V22.0480-004
Web Services Architecture and Programming

Lecture 10
.NET Remoting (cont’d)

Announcements

• Lab 3 handout available from the web site
  • A chat-like application using .NET Remoting
    • Expect you to use only the subset of functionality discussed in the lectures
  • Due back by October 27th (Monday), 11:59pm

• Mid-term exam on Tuesday, October 14th
  • Networking Fundamentals, Sockets, RPC, Distributed Objects
    • IP addresses, port numbers, protocol stack
    • BSD interface for UDP and TCP sockets
    • RPC architecture, SunRPC, XDR, NDR, XML-RPC
    • Distributed objects, CORBA, Java RMI, .NET-Remoting
  • Lectures 1-10
  • Labs 1, 2, sample application of Lab 3
  • Discussions on mailing list

.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
- Can provide custom implementation (by extending **RealProxy** class)
- Useful for load-balancing, partial remoting, …
.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

- Can be activated by the server or by the client
  - Server activation discussed in sample application
  - Alternative to explicit Marshal call:
    RegisterWellKnownServiceType( ... )

.NET Remoting: Client Activation

- Each client activation creates one object
  - Object’s lifetime extends until the earlier of
    - Client dropping a reference to the object
    - Object’s lease expiring [ more about this later in the lecture ]
  - Can store per-client state, receive constructor arguments

- Server registers a client-activated type using RegisterActivatedServiceType
  - Takes as argument the type being registered

- Client requests activation using CreateInstance call
  - Requires a network round-trip (unlike server activated objects)
    - Client sends message to server (w/ constructor arguments)
    - Server instantiates object, and returns ObjRef
    - Client creates proxy from ObjRef

..NET Remoting: Lifetime Management

- Unlike the reference-counting based scheme used in Java RMI-like systems, .NET Remoting relies upon a lease-based scheme
  - Object references valid for a fixed time (the “lease”)
  - Leases can be extended
    - Either implicitly or explicitly
  - Once lease expires, object is garbage collected

- Benefit: Significant reduction in network traffic

- Example: Server-activated singleton objects
  - Initial lease of 5 minutes upon creation (InitialLeaseTime)
  - Lease is extended by 2 minutes on every method invocation (RenewOnCallTime)
  - When lease expires, this lease expires
    - However, object can get recreated on a subsequent request
    - Not true for explicitly “marshaled” objects

.NET Remoting: Lifetime Management (cont’d)

- Leases can be explicitly extended …
  - … By clients using the proxy
  - … By special objects called “Sponsors”

- Sponsors implement an interface: ISponsor
  TimeSpan Renewal( ILease le )

- Sponsors can reside on server, client, or independent machine
  - Makes the most sense to run them on machines which are continually connected to the machine hosting the object