Intro to CORBA

by Rob Ratcliff

Redheaded Stepchild

or

Extreme Web Services?

Outline

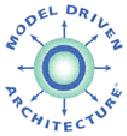
- History of CORBA
- Overview
  - IDL
  - IIOP/GIOP
  - Stubs/Skeletons
  - POA
- Developing a Chat Application
- Demo
- Addressing Objects
- CORBA Services
- Advanced Topics
- CORBA and XML
- Conclusion

What is CORBA?

- An Acronym for **C**ommon **O**bject **R**equest **B**roker **A**rchitecture
- Object Management Group's (OMG) open, vendor-independent architecture and infrastructure to allow computer applications to interact over the network
- A specification of the infrastructure and many horizontal services
- Deployed on embedded systems, hand-helds, desktops to main-frames
- Used, especially by Telecom, for high volume server applications

What is the OMG?

- Open membership, Not-for-profit consortium that produces and maintains computer industry specifications for interoperable enterprise applications



- Board of Directors

2AB IONA Technologies PLC Alcatel LION Bioscience BEA Systems, Inc. MITRE Borland Corporation MSC.Software Computer Associates Object Management Group DoCoMo Communication Laboratories Europe GmbH Objective Interface Systems Ericsson Utvecklings AB Oracle Corporation Fujitsu Limited PrismTech Hewlett-Packard Co. Sun Microsystems Hitachi, Ltd. Unisys Corporation IBM

Popular Slams

- CORBA is DEAD
- CORBA is a failed technology
- CORBA implementations are too expensive
- CORBA is too complex and has a steep learning curve
- CORBA is too rigid
- CORBA isn't interoperable between vendors
- IIOP/GIOP isn't human readable
- CORBA can't be used through firewalls

Popular Slams

Dogma Based

on Old Soggy Data

Anti-Pattern

- IIOP/GIOP isn't human readable
- CORBA can't be used through firewalls

CORBA History

- OMG founded by 11 companies (April 1989)
- CORBA 1.0 (October 1991)
  - IDL Defined
  - Dynamic Request Management and Invocation (DII) and Interface Repository
  - C language mapping
- CORBA 1.1 (February 1992)
  - First widely published version of specification
  - Interface for Basic Object Adaptor defined
- CORBA 1.2 (December 1993)
  - Ambiguous specifications resolved

CORBA History Continued

- CORBA 2.0 (August 1996)
  - First major overhaul
  - Interoperable communication protocols defined (GIOP, IIOP)
  - Layered security and transactional services
  - C++ and Smalltalk language mappings
- CORBA 2.1 (August 1997)
  - Secure IIOP and IIOP over SSL
  - COBOL and Ada language bindings
  - **Open source implementations begin to appear**
  - Competing RMI introduced by SUN in JDK 1.1
- CORBA 2.2 (February 1998)
  - Portable Object Adaptor (POA)
  - IDL/Java mapping specification

CORBA History Continued

- CORBA 2.3 (June 1999)
  - Objects by value
  - Java to IDL Language Mapping
  - IDL to Java Language Mapping
  - Bi-directional GIOP/IIOP
  - Most implementations are at this version
- CORBA 2.4 (October 2000)
  - Messaging
  - Interoperable Naming Service
  - Notification Service
  - Minimum CORBA
  - Real-time CORBA

CORBA History Continued

- CORBA 2.5 (September 2001)
  - Fault Tolerant
  - Portable Interceptors
- CORBA 2.6 (December 2001)
- CORBA 3.0 (December 2002)

Why CORBA?

- Flexibility
  - Full featured interface and data structure definitions
  - Pluggable communication protocols
- Platform/Language Independence
  - Unix, Windows
  - C, C++, Java, Tcl, Python, Perl languages (These are full featured programming languages.)
- Software Reuse – Cross-language support can eliminate porting exercises
- Integration – Can easily "wrap" legacy system

Why CORBA? (Continued)

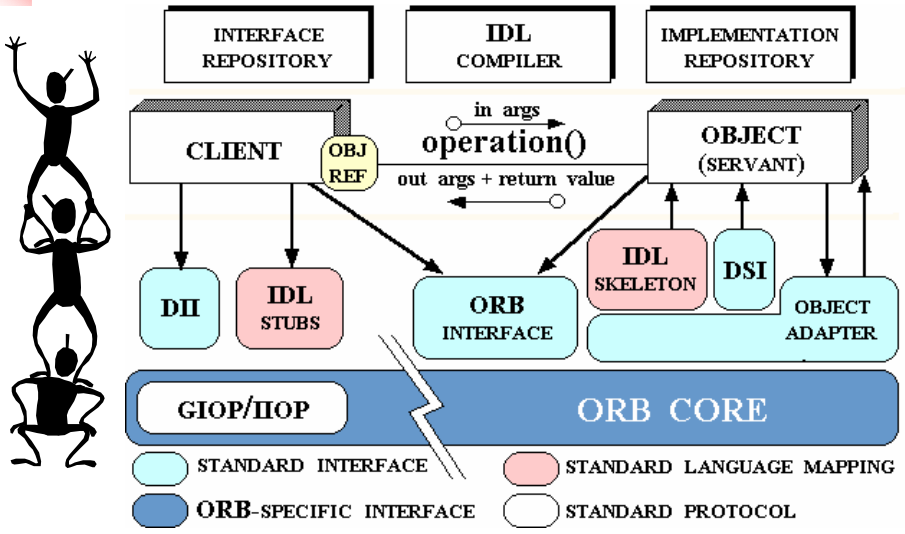
- Transparency – All objects appear to the programmer as if they were in the same memory space
- Strictly based on international standards
- Interoperability - Today's CORBA implementations can communicate reliably with each other
- Many free and commercial implementations
- Powerful Standardized Services
- Educational Materials and Classes
- Trained and experienced programmers available

A CORBA Renaissance is in the making…

Some CORBA Implementations

http://adams.patriot.net/~tvalesky/freecorba.html

CORBA Architecture Overview



Interface Definition Language, IDL

- A way to describe the interfaces to your objects
- No implementation features are specified except for constants
- Interfaces give you a way to specify semantics or a choreography of the work flow

IDL Primitive Data Types

IDL Type	Java Type
boolean	boolean
char/wchar	char
octet	byte
string/wstring	String
unsigned short/short	short
unsigned long/long	int
unsigned long long/long long	long
float	float
double	double
fixed	java.math.BigDecimal

IDL Modules, Interfaces & Methods

IDL

```
module example {

interface Vehicle {};

  interface Car : Vehicle {
    void start();
    void stop();
  };

interface CarLot {
   Vehicle trade (in Vehicle oldCar,
     out float pricePaid,
     inout CreditCard car) raises
      (BadDeal, InsufficientFunds);
  };
};
```

Java

```
package example;

public interface Vehicle {}

public interface Car extends Vehicle {
  void start();
  void stop();
}

public interface CarLot {
   Vehicle trade ( Vehicle oldCar,
     FloatHolder pricePaid,
     CreditCardHolder card ) throws
     BadDeal, InsufficientFunds;
}
```

IDL Structs, Sequences, and Arrays

IDL

```
typedef sequence<Car> Cars;

typedef long MyArray[10][10];

struct MyStuff {
    Cars cars;
    Plane plane;
    sequence<Tool> tools;
};
```

Java

```
Car[]; // and holder classes

int MyArray[10][10];

public final class MyStuff {
    public Car[] cars;
    public Plane plane;
    public Tool[] tools;
}
```

Attributes, Forward References, and Consts

IDL

```
interface Wife;
interface Husband {
    const string description=
                        "Some guy";
    attribute Wife wife;

    readonly attribute short age;
};

interface Wife {
    attribute Husband husband;
};
```

Java

```
public interface Husband {
    String description="Some guy";

    void wife (Wife wife);
    Wife wife();

    short age();
}

public interface Wife {
   void husband (Husband
    husband);
   Husband husband();
}
```

IDL Enum

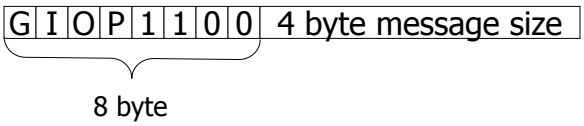
```
enum Sex {
      female,
      male
};
```

```
public final class Sex {
  public static final int _female =0;
  public static final int _male = 1;

  public static final Sex female = new Sex(_female);
  public static final Sex male = new Sex(_male);
  protected Sex ( final int value) {
          this._value=value;
  }
  public int value ()  {
          return _value;
  }
  public static Sex from_int (final int value ) {
          switch (value) {
                  case 0: return female;
                  case 1: return male;
          }
   }
}
```

General Inter-ORB Protocol (GIOP)

- Binary Protocol
- Defined by IDL (Of course)
- Efficiently packs data with binary protocol (Look Ma, no delimiters!)
- Latest version of GIOP, 1.3, supports:
  - Bi-directional communication
  - Message chunking (fragmentation)
  - Hooks for security and transactions
  - Connection Management
  - Request Multiplexing

GIOP Header Definition

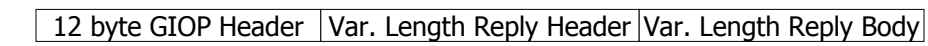



```
module GIOP {
    struct Version {
        octet major;
        octet minor;
    };
    enum MsgType_1_1 {
        Request, Reply, CancelRequest, LocateRequest, LocationReply,
        CloseConnection MessageError, Fragment
    };
    struct MessageHeader_1_1 {
        char            magic[4]; // the string "GIOP"
        Version                 GIOP_version;
        octet           flags; // 1st bit is byte order, 2nd bit is fragment
    indicator
        octet           message_type;
        unsigned long   message_size;
    };
};
```

GIOP Request Message Format


```
module GIOP {
   // ..
   struct RequestHeader_1_1 {
        IOP::ServiceContextList      service_context;
        unsigned long request_id; // id assigned by client
        boolean          response_expected; // oneway responses
        octet                    reserved[3];
        sequence<octet>     object_key;
        string                   operation; // method name
        Principal                requesting_principal;  // deprecated
   };
};
```

GIOP Response Message Format

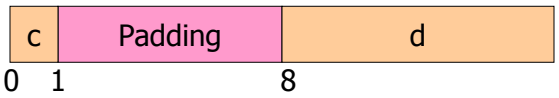



```
module GIOP {
   // ..
   enum ReplyStatusType {
        NO_EXCEPTION, USER_EXCEPTION, SYSTEM_EXCEPTION,
        LOCATION_FORWARD
   };
   struct ReplyHeader {
        IOP::ServiceContextList   service_context;
        unsigned long   request_id; // originally assigned by client
        ReplyStatusType reply_status;
   };
   // ..
};
```

Common Data Representation (CDR)

- Supports both big-endian and little-endian
  - Receiver makes it right
- Aligns primitive types on natural boundaries

```
struct Test {
    char c;
    double d;
 }
```



- Only "in" and "inout" parameters serialized by client
- Only "out", "inout" and return values serialized by server
- Encoded data, except for "Any", is not self-describing for efficiency
- Strings are null terminated, but length is specified too

Internet Inter-ORB Protocol (IIOP)

- Implementation of GIOP for TCP/IP
- Specifies how TCP/IP addressing is encoded in an IOR

```
struct ProfileBody_1_1 {
        Version             iiop_version;
        string              host;
        unsigned short      port;
        sequence <octet>    object_key;
        sequence <IOP::TaggedComponent> components;
    };
```

- Bidirectional communication in current version

Addressing an Object

- IOR - Opaque text string containing all the necessary info to contact:

  IOR:000000000000002b49444c3a6f6d672e6f72672f436f734e616d696e672f4e616d696e67436f6e746578744578743a312e3000000000000020000000000000037000100000000001….

- Accessing a reference to an object from the Naming Server using a URL:

corbaname::futuresparc1:9001#InterfaceRepository

- Accessing an object directly with a URL:

corbaloc:iiop:futuresparc1:9001/InterfaceRepository

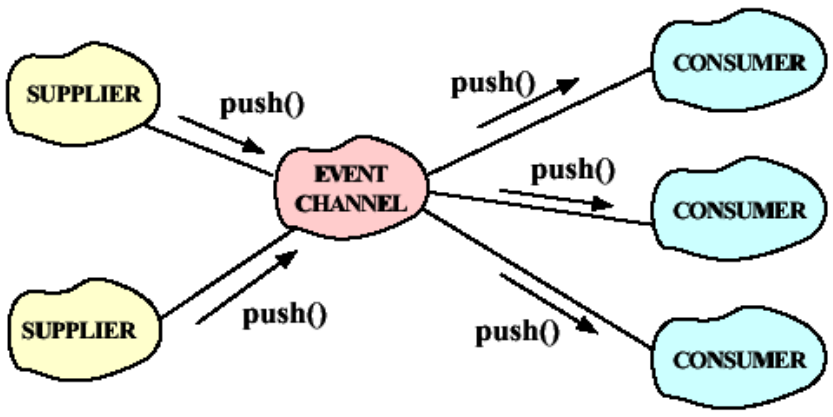
Addressing an Object with Naming

```
orbd -ORBInitialPort 1050 -ORBInitialHost futurelap # start naming server
// begin Java code
Properties props = System.getProperties();
props.put("org.omg.CORBA.ORBInitialPort","1050"); // nameserver port
props.put("org.omg.CORBA.ORBInitialHost","futurelap"); // nameserver location
// initialize CORBA orb
ORB orb = ORB.init(args,props);
String[] services = orb.list_initial_services(); // list bootstrapped services
// get reference to name service
org.omg.CORBA.Object obj = orb.resolve_initial_references("NameService");
NamingContext rootContext = NamingContextExtHelper.narrow(obj);
// add entry for ChatServer
NameComponent[] nc = { new NameComponent("ChatServer","") };
rootContext.rebind(nc,myPOA.servant_to_reference(myServant));
// retrieve the entry using corbaname style url
org.omg.CORBA.Object obj =
    orb.string_to_object("corbaname:iiop:1.2@futurelap:1050#ChatServer");
```

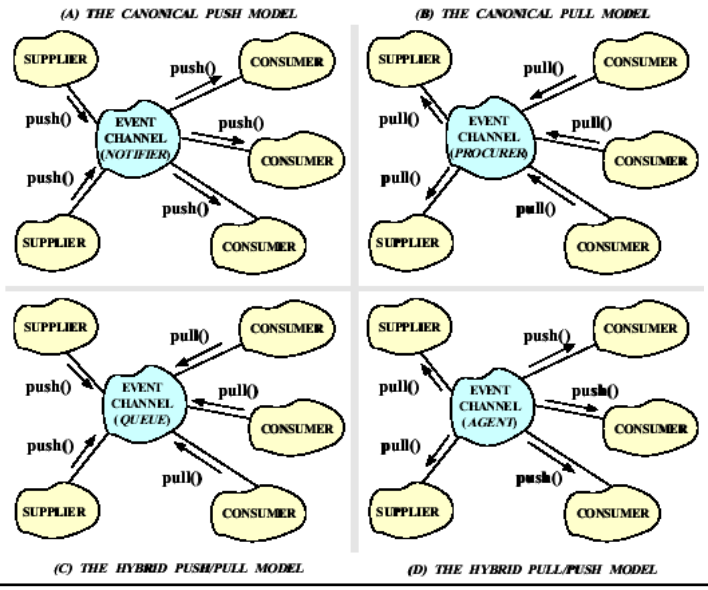
Standardized CORBA Services

- Naming – White pages
- Trader – Yellow pages
- Interface Repository – Interface Descriptions
- Implementation Repository – Bootstrapping servers
- Event – Many to Many event broadcast communication
- Notification – Extension of Event Server with Filtering, Typed Events and Quality of Service (QOS)

Event Service Communication Models



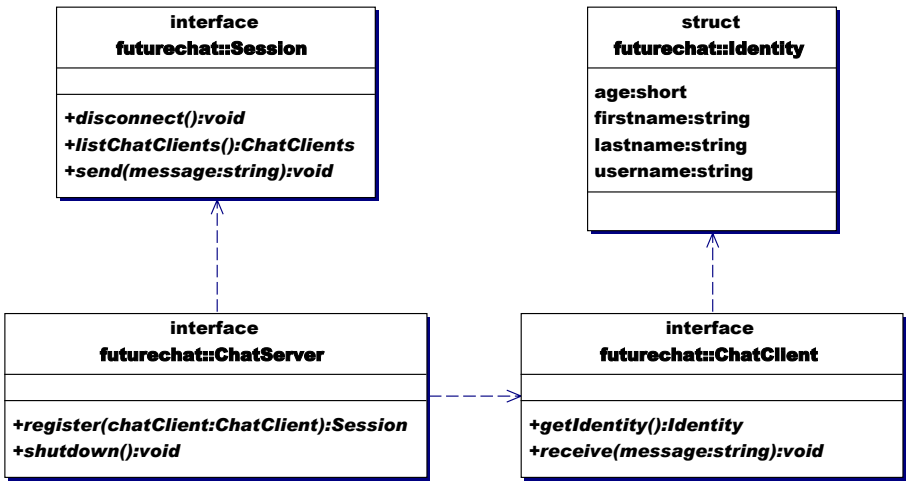
Event Service Variations on a Theme



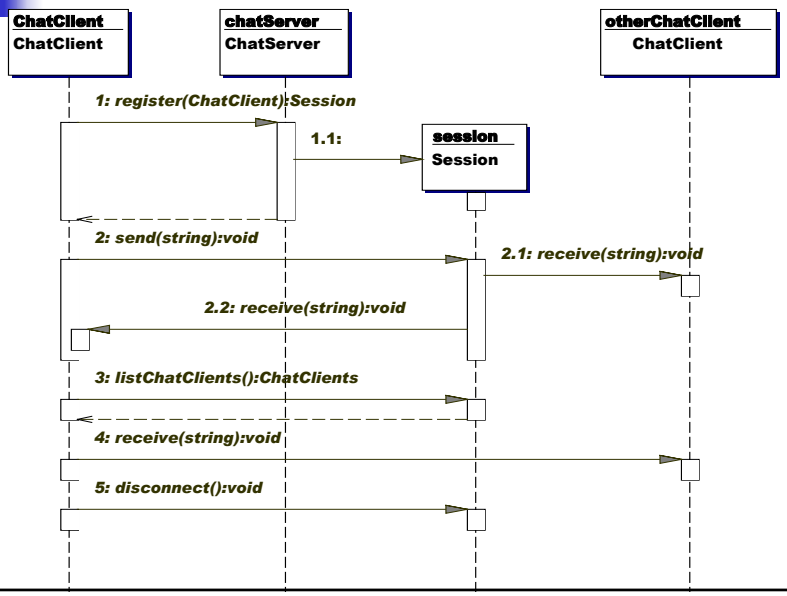
Advanced Topics

- Portable Interceptors
- Dynamic Invocation (DII, DSI, DynAny, IR)
- Security (Authorization, Authentication, Secure Transmission)
- Transactions
- Corba Components Module (CCM)
- Real Time CORBA
- Many other vertical and horizontal services

Class Diagram for Chat



Sequence Diagram for Chat



IDL for Chat Program

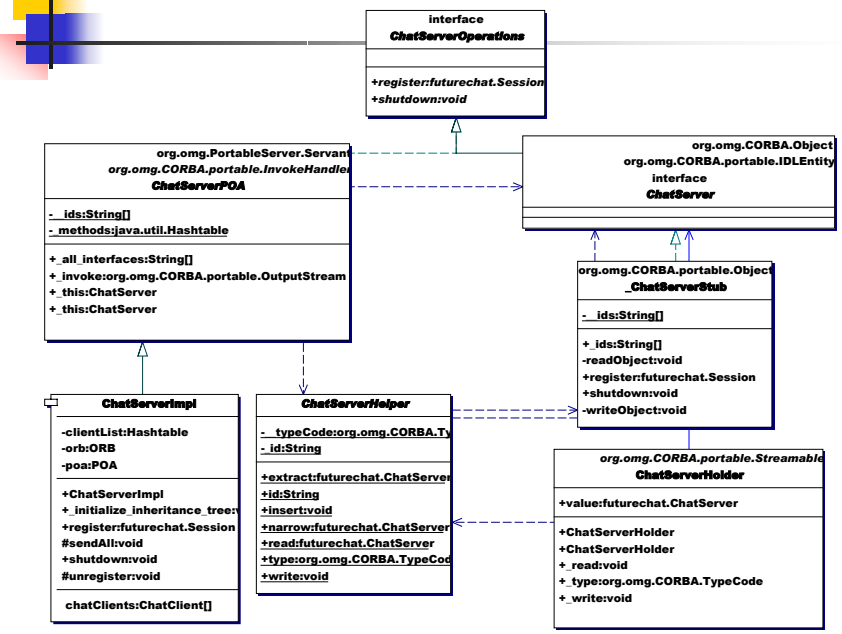
```
module futurechat {

 struct Identity {
   string firstname;
   string lastname;
   string username;
   short age;
 };

 interface ChatClient {
   void receive(in string message);
   Identity getIdentity();
 };
```

```
interface Session {
   oneway void send(in string message);
   sequence<ChatClient> listChatClients();
   oneway void disconnect();
};

interface ChatServer {
    Session register(in ChatClient chatClient);
    oneway void shutdown();
};

};
```

CORBA Mechanics

- idlj –fall chatserver.idl # to generate stubs, skeletons and helper classes.
- Create an implementation of the server and client classes by extending the generated classes
  - (Can use CORBA wizard in NetBeans to automate a lot of this.)
- javac –classpath . *.java */*.java
- orbd -ORBInitialPort 1050 -ORBInitialHost futurelap
- java ChatServerServer
- java ChatClient

Classes Generated by IDLJ



CORBA Client

```
ORB orb = ORB.init(args, null);   // initialize orb
POA poa = (POA)orb.resolve_initial_references("RootPOA");
Policy[] policies;
policies = new Policy[] {};
POA myPOA = poa.create_POA("MyPOA", poa.the_POAManager(), policies);
futurechat.ChatClientImpl myServant = new
futurechat.ChatClientImpl("Rob", "Ratcliff", "rrr6399", (short)40);
byte[] myServantID1 = myPOA.activate_object(myServant);
poa.the_POAManager().activate();
ChatServerCallBackClient handler = new ChatServerCallBackClient();
handler.init(orb);
new Thread(handler).start();
```

Client Code Continued

```
// read location of server from a file
FileReader file = new java.io.FileReader("C:\\ChatServer.ior");
BufferedReader input = new java.io.BufferedReader(file);
String ior = input.readLine(); // read ior
org.omg.CORBA.Object obj = orb.string_to_object(ior); convert to object
 futurechat.ChatServer srv = futurechat.ChatServerHelper.narrow(obj);
 Session session = srv.register(myServant._this()); // register me
  for (int i = 0; i < 10; i++ ) {
      Thread.sleep(1000);
      session.send("Hi for the " + i + " time!");
  }
  session.disconnect(); // disconnect from chat server
  srv.shutdown(); // shutdown chat server
  System.exit(0);
```

Chat Client Callback Code

```
public class ChatClientImpl extends futurechat.ChatClientPOA {
  Identity identity = new Identity();

  public ChatClientImpl (String firstname, String lastname, String username, short
   age) {
      this._initialize_inheritance_tree();
      this.identity.firstname=firstname;
      this.identity.lastname=lastname;
      this.identity.username=username;
      this.identity.age=age;
  }

  public void receive(String message) {
      System.out.println("message received: " + message);
  }

  public Identity getIdentity() {
      return this.identity;
  }
```

CORBA Server Initialization Code

```
ORB orb = ORB.init(args, null);
POA poa = (POA)orb.resolve_initial_references("RootPOA");

Policy[] policies = new Policy[] {
poa.create_id_assignment_policy(IdAssignmentPolicyValue.USER_ID),
poa.create_lifespan_policy(LifespanPolicyValue.TRANSIENT)};
POA myPOA = poa.create_POA("ChatServerPOA",
                           poa.the_POAManager(), policies);

ChatServerImpl myServant = new ChatServerImpl(orb,myPOA);
myPOA.activate_object_with_id("MyServant".getBytes(), myServant);
String ior = orb.object_to_string(myPOA.servant_to_reference(myServant));
FileWriter file = new java.io.FileWriter("C:\\ChatServer.ior");
PrintWriter pfile = new java.io.PrintWriter(file);pfile.println(ior);

poa.the_POAManager().activate();

orb.run();
```

CORBA Server Code

```
package futurechat;
public class ChatServerImpl extends ChatServerPOA {
...
public Session register(ChatClient chatClient) {
      Identity identity = chatClient.getIdentity();
      this.clientList.put(identity.username,chatClient);
      SessionImpl sessionImpl = new SessionImpl(this,chatClient);
      this.poa.activate_object_with_id(identity.username.getBytes(), sessionImpl);
       Session session = SessionHelper.narrow(poa.servant_to_reference(sessionImpl));
      return session;
    }

protected void sendAll(String message) {
     Enumeration enumeration = clientList.elements();
       while(enumeration.hasMoreElements()) {
            ChatClient client = (ChatClient)enumeration.nextElement();
           client.receive(message);
       }
    }
...
```

CORBA Session Code

```
public class SessionImpl extends futurechat.SessionPOA {
    ChatClient client; ChatServerImpl server; Identity identity;

    public SessionImpl(ChatServerImpl server, ChatClient client) {
        this.server = server;
        this.client = client;
        this.identity = client.getIdentity();
    }

    public void send(String message) {
        System.out.println("sending message from " + identity.username);
        server.sendAll(message);
    }

    public futurechat.ChatClient[] listChatClients() {
        return server.getChatClients();
    }

    public void disconnect() {
        server.sendAll(identity.firstname + " " + identity.lastname + " is logging out");
        server.unregister(identity.username);
    }
```

Tcl CORBA Client Code

```
itcl::class ChatClient {
    inherit PortableServer::ServantBase

    public variable firstname "Buford";
    public variable lastname  "Pusser";
    public variable username "TclBoy";
    public variable age "42";
    destructor {
          set poaCurrent [corba::resolve_initial_references POACurrent]
          set poa [$poaCurrent get_POA]
          set oid [$poaCurrent get_object_id]
          $poa deactivate_object $oid
    }

    public method _Interface {} {
          return "IDL:futurechat/ChatClient:1.0";
    }

    public method receive { message } {
          puts "$message"
    }

    public method getIdentity {} {
          set identity [list firstname $firstname lastname $lastname \
                                username $username age $age ]
          return $identity
    }
}
```

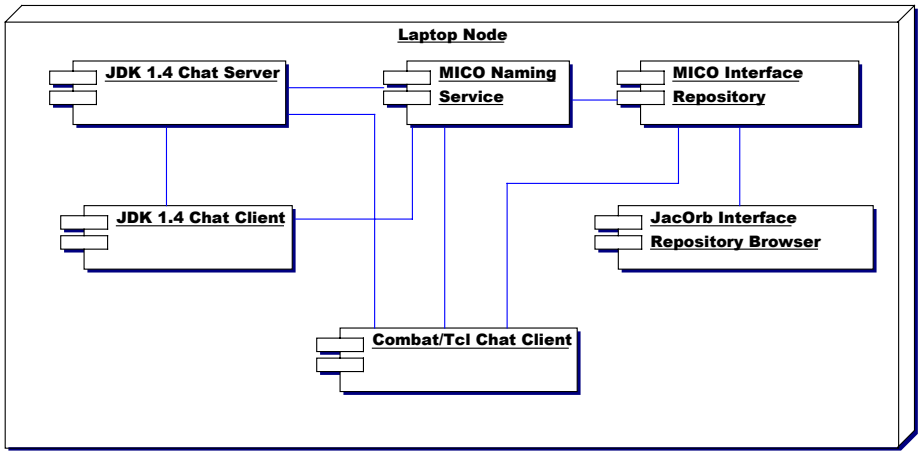
Tcl CORBA Client Code (cont)

```
set hostname [info hostname]
set myargv  [list -ORBInitRef NameService=corbaloc:iiop:${hostname}:1050/NameService  \
    -ORBInitRef InterfaceRepository=corbaloc:iiop:${hostname}:9005/InterfaceRepository]
  eval corba::init $myargv

  set ns [corba::resolve_initial_references NameService]
  $ns _is_a IDL:omg.org/CosNaming/NamingContextExt:1.0
  set ir [corba::resolve_initial_references InterfaceRepository]
  set chatServer [$ns resolve_str ChatServer]
      set poa   [corba::resolve_initial_references RootPOA]
  set mgr   [$poa the_POAManager]
  $mgr activate

  set chatClient [ChatClient #auto]
  set ref [$chatClient _this]
  set session [$chatServer register $ref]
  set clients [$session listChatClients]
  set rob [lindex $clients 0]
  puts "robs stats are: [$rob getIdentity]"
  for { set i 0 } { $i < 10 } { incr i } {
      $rob receive "hi from busser $i"
  }
```

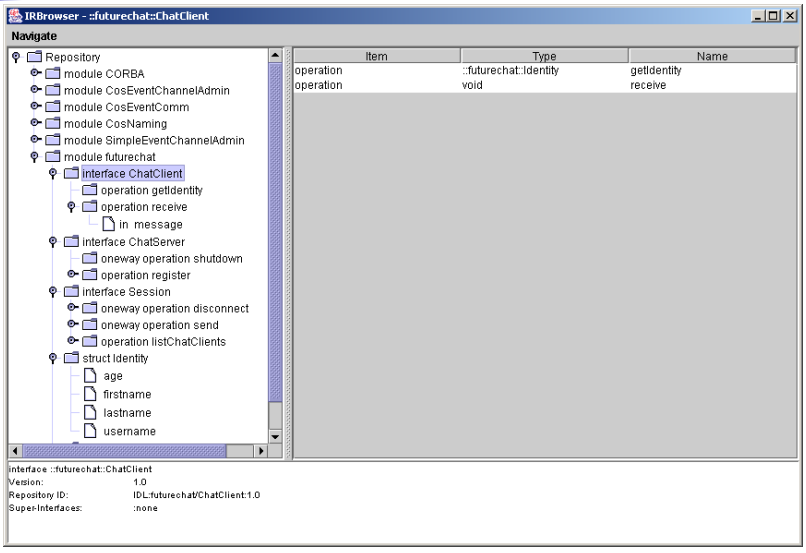
Chat service using Event Service

- The chat messaging could leverage CORBA Event Service
- Two different approaches
  - Purely asynchronous event driven
  - Hybrid approach
    - Control done through RPC mechanisms
    - Messaging done through Event Service

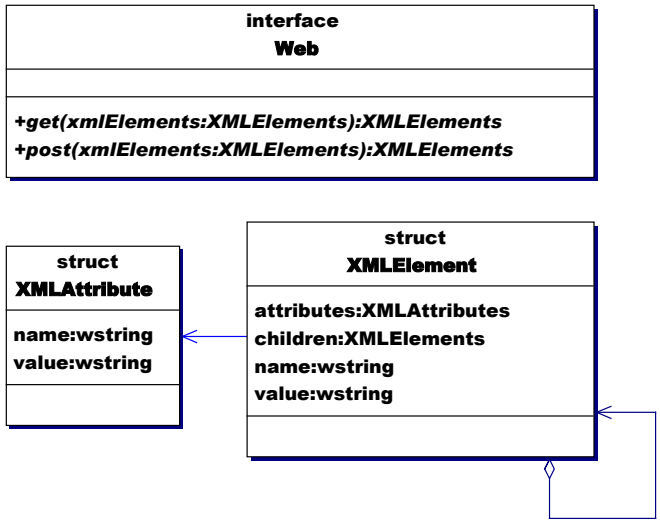
Demo of CorbaChat



JacOrb's Interface Repository Browser Talking to MICO's Repository



Using CORBA and XML



Using CORBA and XML – IDL

```
struct XMLElement;
typedef sequence<XMLElement> XMLElements;

struct XMLAttribute {
    wstring name;
    wstring value;
};
```

```
interface Web {
    XMLElements get(in XMLElements xmlElements);
    XMLElements post(in XMLElements xmlElements);
};
```

```
typedef sequence<XMLAttribute>XMLAttributes;
struct XMLElement {
    wstring name;
    wstring value;
    XMLAttributes attributes;
    XMLElements children;
};
```

Using CORBA and XML – IDL Non-Type Safe Version

Non-type safe interfaces can be easily created with IDL similar to HTTP's GET and POST:

```
interface Web {
    string get(in string xmlString);
    string post(in string xmlString);
};
```

Leave it up to the programmer to create and parse the strings using his favorite XML libraries

Upcoming OMG DOM/Value Mapping

- XML nodes defined by the IDL valuetypes (rather than interfaces, strings or structs)
- Serialized Smart nodes – data and behavior
- No parsing of strings required on the receiving side
- Implementations in the works

Comparison of CORBA and Web Services

	CORBA	Web Services
Type System	IDL - static and runtime checks	XML Schemas - runtime checks only
Marshaling Syntax	CDR - binary	XML - UTF
Connection State	Stateful or Stateless	Stateless
Registry	Interface Repository and Implementation Repository	UDDI/WSDL
Service Discovery	Naming and Trading Services	UDDI
Security	CORBA security service & IIOP over SSL	HTTPS, XML Signature
Firewall Tunneling	IIOP Proxies, Bi-directional IIOP, Spec. being revised	Layered over HTTP, works HTTP proxies too

Comparison of CORBA and Web Services Continued

CORBA Layers	Web Services Layers
IDL	WSDL
CORBA Services	UDDI
CORBA stubs/skeletons and DII/DSI	SOAP messages
CDR binary encoding	XML UTF encoding
IIOP/GIOP	HTTP (SMTP and others)
TCP/IP (and others)	TCP/IP

IDL for StockQuote Service

```
module StockQuoteService {
   interface StockQuotePortType {

        typedef sequence<float> ArrayOfFloat;
        typedef struct TimePeriod {
               wstring startTime;
               wstring endTime;
        };

        ArrayOfFloat GetTradePrices { in wstring tickerSymbol,
        in TimePeriod timePeriod, out float frequency);
   };
};
```

WSDL of StockQuote Service

```
<?xml version="1.0"?>
<definitions name="StockQuote" targetNamespace="http://example.com/stockquote.wsdl"
          xmlns:tns="http://example.com/stockquote.wsdl"
          xmlns:xsd="http://www.w3.org/2001/XMLSchema"
          xmlns:xsd1="http://example.com/stockquote/schema"
          xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
          xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
          xmlns="http://schemas.xmlsoap.org/wsdl/">

    <types>
        <schema targetNamespace="http://example.com/stockquote/schema"
              xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
              xmlns="http://www.w3.org/2001/XMLSchema">
           <complexType name="TimePeriod">
              <all>
                  <element name="startTime" type="xsd:string"/>
                  <element name="endTime" type="xsd:string"/>
              </all>
           </complexType>
           <complexType name="ArrayOfFloat">
              <complexContent>
                  <restriction base="soapenc:Array">
                      <attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:float[]"/>
                  </restriction>
              </complexContent>
           </complexType>
        </schema>
    </types>

    <message name="GetTradePricesInput">
        <part name="tickerSymbol" type="xsd:string"/>
        <part name="timePeriod" type="xsd1:TimePeriod"/>
    </message>

    <message name="GetTradePricesOutput">
        <part name="result" type="xsd1:ArrayOfFloat"/>
        <part name="frequency" type="xsd:float"/>
    </message>

    <portType name="StockQuotePortType">
        <operation name="GetTradePrices" parameterOrder="tickerSymbol timePeriod frequency">
```

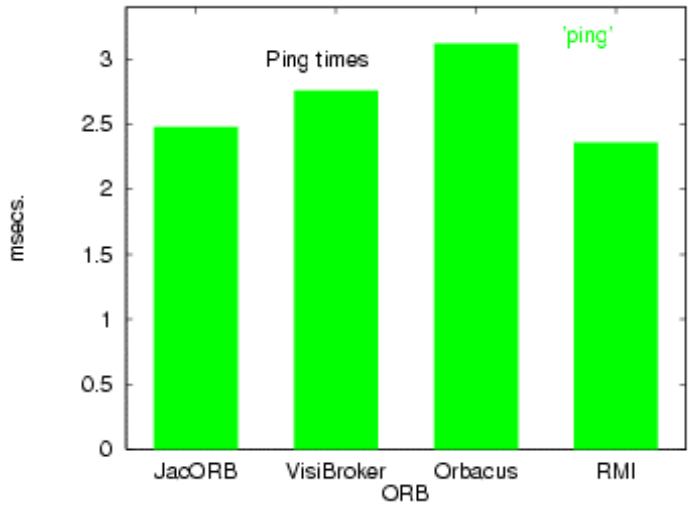
IIOP Performance Comparison with SOAP

- I'm guessing that SOAP over HTTP is 10-100 times slower than IIOP
- It's like bringing a knife to a gun fight!

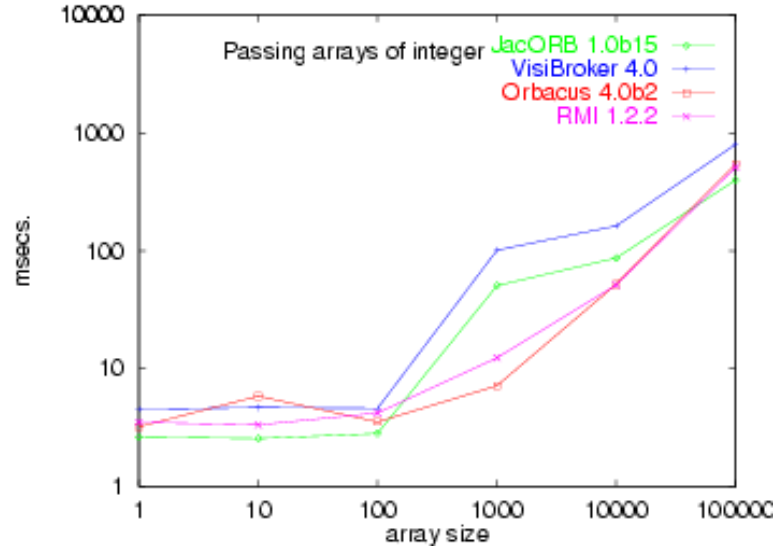
(To do: show actual performance comparisons)

http://www.extreme.indiana.edu/xgws/papers/sc00_paper/node13.html#observations
for SOAP over RMI

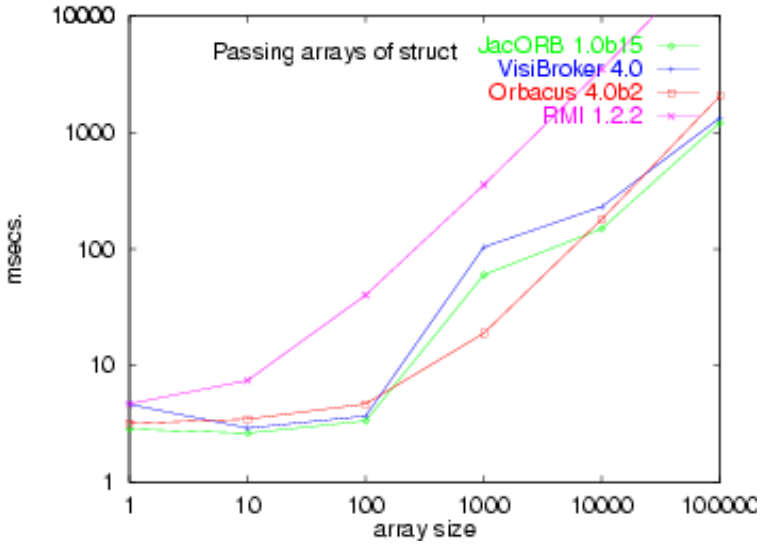
Benchmarks of Selected ORBS



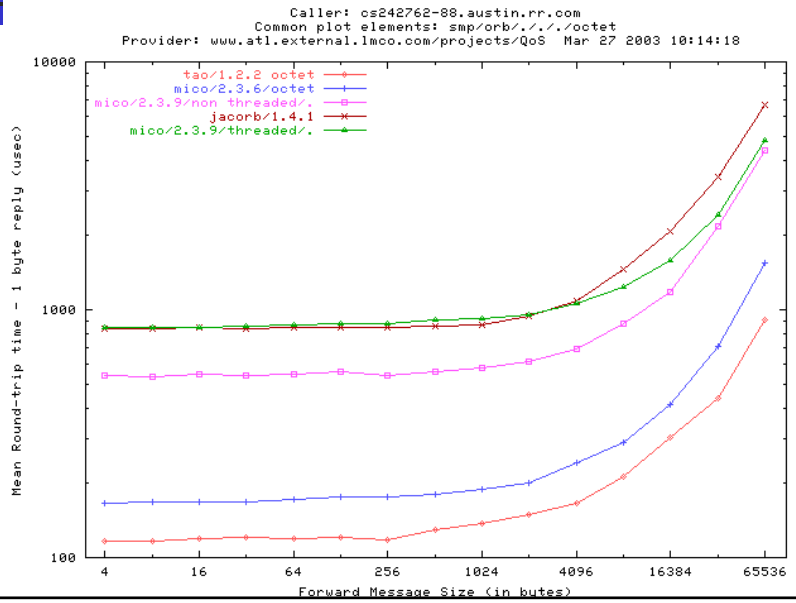
Benchmarks of Selected ORBs



Benchmarks of Selected ORBs



Benchmarks of Selected ORBs



Memorable Quotes

Remember , web services are just large distributed apps with a crude protocol, high latency, and interoperability problems between implementations. Oh, and no callbacks through firewalls.”

- Steve Loughran
Axis User List

WWW = World Wide Wait ... Still!

Are we stuck in a rut?

Swing + CORBA = High Performance, Feature Rich Apps

Conclusion

Maybe this stepchild is just getting started!