(Review) Distributed Objects and .NET Remoting

- Model a service as an object (with state and methods), whose interface and implementation reside on different machines
  - Clients access object using a proxy that presents the same interface
- Benefits
  - Better transparency: local and remote interactions look similar
  - Better support for state management
  - Tighter integration with language type system

Functionality of .NET Remoting

- Server makes available a type at a well-known end-point (channel + URL)
  - An operation against this type results in an instance being created (unless one already exists)
- Client makes a request for the type
  - Obtains a proxy that provides the same interface as the type
    - Method invocations against the proxy are forwarded to the server object
  - Can forward the proxy on to other objects

.NET Remoting: Example

- Namespaces
  - System.Runtime.Remoting.RemotingServices
    - Provides fundamental remoting infrastructure
      - Remoting configuration, Connecting to remote object instances, Exposing “well known objects”
  - System.Runtime.Remoting.ChannelServices
    - Channel registration and management
  - System.Runtime.Remoting.LifetimeServices
    - Lease-based lifecycle management for objects
  - System.Runtime.Remoting.TrackingServices
    - Universal hooks for tracking remoting activities

- [ Code walkthrough of a simple string-reverser application using .NET Remoting ]
Example Details

Application
- Server: A “string reverser” object with internal state
  - Two classes: RType, RTypeExtended (extends RType)
- Client: Interacts with server to get strings reverses

Highlighted functionality
- Basic setup using configuration files
  - Stateful nature of server
    - singleton and single call options
  - Modular selection of channels
- Explicit setup of remoting: server and client ends
- Passing of proxies and object references
  - Creation of proxies from an object reference
- Run-time type inspection of object reference to generate a custom proxy

.NET Remoting: Remotable Types

- Any type that extends the class MarshalByRefObject
- Type can have fields …
  - Define the state of the object
- … and methods
  - Object references that are not remote should be passed by value
    - Require that the object implement the ISerializable interface or have the [serializable] attribute
    - A new copy gets created at the destination
  - Remote object references are passed by reference
    - Results in an ObjRef instance: the representation of the object reference
      - Contains all of the information required to locate and access the object from anywhere on the network
      - Class hierarchy, interfaces it implements, object URI, …

.NET Remoting: End-Points

- End-point = Channel + Name
- Channel: Transport for transferring messages to/from the remote object
  - .NET framework provides the following three: TCP, HTTP, SMTP
    - In each case, a unique port number is required
      - Application developer can build their own
      - At least one channel must be registered for remotable objects
      - Can have many channels per application
- Name: A Universal Resource Identifier (URI) that identifies the type being exported
  - Must be consistent with a registered channel
    - tcp://..., http://..., smtp://...
- Clients request type by supplying the name and port number using standard URI conventions

.NET Remoting: More About Channels

- In general, a channel is built up out of a chain of processes
  - “channel sink chain”
- Formatter sinks
  - Serialize messages into streams of bytes (wire-format)
    - Built-in: SOAP and Binary Formatter
    - Custom Formatters allow talking to any endpoint
      - E.g., IIOP.NET allows interoperability with CORBA
- Transport Sinks
  - Establish a connection to the transport sink on the client/server
    - Forward the formatted message to another transport sink
    - Built-in: TCP and HTTP
- Custom sinks can be added to the chain
  - Logging, encryption, …
.NET Remoting: Proxies

- Implementation consists of two parts
  - Real proxies: The (generic) communication layer
  - Transparent proxies: Provide the same interface as the remote object
    - Built dynamically by real proxy

Client

"Proxy"

IMessageSink

Channel

Server

.NET Remoting: Object Activation

- Type available only as long as there is an active listener
  - With registered channel
  - Different from COM, Java RMI

- Server exposes well known object for clients to connect
  - Bound to known channels with known name
    - Two kinds:
      - "single call": Object instance is created for each call on channel
        - Implements the stateless model of the web.
      - "singleton": One shared instance provided for all clients
        - Serves as "gateway" into stateful application
    - Object instances are created on-demand by the server (server activation),
      or can be managed by the client (client activation)