance, even if the call is made on the same client proxy object.

→ Server registration as single-call object:

```
RemotingConfiguration.RegisterWellKnownServiceType(
        typeof( SomeType ), "SomeURI", WellKnownObjectMode.SingleCall );
```

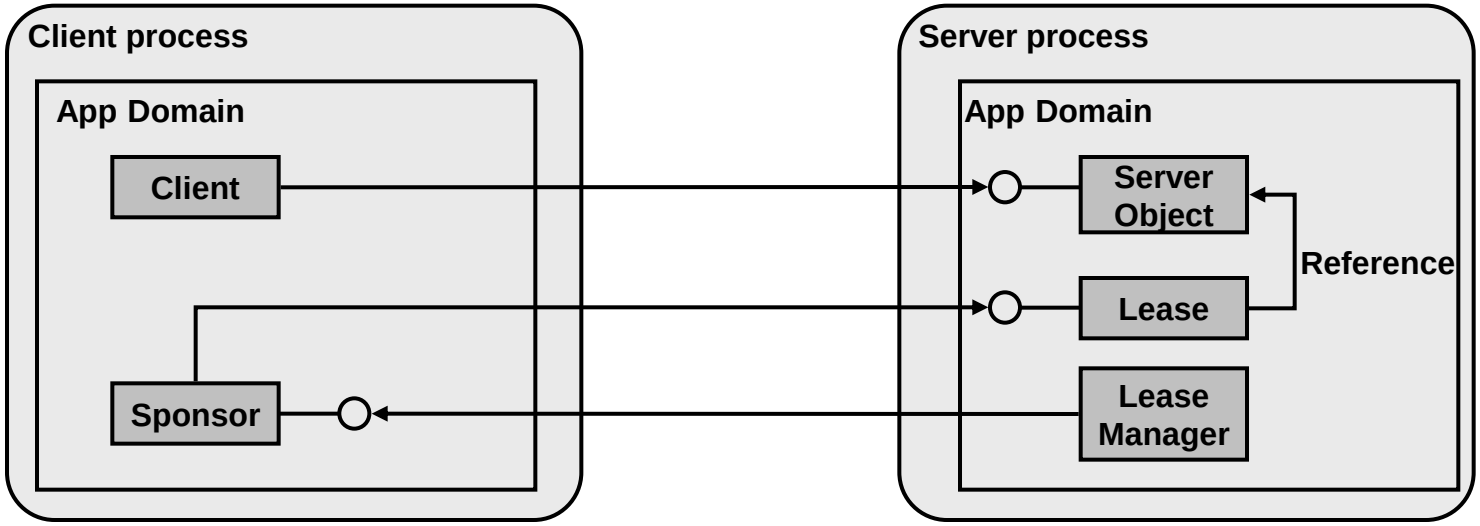


5. .Net remoting object lifetime control

SAO single-call objects: Server object lives for 1 call only
SAO singleton and CAO objects: Lifetime managed by the Lease Manager

The Lease Manager decides if a remote object (server object) can be marked for deletion (actual deletion is the job of the GC).
The Lease Manager contacts a sponsor in order to determine if a remote object can be marked for deletion or if the lifetime of the object should be extended.

- **Flexible design where client and server object lifetime are de-coupled.**
- **Lifetime of objects that are costly to create (lots of initialization etc.) can be given long lifetimes.**
- **Objects that hold precious resources may be given short lifetimes (free resources quickly).**



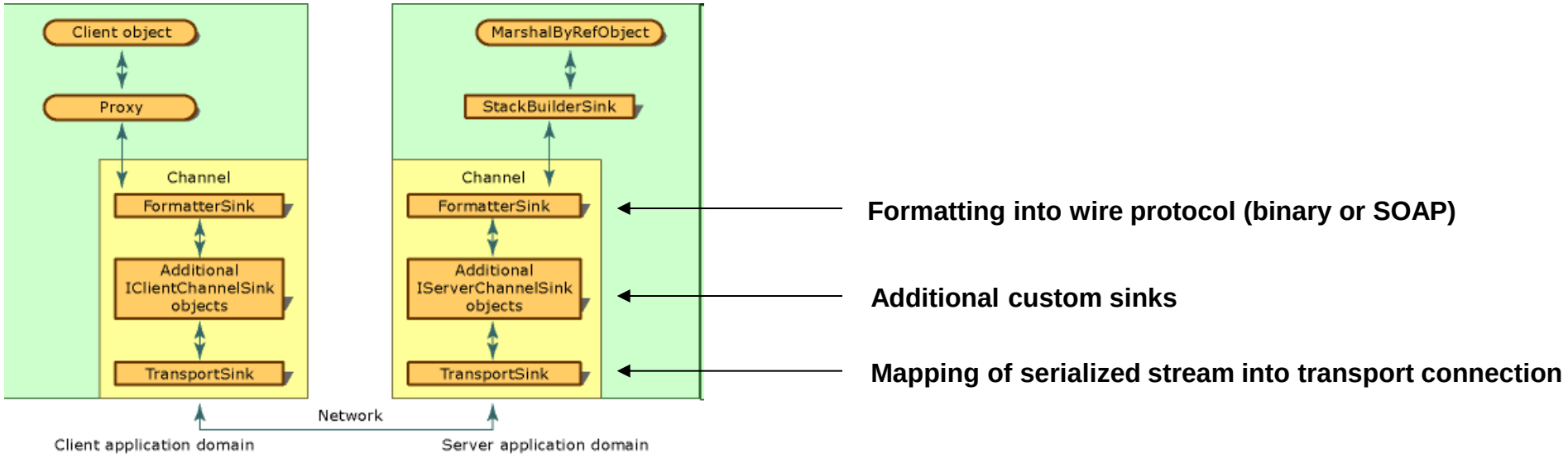
6. .Net remoting channel

.Net remoting channels are complex object-chains with at least 2 so called sinks (message processing objects):

a. Formatter sink: Convert the message or object to be transported to the required wire protocol (binary or SOAP)

b. Transport sink: Mapping of the serialized message stream into a transport connection (binary formatter: plain TCP, SOAP: HTTP)

The programmer may add additional sink objects (e.g. logging or filtering sink object that logs each message passing by).



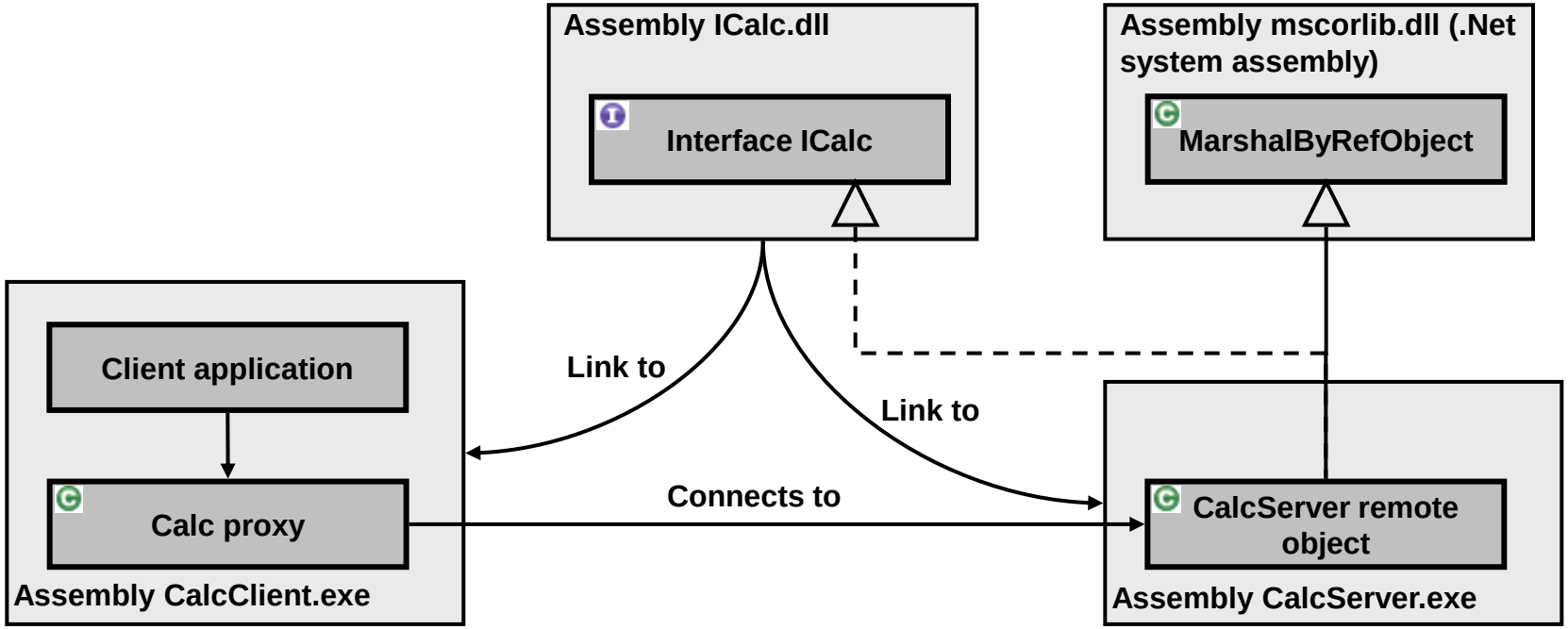
Source: http://msdn.microsoft.com/en-us/library/tdzwhfy3(VS.71).aspx

7. Assembly with remoting objects (1/2)

Both client and server must have the same assembly (.Net library in the form of a DLL or executable) containing the shared interface. Both client and server must be linked with an identical assembly containing the shared interface; only sharing the shared interface on source level does not work (.Net remoting run-time throws an exception).

1. SAO scenario:

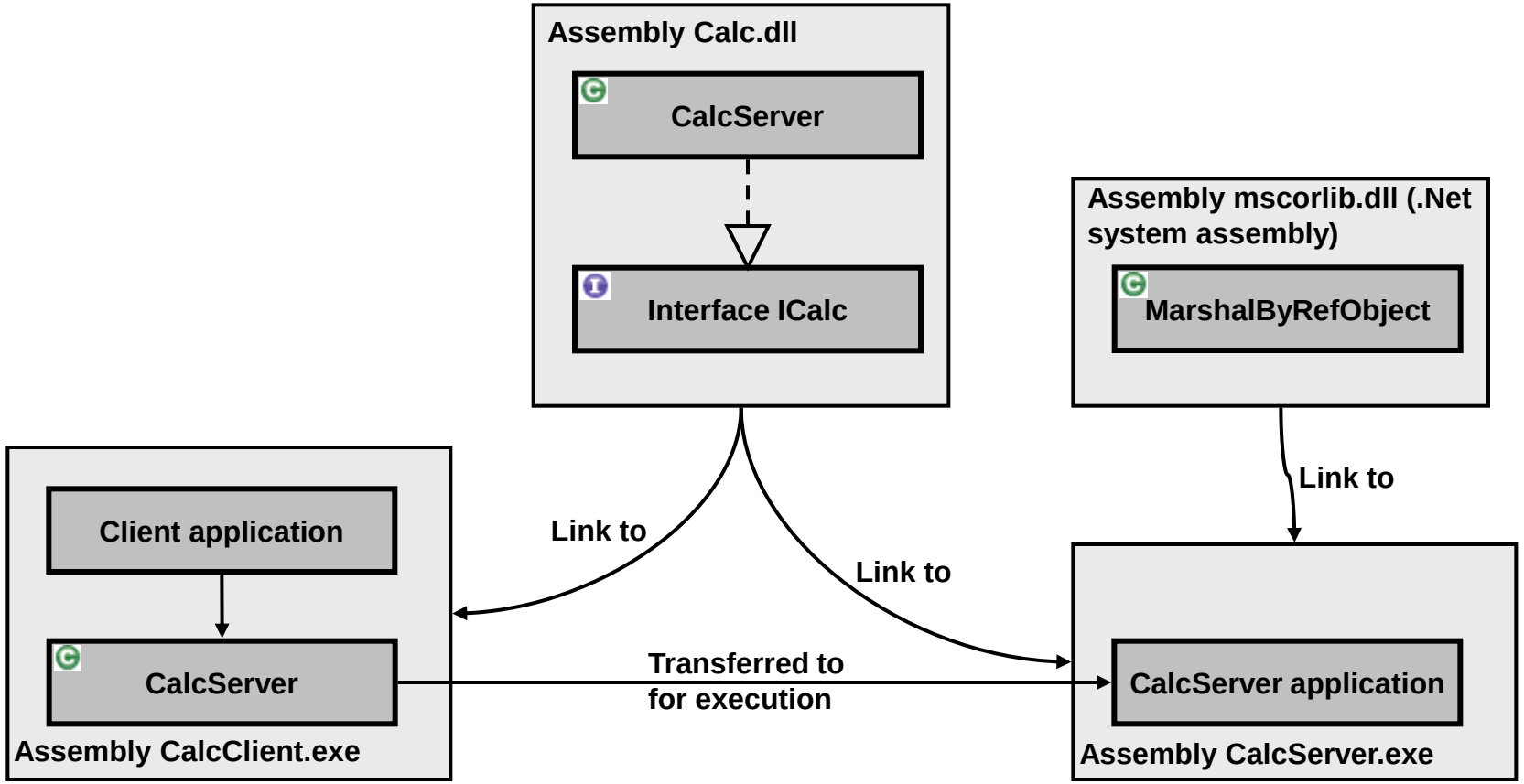
The shared assembly may only contain the interface (only minimal assembly with the shared interface needs to be deployed). The server implementation (class CalcServer) is completely hidden to the client.



7. Assembly with remoting objects (2/2)

2. CAO scenario:

Remotable object that extends MarshalByRefObject must be in the shared assembly because it is created / activated by the client, but executed on the server; so both client and server need the CalcServer class / object.



8. Configuration files instead of programmatic creation of objects (1/2)

.Net remoting allows using XML-files for configuring various settings on the server and client side, e.g. port numbers and formatters.
Advantage: Meta-programming without the need to change the code.
Disadvantage: If things don‘t work as expected debugging of configuration in XML-files is difficult (Visual Studio does not provide help for creating configuration files).

Example server config file:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.runtime.remoting>
    <application>
      <service>
        <wellknown mode="SingleCall" type="ICalc.CalcServer, ICalc" objectURI="ICalc.CalcServer"/>
      </service>
      <channels>
        <channel ref="tcp" port="60000" bindTo="127.0.0.1">
          <serverProviders>
            <formatter ref="binary" typeFilterLevel="Full"/>
          </serverProviders>
        </channel>
      </channels>
    </application>
  </system.runtime.remoting>
</configuration>
```

8. Configuration files instead of programmatic creation of objects (2/2)

Example client config file:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.runtime.remoting>
    <application>
      <client>
        <wellknown type="ICalc.CalcServer, ICalc"
          url="tcp://127.0.0.1:60000/ICalc.CalcServer.soap"/>
      </client>
    </application>
  </system.runtime.remoting>
</configuration>
```

9. Asynchronous remoting

Problem: Server method execution may take considerable time during which the client is blocked (waits for the response).
Solution: Use of standard asynchronous delegates of .Net

→ Further decoupling of client and server.
→ Similar to asynchronous message oriented interaction between client and server.