

Agenda

- **Last time: Thurs (5-6:15) / Friday (3-4:15):** chpts 4-5
 - General RPC/RMI example
 - General RPC/RMI terminology
 - Case study: SUNRPC
 - Case study: Java RMI
- **This time**
 - Brief CORBA overview
 - General pros/cons of RPC/RMI
 - Security intro (chapt 7)
 - Assignment #1 Due, Assignment #2 Out (due Thur Mar 1 – 2 weeks)
- **Next time (Tues Feb 20)**
 - "A Note on Distributed Computing" – *Please Read it before class!*
 - Security (chapt 7)
- **Note: No class next Thursday Feb 22 (Marty at Conference)**

CS451: Distributed Systems (Spring 2007)

Before we start

- **Change the make-up dates/times? (must decide by Mar 15, when next make-up is scheduled)**

CS451: Distributed Systems (Spring 2007)

Before we start: Assignment #2

- **Goal: Availability, persistence, and Java RMI experience**

CS451: Distributed Systems (Spring 2007)

Recap from last time

- **Goal of RPC/RMI: network transparency**
 - But does not necessarily achieve this goal
- **Many general issues**
 1. How do we specify the interface?
 2. How we generate stubs?
 3. How are parameters passed?
 4. How does a client find/bind to a server?
 5. What are the invocation semantics?
 6. Ease of use
 7. Performance
- **SUNRPC: pioneering effort**
- **Java RMI: more advanced**

CS451: Distributed Systems (Spring 2007)

Recap from last time: RPC/RMI

- **Please remember that all communication is over sockets**
 - But you (the programmer) do not have to deal with them directly (yay!)

CS451: Distributed Systems (Spring 2007)

Events and Notifications

- **Heterogeneous: Components in a distributed system that were not designed to interoperate can be made to work together.**
- **Asynchronous: Publishers and subscribers are decoupled.**

CS451: Distributed Systems (Spring 2007)

Events and notifications - issues

- **Delivery semantics.**
- **Roles for observers:**
 - Forwarding.
 - Filtering of notifications.
 - Patterns of events.
 - Notification mailboxes.

CS451: Distributed Systems (Spring 2007)

Remote Object Activation

- **Why?**
 - To free resources from servers with seldom-used services.
 - To enable devices with limited resources to activate multiple kinds of services.
- **Transparent to RMI clients.**
 - Remote interface/client code is the same.
- **Server code needs modifications:**
 - Extends `Activatable` class
 - Constructor receives `ActivationID`, `MarshaledObject`.
 - **Main method steps:**
 - Create `ActivationGroupDesc`
 - Register activation group descriptor with `ActivationSystem`
 - Create an `ActivationGroup`
 - Create an `ActivationDesc` with class name, codebase
 - Register the activation descriptor with `ActivationSystem`
 - Register the stub (Returned in previous step) in registry.

CS451: Distributed Systems (Spring 2007)

Distributed Garbage Collection

- **Remote service developers don't need to track remote object clients to detect termination.**
- **How?**
 - When a reference to a remote object is created in a JVM, a `referenced` message is sent to the object server.
 - A reference count keeps track of how many local references there are.
 - When the last reference is discarded, an `unreferenced` message is sent to the server.
 - When a `Remote` object is not referenced by any client, the run-time refers to it as a `weak` reference.
 - The weak reference allows the JVM's garbage collector to discard the object if no other local references exist.

CS451: Distributed Systems (Spring 2007)

RMI Deployment Issues

- **Dynamic Class Loading**
 - What happens if a new object is passed using RMI, and the defining class is not available to the remote system?
 - Recall that you can pass an object with an interface type (e.g., `Runnable`) which can have multiple implementations.
 - We need a way to download such code dynamically.

CS451: Distributed Systems (Spring 2007)

RPC vs. RMI

- **Many of the same issues...**
 - Can be evaluated in terms of ease-of-use, "underlying mechanisms",
- **...but there are some differences**
 - "procedure-based" vs. "object-based"
 - Event-based programming (e.g., callbacks)
 - Remote object activation
 - Garbage collection

CS451: Distributed Systems (Spring 2007)

A Few Words about CORBA...

- **CORBA = Common Object Request Broker Architecture**
- **Why use CORBA?**
 - Need to share information across enterprises
 - Need to integrate applications and systems
- **OMG creates/promotes/standardizes CORBA**
- **What is CORBA?**
 - a specification for the distributed object architecture defined by OMG
 - a middleware that sits between the application and the system
 - based on the OO model, allows software reuse
 - uses a broker to handle messages; broker provides the ability to choose servers that best fill the client's request
 - language independent
 - client can be developed in one language, say C++
 - server can be developed in another language, say Java

CS451: Distributed Systems (Spring 2007)

OMG

- **Object management group (OMG)**
 - Founded April 1989 with 11 members, now 900+
 - Purpose: "provide a common architectural framework for object-oriented applications based on widely available interface specifications"
 - Produces *specifications*, not *implementations*
 - 1st set of specs from OMG: 1990
 - Multiple specs from OMG, including UML and the Common Object Request Broker Architecture (CORBA)
- **Java RMI: Feb 1997 (JDK 1.1)**

CS451: Distributed Systems (Spring 2007)

Where in History?

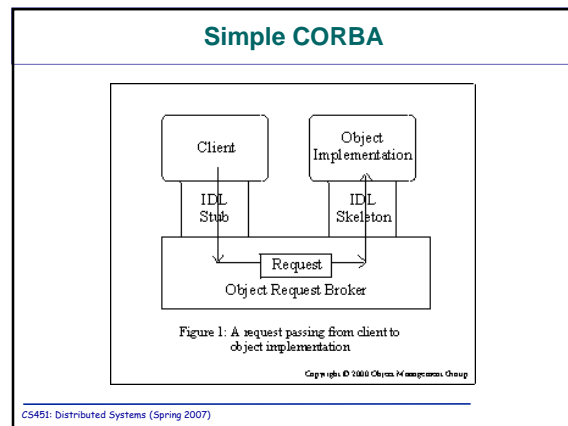
- **Object management group (OMG) cont.**
 - 1st commercial implementation of CORBA specs: 1992
 - COM: 1991-1993 (single machine)
 - Java: (May 23, 1995)
 - DCOM: 1996 (networked applications)
 - Java RMI (JDK 1.1 – Feb 18 1997)
 - COM+: 1998 (richer runtime and services)
 - .NET: Jan 2002 (integrates/defines web services)
 - Most recent: CORBA 3.0.2 in December 2002
 - <http://www.omg.org/gettingstarted>

CS451: Distributed Systems (Spring 2007)

Object Management Architecture (OMA)

- **System-oriented components**
 - Object request broker (ORB)
 - Object services (CORBA services)
- **Application-oriented components**
 - Application objects
 - Common facilities (CORBA facilities)

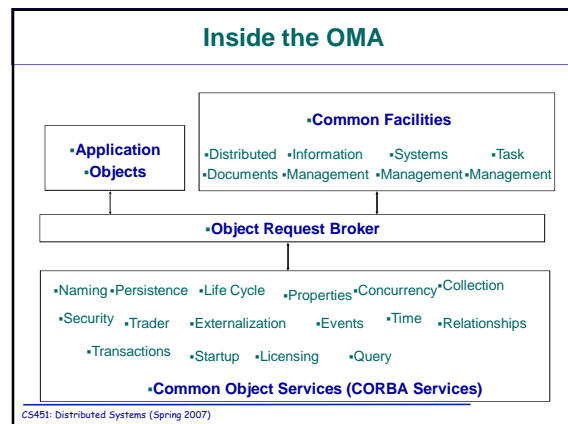
CS451: Distributed Systems (Spring 2007)



Benefits of CORBA

- **Strong support for interoperability**
 - language independence (IDL)
 - operating system independence
 - transport independence (GIOP/IIOP)
 - legacy integration
- **Object Request Broker (ORB)**
 - Simplifies distributed computing (broker/bridge)
- **Industry standard specifications**
 - design reuse
 - COTS
 - portability

CS451: Distributed Systems (Spring 2007)



Object Request Broker (ORB)

- **More conceptual than architectural**
 - There is no distinct process you can point to and say "that's the orb"
 - It's a library you link into your code
 - ORB is not required to be implemented as a single component, but is defined by its interfaces
- **ORB lets objects communicate transparently without concern for:**
 - the hosts where the objects run
 - the operating systems
 - the programming languages
- **ORB responsibilities**
 - mechanisms to find service implementations for requests
 - prepare service providers to receive requests
 - communicate data making up the requests
- **Commercial ORBs: Orbix – IONA; Visibroker – Visigenic; SUN**

CS451: Distributed Systems (Spring 2007)

What is an Object Reference?

- **Distributed computing equivalent of a pointer**
- CORBA defines the Interoperable Object Reference (IOR)
 - IORs can be converted from raw reference to string form, and back
 - Stringified IORs can be stored and retrieved by clients and servers using other ORBs
- an IOR contains a fixed object key, containing:
 - the object's fully qualified interface name (repository ID)
 - user-defined data for the instance identifier
- An IOR can also contain transient information, such as:
 - The host and port of its server
 - metadata about the server's ORB, for potential optimizations
 - optional user defined data

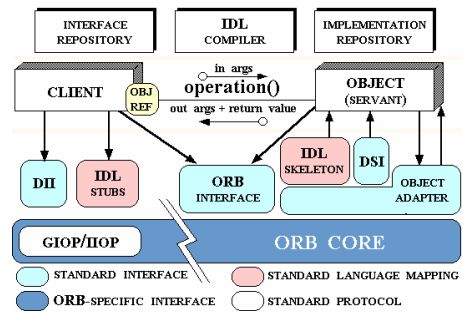
CS451: Distributed Systems (Spring 2007)

CORBA Object Characteristics

- **CORBA objects have identity**
 - A CORBA server can contain multiple instances of multiple interfaces
 - An IOR uniquely identifies one object instance
- **CORBA object references can be persistent**
 - Some CORBA objects are transient, short-lived and used by only one client
 - But CORBA objects can be shared and long-lived
 - business rules and policies decide when to "destroy" an object
 - IORs can outlive client and even server process life spans
- **CORBA objects can be relocated**
 - The fixed object key of an object reference does not include the object's location
 - CORBA objects may be relocated at admin time or runtime
 - ORB implementations may support the relocation transparently
- **CORBA supports replicated objects**
 - IORs with the same object key but different locations are considered replicas

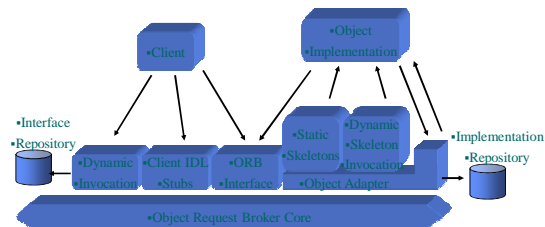
CS451: Distributed Systems (Spring 2007)

CORBA ORB Architecture



CS451: Distributed Systems (Spring 2007)

CORBA 2.0 ORB



CS451: Distributed Systems (Spring 2007)

Client-side Functions

- **Client IDL Stubs**
 - provide the static interfaces to object services
 - act as proxies for remote server objects
 - services are defined using IDL
- **Dynamic Invocation Interface (DII)**
 - support for run time discovery of methods to be invoked
 - CORBA defines standard API for looking up the metadata that defines the server interface, generating the parameters, issuing the remote call, and getting back the results
 - *Arguably, not used much.*
- **Interface Repository**
 - provides storage to store IDL information, which may be used by the ORB to perform requests
 - APIs allow you to obtain and modify the descriptions of all the registered component interfaces, the methods they support, and the parameters they require
- **The ORB Interface**
 - Consists of a few APIs to local services for an application

CS451: Distributed Systems (Spring 2007)

Server-side Functions

- **Server IDL Stubs**
 - provide static interfaces to each service exported by the server
 - stubs are created using IDL and compiler
- **Dynamic Skeleton Interface (DSI)**
 - provides a run-time binding mechanism for servers that need to handle incoming method calls for components that do not have IDL-based compiled stubs
 - rather than calling a specific stub routine for an operation, it is possible to specify an object, operation on it, and parameters to it through a call or sequence of calls
 - client must also supply the types of parameters passed
- **Object Adapter**
 - Primary goal is to interface an object's implementation with its ORB
 - accepts requests for service on behalf of the server's objects
 - provides the run-time environment for instantiating server objects, passing requests to them, and assigning them object IDs
 - each ORB must support a standard adapter called the Basic Object Adapter (BOA)
- **Implementation Repository**
 - contains information that allows ORB to locate and activate the implementation of a required server object

CS451: Distributed Systems (Spring 2007)

CORBA Services (1/3)

- **Naming Service**
 - allows components on the bus to locate other components by name
 - registry of pairs: *name, Interoperable Object Reference (IOR)*
 - Objects publish themselves; Clients subscribe and get references
 - "White pages"
 - Operations:
 - bind(), resolve() etc.
- **Life Cycle Service**
 - defines operations for creating, copying, moving, and deleting components on the bus
 - Operations:
 - move(), copy(), remove()
- **Externalization Service**
 - provides a standard way of getting data into and out of a component using a stream-like mechanism
 - Operations:
 - externalize_to_stream(), internalize_from_stream()
- **Startup Service**
 - enables requests to automatically start up when an ORB is invoked.

CS451: Distributed Systems (Spring 2007)

CORBA Services (2/3)

- **Persistence Service**
 - provides a single interface for storing components persistently on storage servers such as ODBMS, RDBMS
- **Concurrency Control Service**
 - provides a lock manager that can obtain locks on behalf of either transactions or threads
- **Transaction Service**
 - provides two-phase commit coordination among recoverable components
- **Query Service**
 - provides query operations for objects. It is a superset of SQL
- **Time Service**
 - provides interfaces for synchronizing time in a distributed object environment
- **Security Service**
 - provides a complete framework for distributed object security (authentication, access control, message protection, audit, non-repudiation, security management)

CS451: Distributed Systems (Spring 2007)

CORBA Services (3/3)

- **Relationship Service**
 - provides a way to create dynamic associations between components that know nothing of each other.
- **Licensing Service**
 - provides for monitoring usage of components
 - lets you meter the use of your components and charge accordingly
- **Properties Service**
 - provides operations that let you associate named values (or properties) with any component
- **Event Service**
 - allows components on the bus to dynamically register or unregister their interest in specific events
- **Trader Service**
 - provides a "Yellow Pages" for objects
- **Collection Service**
 - provides CORBA interfaces to generically create and manipulate the most common collections

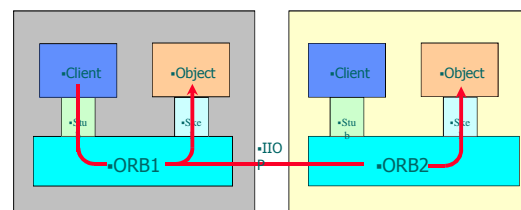
CS451: Distributed Systems (Spring 2007)

ORB Interoperability

- **Support networks of objects distributed across multiple heterogeneous CORBA-compliant ORBs**
- **GIOP (General Inter-ORB Protocol):**
 - Standard transfer syntax and a set of message formats for communication between ORBs
- **IIOIP (Internet Inter-ORB Protocol):**
 - The TCP/IP mapping of GIOP
 - Developers don't need to "learn" IIOIP; the ORB handles this for them

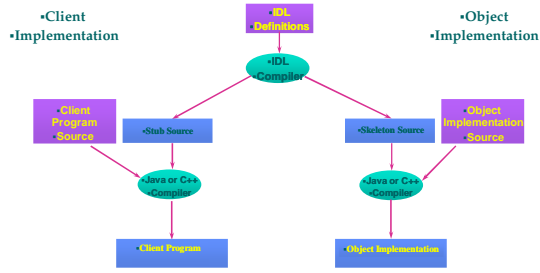
CS451: Distributed Systems (Spring 2007)

IIOIP Example



CS451: Distributed Systems (Spring 2007)

CORBA Development Process



CS451: Distributed Systems (Spring 2007)

Building a CORBA Application (using Java)

1. Write IDL interfaces for Server: Server.idl
2. Compile IDL file and generate Server_c.cpp and Server_s.cpp.
3. Write server implementation in C++: ServerMain.cpp
4. Compile the ServerMain.cpp with the files created by IDL
5. Write IDL interface for Client: Client.idl
6. Compile client.idl and generate associated java files such as ServerSymbolHelper.java and ServerSymbolListHelper.java etc.
7. Write client implementation in Java.
8. Compile client implementation and helper files together.
9. Run server and client programs together.

CS451: Distributed Systems (Spring 2007)

ORB/CORBA Advantages

- Static and dynamic methods invocations
- High-level language binding
- Self-describing systems (Interface Repository)
- Local/Remote transparency
- Built-in security and transactions
- Coexistence with existing systems
- Implementation language for client/server can be different

CS451: Distributed Systems (Spring 2007)