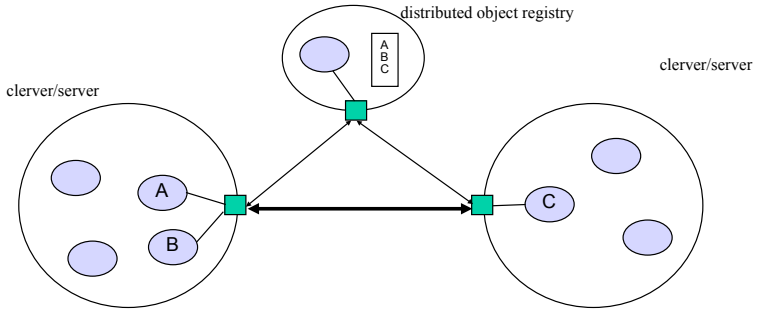
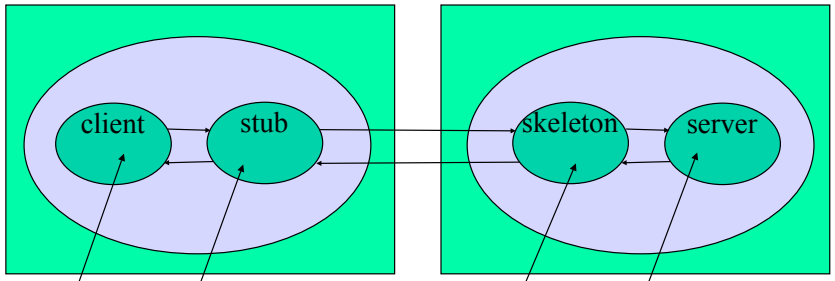
Distributed Objects

Java Remote Method Invocation
Enterprise Java Beans

DO Basic Idea



Marshalling Parameters



Client innocently sends message to proxy

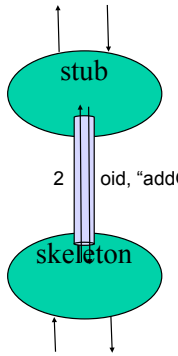
Stub “marshals” parameters

Skeleton “demarshals” parameters

Server code sent message

Marshalling Parameters

```
int result = arithmeticServer.addOne(1)
```




```
public int addOne(int x) { return x+1; }
```

Three Major Standards

- CORBA
  - Common Object Request Broker Architecture
  - Industry sponsored standard
- DCOM
  - Distributed Component Object Model
    - Microsoft
    - from COM from OLE
- Java RMI
  - Remote Method Invocation
- all can be made to be inter-operable

Java RMI Client Code

```
public interface ArithmeticServer extends java.rmi.Remote {
    public int addOne(int i) throws java.rmi.RemoteException;
}


public class ArithmeticClient {
    public static void main(String args[]) throws Exception {
        ArithmeticServer =
        (ArithmeticServer)java.rmi.Naming.lookup(
                                    “rmi://penny.dhcp/ArithmeticServer”);
        System.out.println(as.addOne(1));
    }
}
```


Java RMI Server Code

```
public interface ArithmeticServer extends java.rmi.Remote {
    public int addOne(int i) throws java.rmi.RemoteException;
}

public class ArithmeticServerImpl
extends    java.rmi.server.UnicastRemoteObject
implements ArithmeticServer
{
    public ArithmeticServerImpl() throws java.rmi.RemoteException {
        super();
    }

    public int addOne(int i) { return i+1; }

    public static void main(String[] args) throws Exception {
        java.rmi.Naming.rebind(“ArithmeticServer”,
                               new ArithmeticServerImpl());
    }
}
```


Compilation

```
[CLIENT]
% javac ArithmeticServer.java ArithmeticClient.java


[SERVER]
% javac ArithmeticServer.java ArithmeticServerImpl.java
% rmic -keep ArithmeticServerImpl
% javac ArithmeticServer_Stub.java ArithmeticServer_Skel.java
```

Generated (Client side) Stub Code

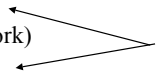
```
public final class ArithmeticServerImpl_Stub
extends RemoteStub
implements ArithmeticServer, Remote
{
    private static final java.rmi.server.Operation[] operations =
{ new java.rmi.server.Operation("int addOne(int)") };
    private static final long interfaceHash = 2100571976616716783L;

    public int addOne(int param_int_1) throws java.rmi.RemoteException {
        java.rmi.server.RemoteCall call =
            super.ref.newCall( (java.rmi.server.RemoteObject) this,
                               operations, 0, interfaceHash);
        java.io.ObjectOutput out = call.getOutputStream();
        out.writeInt(param_int_1);
        super.ref.invoke(call);
        int result;
        java.io.ObjectInput in = call.getInputStream();
        result = in.readInt();
        ref.done(call);
        return result;
    }

}
```


Generated (Server side) Skeleton Code

```
public final class ArithmeticServerImpl_Skel implements java.rmi.server.Skeleton {

    public void dispatch(Remote obj, RemoteCall call, int opnum, long hash) {
        if (hash != interfaceHash)
            throw new SkeletonMismatchException("interface hash mismatch");

        ArithmeticServerImpl server = (ArithmeticServerImpl) obj;
        switch (opnum) {
            case 0: // addOne(int)
            {
                int param_int_1;
                java.io.ObjectInput in = call.getInputStream();
                param_int_1 = in.readInt();
                call.releaseInputStream();
                int $result = server.addOne(param_int_1);

                java.io.ObjectOutput out = call.getResultStream(true);
                out.writeInt($result);

                break;
            }

            default: throw new UnmarshalException("invalid method number");
        }
    }
```


Performance

- Latency: arithmeticServer.addOne(1);
  - Local method calls
    - .07 usec
  - Remote method call (same machine)
    - 656 usec
  - Remote method call (network)
    - 2000 usec



i.e. Pretty Bad

- DB access
  - 1600 usec

Software architecture implications

- We see that we can deploy objects in different address spaces and connect them up into distributed applications.
  - performance problems notwithstanding
- Does this open the door to a new style of corporate software environment?
  - Coarse grain objects communicating via RPC sometimes deployed on the same machine other times not.
- We will explore this further by very briefly illustrating one approach to components and one approach to combining components that takes transactional integrity into account:
  1. JavaBeans
  2. Enterprise JavaBeans or EJB

Java Beans

http://developer.java.sun.com/developer/onlineTraining/Beans/bean01/page2.html

> Introspection, the process by which a builder tool analyzes how a Bean works, differentiates Beans from typical Java classes. Because Beans are coded with predefined patterns for their method signatures and class definitions, tools that recognize these patterns can "look inside" a Bean and determine its properties and behavior

- For instance, toy BeanBox application interprets pairs of get/set accessor methods as a bean “property”.

Introspection and Design Time

- In order to be able to reuse software components we would like to build tools that are able to combine components without modifying any source.
- BeanBox demonstrates this is possible that with reflection and simple coding conventions.
- BeanBox is a toy builder but at one time was an important reference implementation of Bean/tool interaction.
- Suppose we create a trivial Bean that has a property called csc407color

csc407Color property

```
import java.awt.*;

import java.io.Serializable;
public class Csc407Bean extends Canvas
implements Serializable {
    private Color csc407color = Color.green;
    //getter method
    public Color getCsc407Color() {
        return csc407color;
    }
    //setter method
    public void setCsc407Color(Color newColor) {
        this.csc407color = newColor;
        repaint();
    }
    //override paint method
    public void paint (Graphics g) {
        g.setColor(getCsc407Color());
        g.fillRect(20,5,20,30);
    }
    //Constructor: sets inherited properties
    public Csc407Bean() {
        setSize(60,40);
        setBackground(Color.red);
    }
}
```

Is that all there is?

Enterprise Java Beans

http://java.sun.com/j2ee/1.4/docs/tutorial/doc/Overview7.html#wp79950

which goes on at length like so:

> An Enterprise JavaBeans (EJB) component or enterprise bean is a body of code with fields and methods to implement modules of business logic. You can think of an enterprise bean as a building block that can be used alone or with other enterprise beans to execute business logic on the J2EE server.

- Beans started out as simple coding conventions defining properties
- Moved on to “beanbox” bean lifecycle maintenance. Design time focus.
- Was given steroids (in preparation for Olympics??) and turned into an infrastructure to manage persistent object oriented data with a view to maintaining transactional integrity.
- Transparent distribution.
- For EJB 2.0 emphasis less on distribution more on persistence.

Enterprise Java Beans

- Component Object Model
  - Distributed
  - Persistent
  - Secure
  - Transactional
    - ACID
      - Atomicity: all or none
      - Consistency: database will always be in a consistent state
      - Isolation: intermediate state not visible until completed
      - Durability: when completed, the changes are stored permanently
- EJBs are a standard
  - allows application developers to write simple, standard code
  - allows implementers to get all the underlying stuff done well

EJB and persistence

- Object persistence in general beyond our scope.
- However, suppose each type of Entity Bean can serialize (think write) itself as some stream (now-a-days XML).
  - also deserialize, or read.
  - or read and write themselves to a RDBMS.
- Code on following shows an old fashioned way of serialization objects. It will serve to illustrate the point

Serialization code

```
import java.io.ObjectOutputStream;
import java.io.IOException;
import java.awt.Color;

class SerializeDemo {
    public static void main(String[] args){
        Csc407Bean b = new Csc407Bean();

        try{
            new ObjectOutputStream(System.out).writeObject(b);
        }catch(IOException ioe){
            System.err.println("oops, exception " + ioe);
        }
    }
}
```

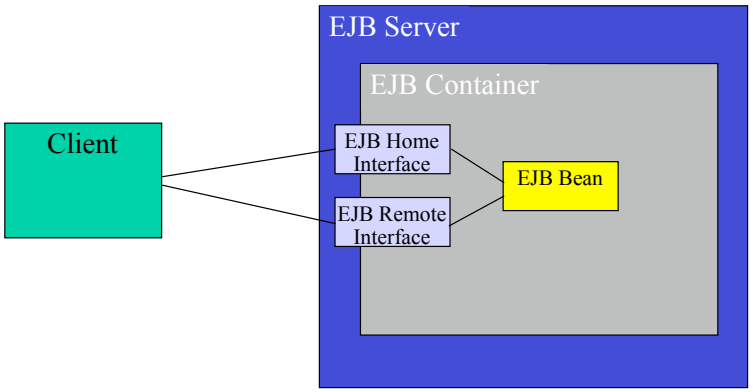

Deserialization Code

```
import java.io.ObjectInputStream;
import java.awt.Frame;

class DeserializeDemo {
    public static void main(String[] args){
        try{
            Csc407Bean b = (Csc407Bean)
                    new ObjectInputStream(System.in).readObject();
            Frame f = new Frame();
            f.add(b);
            f.setSize(100,100);
            f.setVisible(true);
            //here do window things to display the canvas..
        }catch(Exception ioe){
            System.err.println("oops, exception " + ioe);
        }
    }
}
```

EJB and persistence

- A EJB container can "wake up" entity EJB's when they are required, send messages to them, then put them back to sleep..
- These and many other factors promote a vision of enterprise computing whereby a EJB container manages purchased EJB's cooperating with those developed in-house to construct (potentially distributed) applications.
- Transactional issues are managed by the container by setting "transactional properties" of bean methods.
- Programmers are freed from much ugly housekeeping
  - and currently performance leaves much to be desired..

EJB Architecture



Types of EJBs

- Two types of beans:

– Session bean

- encapsulates transactional operations
- stateful/stateless

– Entity bean

- encapsulates persistent state
- container-managed persistence / bean-managed persistence

EJBs

- Remote Interface

```
public interface GroceryOrder extends javax.ejb.EJBObject {
    public Date getDate() throws RemoteException;
    public void setDate() throws RemoteException;
    …
}
```

- "Home" Interface
  - Sort of like static methods
  - Note the factory method and a lookup method

```
public interface GroceryOrderHome extends javax.ejb.EJBHome {
    public GroceryOrder create(int id)
        throws CreateException, RemoteException;
    public GroceryOrder findByPrimaryKey(GroceryOrderPK pk)
        throws FinderException, RemoteException;
}
```

EJB Implementation Class

```
public class GroceryOrderBean implements javax.ejb.EntityBean {
    public int id;
    public Date date;

    public void ejbCreate(int id) { this.id = id; }
    public Date getDate() { return date; }
    public void setDate(Date date) { this.date = date; }

    public void setEntityContext(EntityContext ctx) { }
    public void unsetEntityContext() { }

    //container calls to instruct Bean to do things
    public void ejbActivate() { }
    public void ejbPassivate() { }
    public void ejbLoad() { }
    public void ejbStore() { }
}
```


Session Beans

```
public class ShopperBean implement javax.ejb.SessionBean {
    public Customer customer;
    public GorceryOrder order;

    public void ejbCreate(Customer cust) { customer = cust; }

    public Receipt processOrder(CreditCard card)
        throws RemoteException,
        IncompleteConversationalState,
        BadCredit
    {
        if(customer==null||order==null) throw new IncompleteConversationalState();

        ProcessOrderHome poh = (ProcessOrderHome)getHome("ProcessOrderHome");
        ProcessOrder po = poh.create(customer, order);

        ProcessPaymentHome pph = (ProcessPaymentHome)getHome("ProcessPaymentHome");
        ProcessPayment pp = ppHome.create();

        pp.byCreditCard(customer, card, order.price());
        po.process();

        Receipt r = new Receipt(customer, order, card);
        return r;
    }
```


EJB Summary

- Transparent
  - Distribution
    - ejb can be anywhere
  - Replication & Load-Balancing
    - ejb can be moved around
    - ejb can be replicated (e.g., Toronto – London)
  - Resource Management
    - ejb shells can be reused
    - persistent data can be cached
  - Persistence Management
    - ejb automatically mapped to persistent storage
  - Transaction Management
    - session beans mapped to transactional system
  - Security
    - Identities, roles, access control lists

EJB Implementations

- Still pretty flaky and none support everything on the previous list.
  - WebLogic
  - EJBHome
  - SapphireWeb
  - BEA
  - Gemstone
  - IBM CICS/EJB, ComponentBroker, WebSphere
  - NetDynamics
  - Oracle Application Server

  …