Announcements

- Lab 2 (XML-RPC) out today, due back October 7th
- Please follow all of the instructions in the writeup
  - URL in RPCInterface.cs should have your user name, port number
  - Both the bin and appstate folders need to exist, with those names and specified permissions
  - web.config file should exist and contain no special characters
    - Cutting-and-pasting from PDF may introduce these characters
    - The attribute values in the add command are just strings within double quotes
  - Remember to run tcpTrace with the appropriate port number before you start the client
  - Verify that
    http://localhost/VSDev/Public/<your user name>/XMLRPC/sumAndDiff.rem
    produces a listing of the server XMLRPC methods

(Review) Sockets and RPC

- Sockets API
  - Explicit setup, use, teardown of connections
  - Service naming: Network end-point (IP address + port number)
  - Wire format: Byte streams where message boundaries are not preserved
    - Client and server need to agree upon what each byte in the message means

- RPC API
  - Procedural interface to service
  - Client- and server-side stubs translate between procedure-call like interface and underlying message passing
    - Stubs created automatically starting from IDL-based service interface
  - Service naming: Host name + string (or URL)
  - Wire format: IDL defines restricted set of types and their encoding
  - Shortcoming: Protocol is stateless

Distributed Objects

Rationale: Object-orientation applied to distributed programming

- Hiding of differences between local and remote interactions
  - By defining notion of remote/distributed objects
  - Call site looks the same in local and remote case
    - All necessary information encapsulated in the object reference
      - XML-RPC proxies already provide this benefit, unlike SunRPC handles

- Better support for state management at client and server
  - Object reference identifies necessary state
    - E.g., mail folder on the server in Lab 2
  - Transmission of object references avoids (expensive) state transfer

- Closer integration of type system with distributed architecture
  - Compile-time checking of message format, other errors
  - Run-time inspection of object reference to determine appropriate format

Example systems: CORBA, DCOM, Java RMI, .NET Remoting
Distributed Objects – Definitions

Object
- Encapsulates data (state) and operations on that data (methods)
- External access to object state only via its public methods (interface)
  - An object may implement multiple interfaces
  - An interface can be implemented by multiple objects

Distributed object
- Interface exists separate from the implementation (on separate hosts)
- Implementation can involve one or more objects
  - When implementation contained in one object: remote object
  - Reasons for splitting implementation across multiple objects
    - Modularity, convenience, security
    - Load-balancing
    - ...

Programming with Distributed Objects – Overview

Client host
- Client program
- Proxy
- marshalled invocation
- Same interface as object
- Network

Server host
- Server program
- Server Stub
- object
- methods
- interface
- Stub invokes method

Client OS
- Marshalled invocation
- Message can contain remote object references
- Differences from RPC shown in green font

An Application Example

- Conference room reservation system

- Reserve(429, 10/02/2003, …)
- Register with coordinator
- Who controls Room 429?

- CS Dept. Office
- Math Dept. Office
- WWH Conf. Room Coordinator

Host A
- Ocs

Host B
- Ocs

Host C
- Omult

Host D
- Ocs

Common Issues for Distributed Objects – (1) Parameter Passing

How are parameters/results passed between client and server?

- RPC-like passing of structures
  - Well-defined wire format for structures built out of a restricted set of types
- Plus, passing of (opaque) object references
  - Provides generality: object definition may use types other than above set
- Distributed object implementations automatically determine whether to pass the structure (by value) or to pass a reference
  - References to remote objects passed as is
  - References to other objects passed by value
  - Copies of objects can be created as a side-effect of method invocations
    - Would like to prevent this, but exposing it would violate transparency
    - Distributed object systems provide mechanisms to control what state gets sent
      - Custom marshalling code
Parameter Passing (cont’d)

Client code contains reference to local object L and remote object R

Remote invocation with L and R as parameters

New object, reference

Copy of object L

Copy of reference to R

Server code (method implementation)

Common Issues for Distributed Objects – (2) Object Implementation

How is an implementation (code) associated with the object?

- Compile-time objects: Implementation is instance of a class declared at compile-time in an object-oriented language (e.g., Java, .NET languages)
  - One-to-one correspondence between object (distributed application) and corresponding class fields/methods
  - Compile-time construction of client and server-side stubs
  - Dependence on a particular programming language
    - Less of an issue with the .NET framework because of the CLR

- Run-time objects: Implementation is made to appear as an object
  - Objects defined solely in terms of their interfaces
  - Implementation can be written in any language
  - Implementations registered with an object adaptor, which handles the mapping between interfaces and implementations
    - More complicated to use

Common Issues for Distributed Objects – (3) Object Binding

How are client requests routed to the server object?

- Unlike RPC, distributed object systems support system-wide object references
  - References can be passed along as method parameters
  - Client code must bind to reference before it can invoke methods on it

Two types of binding

- Implicit: Client code treats all object references the same
  - Typically requires language-level support
- Explicit: Client must call a separate function to obtain a local proxy
  - As with RPC protocols

In each case, implementation must map object reference to lower-level network constructs (host names, ports)

- Representation of a object reference must provide necessary information

Common Issues for Distributed Objects – (4) Representation of Remote Object References

- Object reference can store end-point (host, port) information
  - Convenient for mapping, but
  - Cannot migrate object to a different machine
  - Cannot restart/rebind server program to a different port

- As in RPC systems, can add a nameserver
  - Unlike RPC systems, can provide information about remote objects on other machines as well
  - Object reference:
    [nameserver address, globally unique server identifier, object identifier ]

  - Nameserver knows about several server programs
  - Server program can contain multiple remote objects

- References can also contain additional information
  - Protocol for connecting to the server
  - Proxy information
Common Issues for Distributed Objects – (5) Object Persistence and Activation

What happens to an object’s state when the server program is not running?

- **Persistent** objects continue to exist even when not in memory
  - Independent of server program
  - State is stored on disk or other persistent storage
  - Object is **activated** on demand (e.g., when one of its methods is invoked)

- **Transient** objects exist only as long as they are in memory
  - Object lifetime determined by containing server process, or by the presence of references to it
    - Languages with garbage collection must account for remote references
      - E.g., Java, .NET framework languages

CORBA

**Common Object Request Broker Architecture**

- A **language-neutral** distributed object architecture
- Standardized by the Object Management Group (OMG)
  - Same folks who standardized DCE RPC
  - Microsoft’s answer to CORBA: DCOM
    - Built on top of Microsoft RPC and COM (Common Object Model)
    - Efficient support for interactions between objects located on the same machine

**Features**

- **Parameter passing**: Uses own wire format, called CDR
- **Object implementation**: Associated with IDL interface at run time
- **Object binding**: Explicit, via separate function calls
- **Object references**: Refer to a nameserver, support dynamic invocation
  - Permit queries about implemented interfaces
- **Object persistence**: Persistent, Activatable

CORBA Architecture

![CORBA Architecture Diagram]

CORBA Object Services

A standard set of services (also implemented as CORBA objects) that provide commonly-required low-level and basic functionality

- Collection Service
- Concurrency Service
- Event Service
- Externalization Service
- Licensing Service
- Life Cycle Service
- Naming Service
- Persistence Service
- Properties Service
- Query Service
- Relationship Service
- Security Service
- Time Service
- Trader Service
- Transaction Service
- …
Java Remote Method Invocation (RMI)

Java language-level support for remote objects

- **Parameter passing**: Uses own wire format
  - Parameters to RMI methods extend a predefined interface (Serializable)
    - “serializing” = marshalling
  - Possible to define custom serialization routines

- **Object implementation**: Compile-time definition
  - RMI interfaces extend a predefined interface (java.rmi.Remote)
  - Implementation class implements RMI interface
    - Typically by extending a predefined class
      - java.rmi.Activatable, java.rmi.UnicastRemoteObject
  - Client and server stub code automatically generated by invoking rmic
  - Additional exceptions defined for remote interactions

Java RMI (cont’d)

- **Object binding**: Both explicit and implicit
  - **Explicit**: Using static methods in the java.rmi.Naming class
    - Server binds/rebinds name (string) with a nameserver (rmiregistry)
    - Client looks up name with the nameserver
  - **Implicit**: Whenever a client receives a remote object reference
    - Can invoke methods on it as if it were a local interface

  - What happens if the client does not have the stub classes for the reference?
  - RMI implementation includes a dynamic class-loading feature
    - A reference to the bytecode is sent along with the reference
  - Cannot really do this in a language-neutral fashion

- **Object references (RemoteRef)**
  - Stores information about the server name, unique object ID, codebase
  - Distributed GC using reference counts

- **Object persistence**: Both transient and persistent objects supported