JMS & Message-Driven Beans

The Enterprise Gets the Message

William F. Field

Principal Consultant

wff@bea.com

Agenda

- **Introduction**
  - Distributed Computing Historical Context
    - Raw Sockets → RPCs → CORBA → J2EE
  - MOM
    - Scalability: Socket MUX
    - Flexibility: Sync. vs. Async. Messages
- **J2EE, EJB, & JMS**
  - EJB 1.1 & JMS
  - Example
  - EJB 2.0 & Message-Driven Beans
  - Example
  - WebLogic Server 6.0

Introduction

- **Distributed Computing Historical Context**
  - Raw Sockets

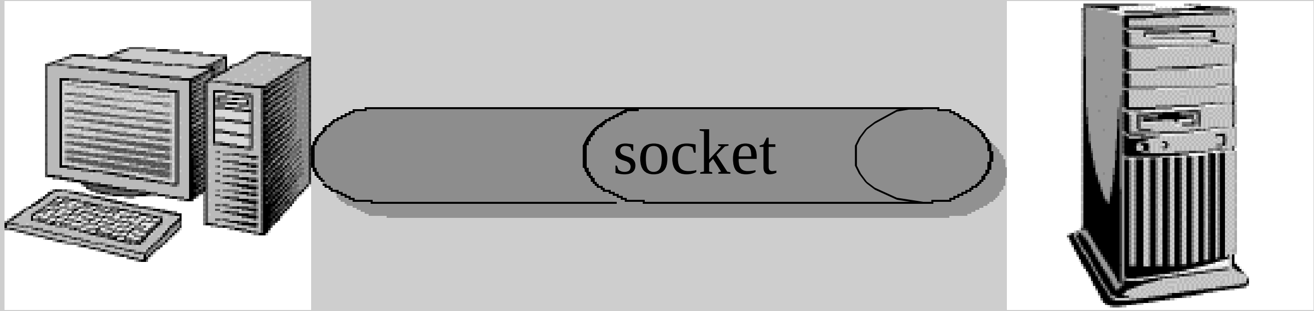
  - RPC - Remote Procedure Calls
  - CORBA - Common Object Request Broker Architecture
  - J2EE - Java 2 platform, Enterprise Edition

  - MOM - Message Oriented Middleware

Distributed Computing History

Raw Sockets

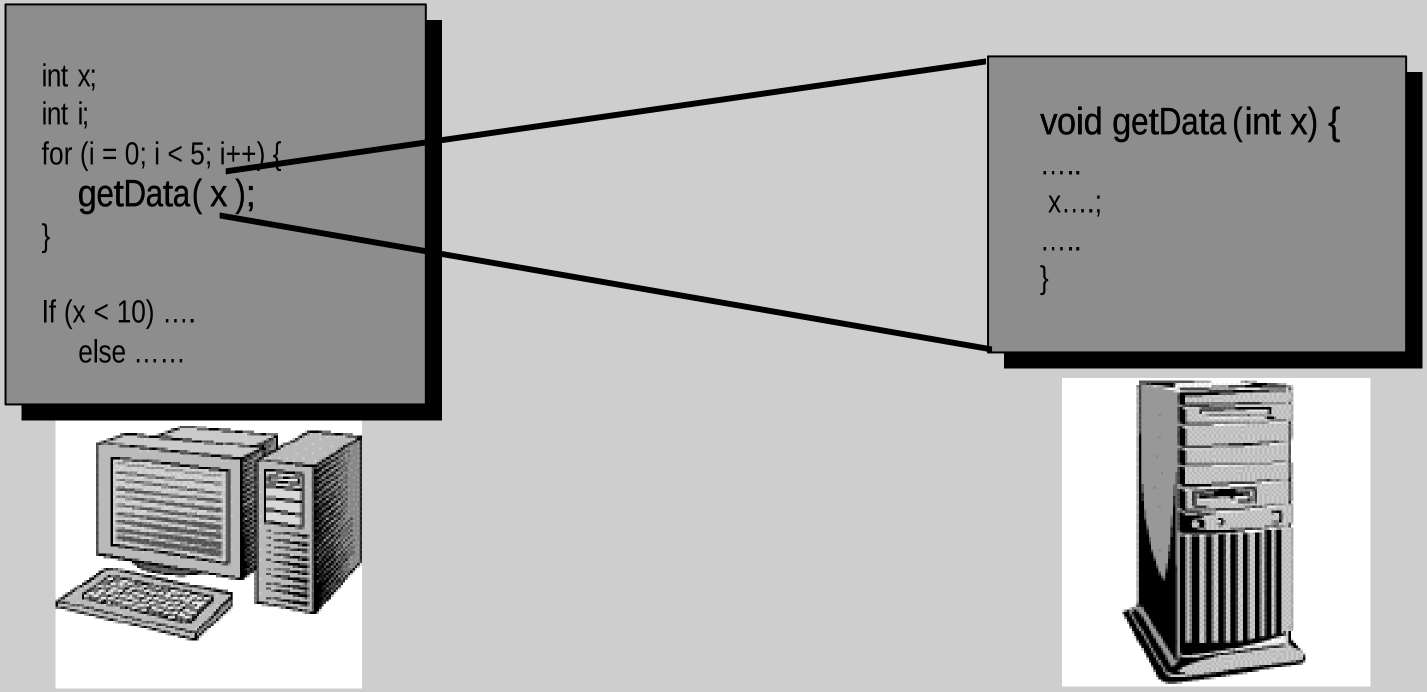
- Difficult, tedious programming requirements
- Not intuitive, It is too Low-Level
- We need higher-level abstractions / concepts



Distributed Computing History…

Distributed Computing Architectures

- RPC - Remote Procedure Calls



```
int x;
int i;
for (i = 0; i < 5; i++) {
    getData( x );
}

If (x < 10) ….
    else ……
```

```
void getData (int x) {
…..
 x….;
…..
}
```

Distributed Computing History… RPC Disadvantages

- **All RPC messages are synchronous**
- **Lack of control of remote services: startup, shutdown**
  - No concept of lifecycle control
- **Only a few default services provided:**

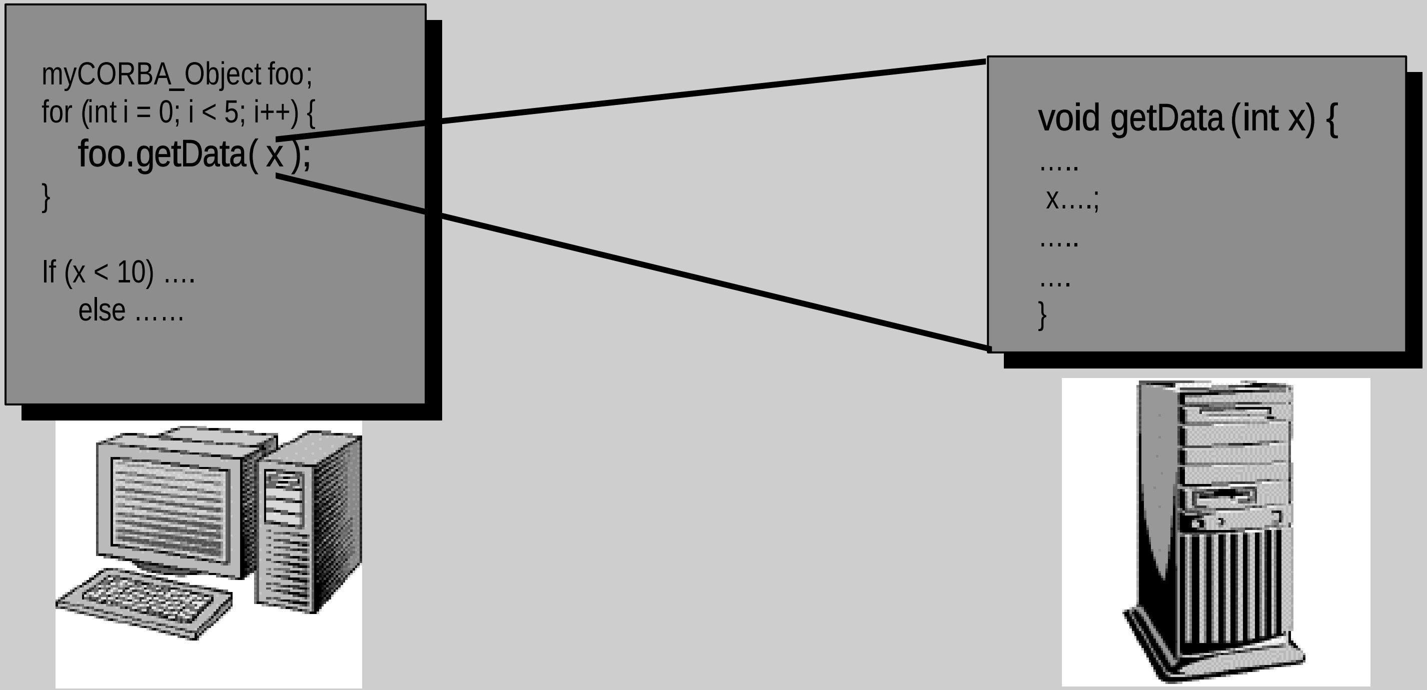
  For Example, DCE-RPC provides

  - Naming/Directory Service
  - Time Service
  - Security Service
  - Transaction Service

Distributed Computing History…

Distributed Computing Architectures

- CORBA - Common Object Request Broker Architecture



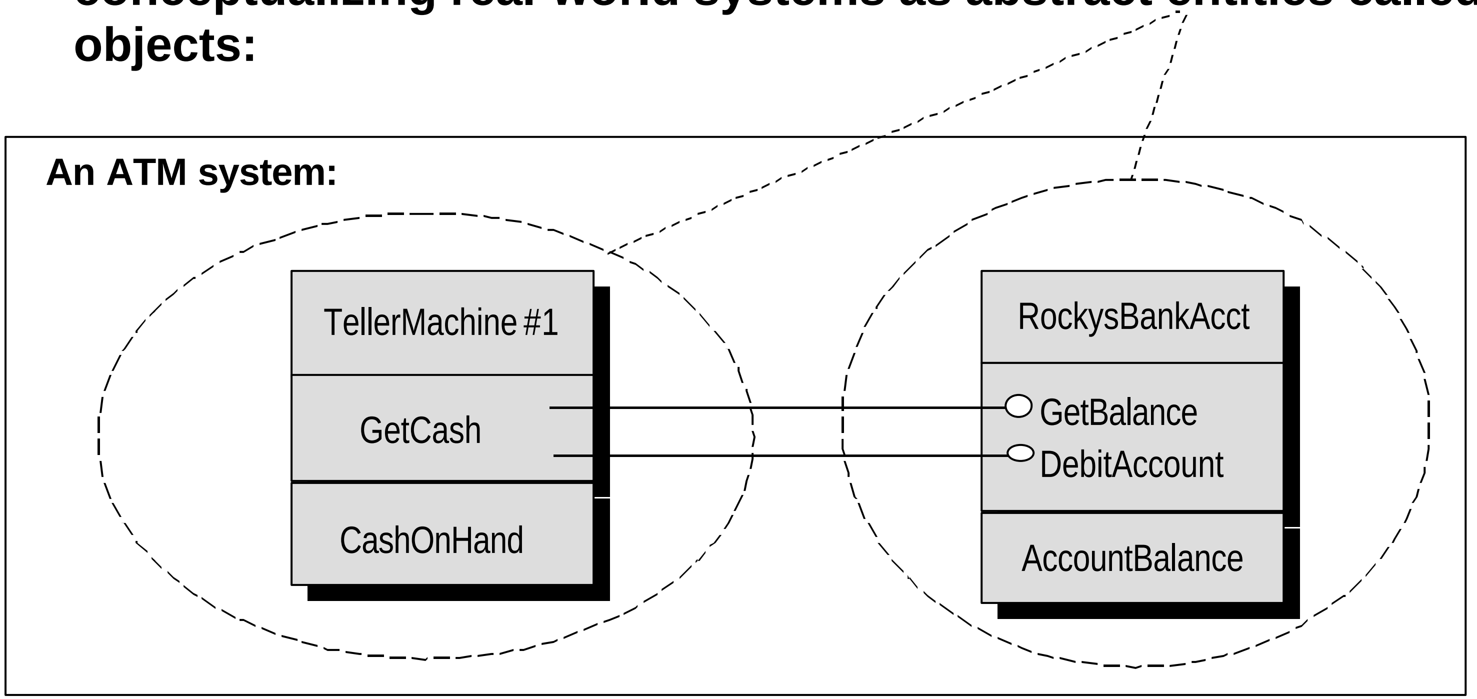
```
myCORBA_Object foo;
for (int i = 0; i < 5; i++) {
    foo.getData( x );
}

If (x < 10) ….
    else ……
```

```
void getData (int x) {
…..
 x….;
…..
….
}
```

Distributed Computing History…

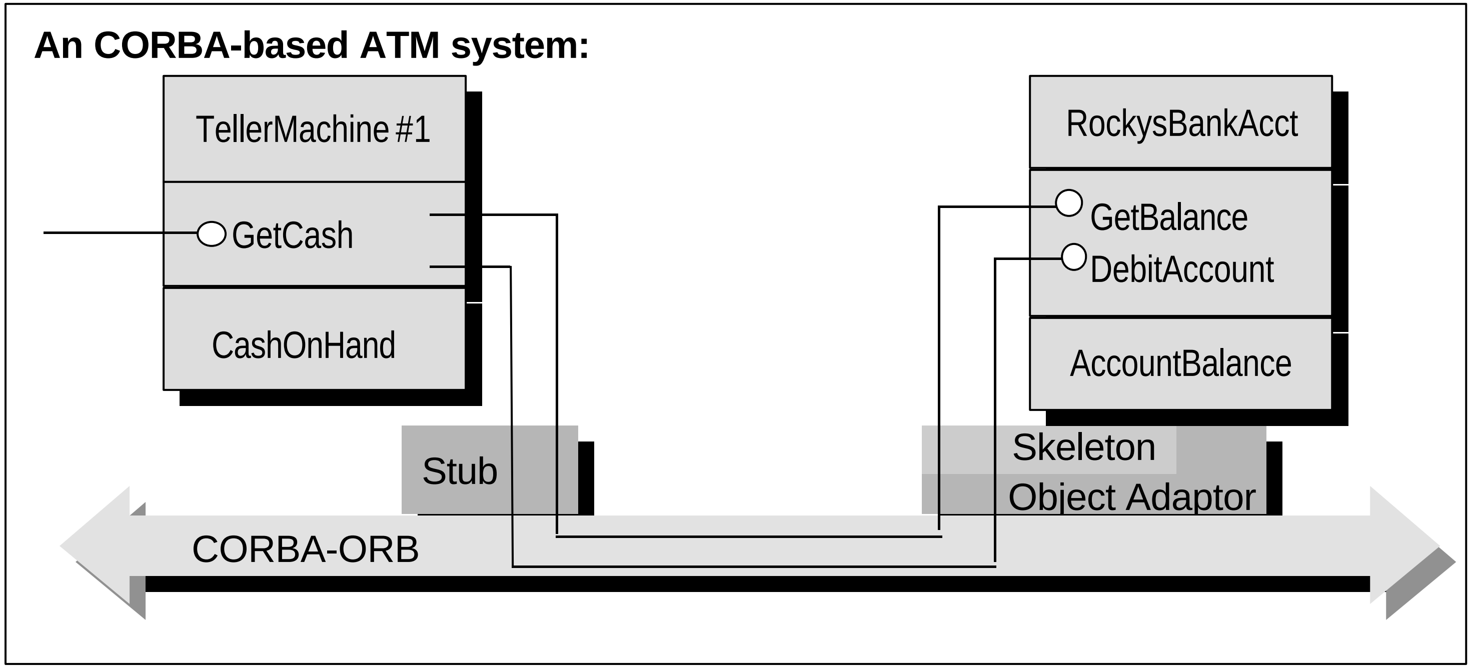
Objects (O-O): An abstraction for modeling the real-world.

Basically, the allure of O-O centers around our intuition for conceptualizing real-world systems as abstract entities called objects:



Distributed Computing History…

CORBA technology provides the infrastructure to enable these objects to do their work transparently in a distributed system:



Distributed Computing History…

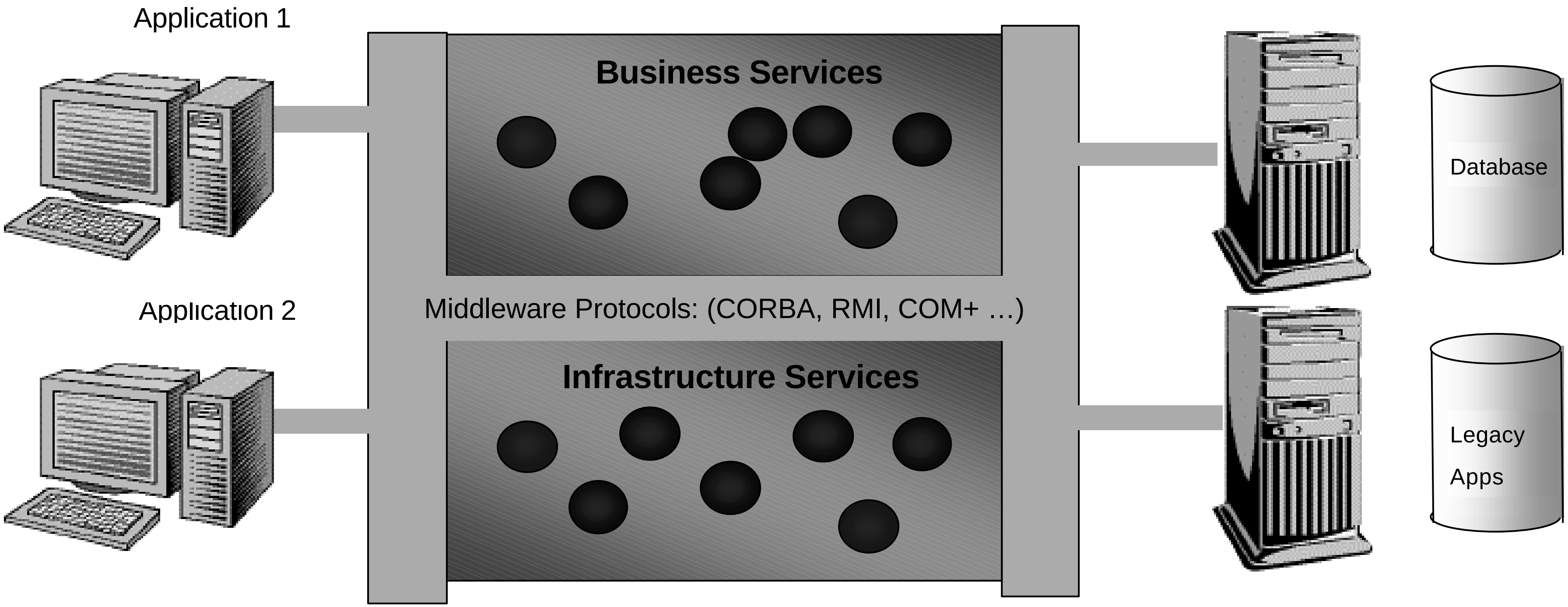
CORBA Advantages/Disadvantages

- **Advantages**
  - Complete framework of services
  - Automatic control of Object Lifecycle
  - Language/OS independent

- **Disadvantages**
  - Again, all CORBA messages are synchronous
  - Over-reliance on inheritance: inflexible implementation
  - Use of a IDL (Interface Definition Language) adds complexity

Distributed Computing History…

Distributed Object Architectures

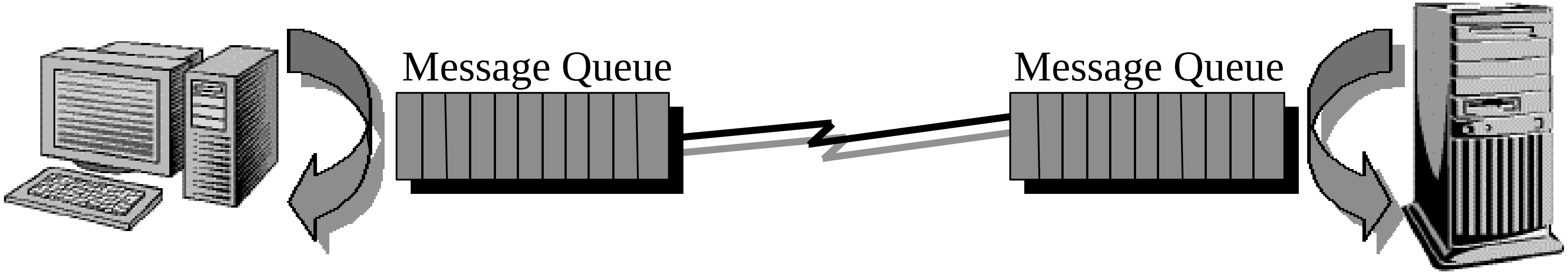


- **Middle Tier Services implement the business rules and processes**
- **Middle Tier as an abstraction layer**
  - technology details are encapsulated
  - focuses design effort on the domain

- **Adding a Middle Tier enables us to de-couple our design**
- **Scalability/Availability is enhanced**
  - we can transparently add features:
    - multiple hosts, concurrency, etc.

Distributed Computing History…
Message Oriented Middleware

Meanwhile… a more pragmatic focus led to...



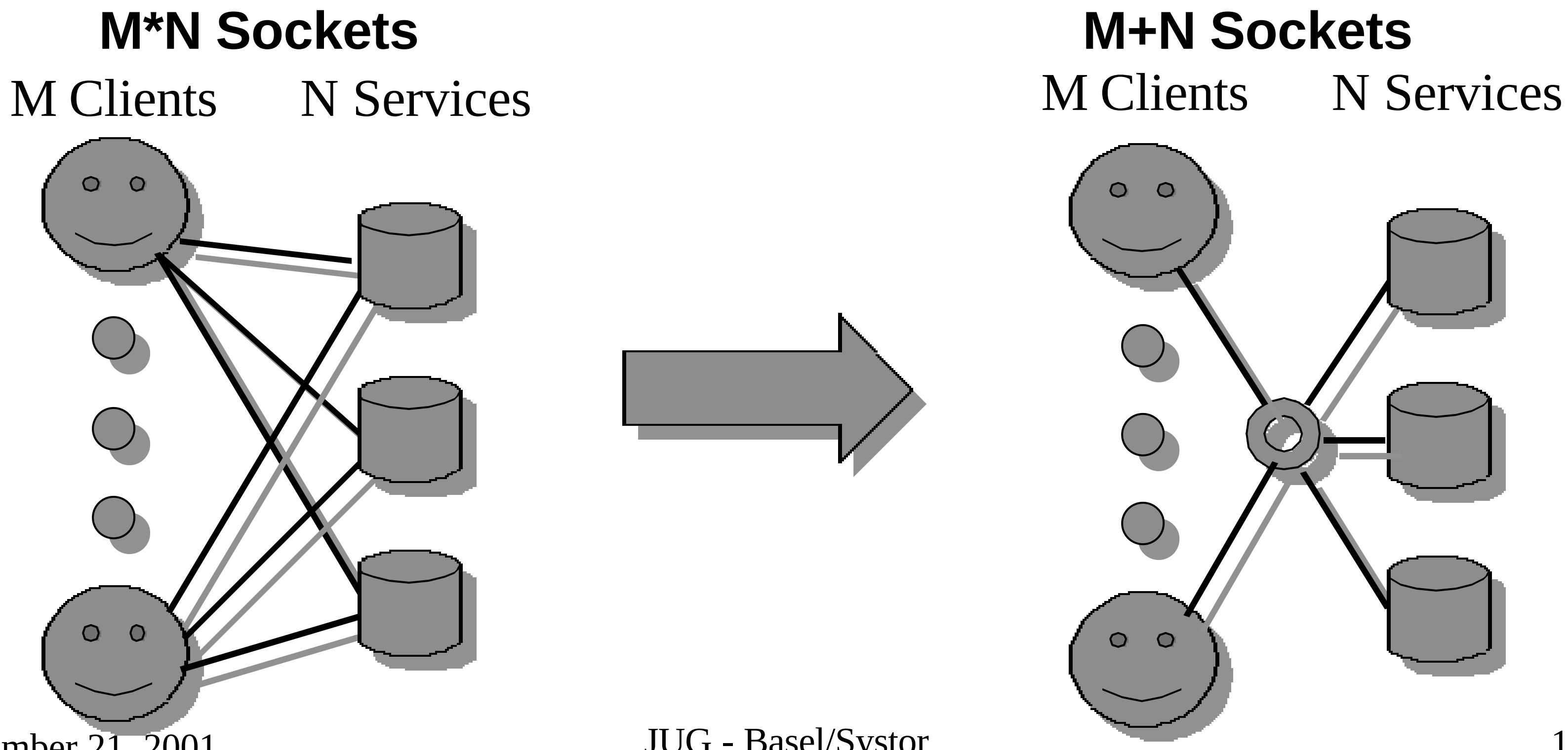
MOM – Message Oriented Middleware:

- **Messages as the unit of distribution**
- **Architecture Scalability**
  - **Socket MUX - (N+M sockets vs. N*M sockets)**
- **Architecture Flexibility**
  - **Synchronous vs. Asynchronous Messages**

Distributed Computing History…
Message Oriented Middleware

Scalable Architecture:

- Simplistic RPC or CORBA frameworks: M*N sockets
- MOM frameworks: M+N sockets



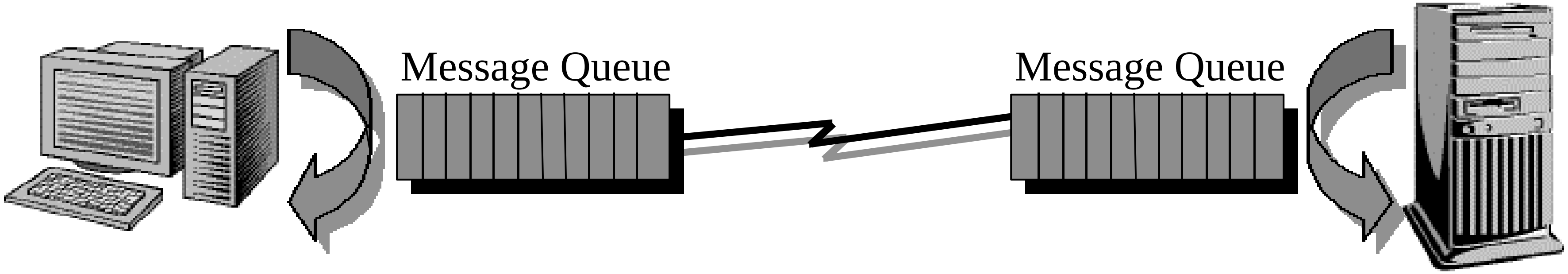
M*N Sockets

M Clients N Services

M+N Sockets

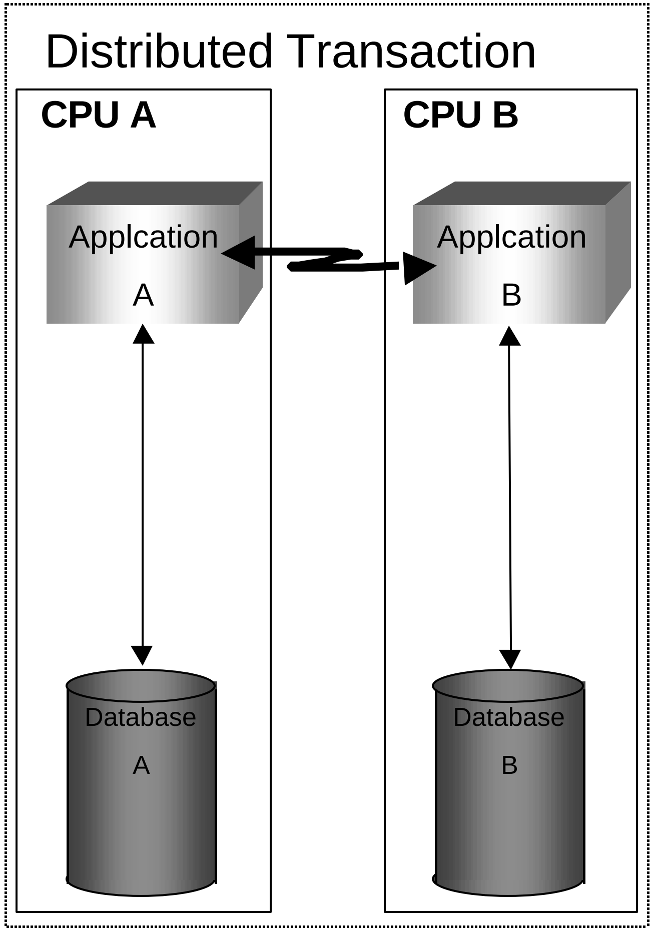
M Clients N Services

Distributed Computing History…
Message Oriented Middleware



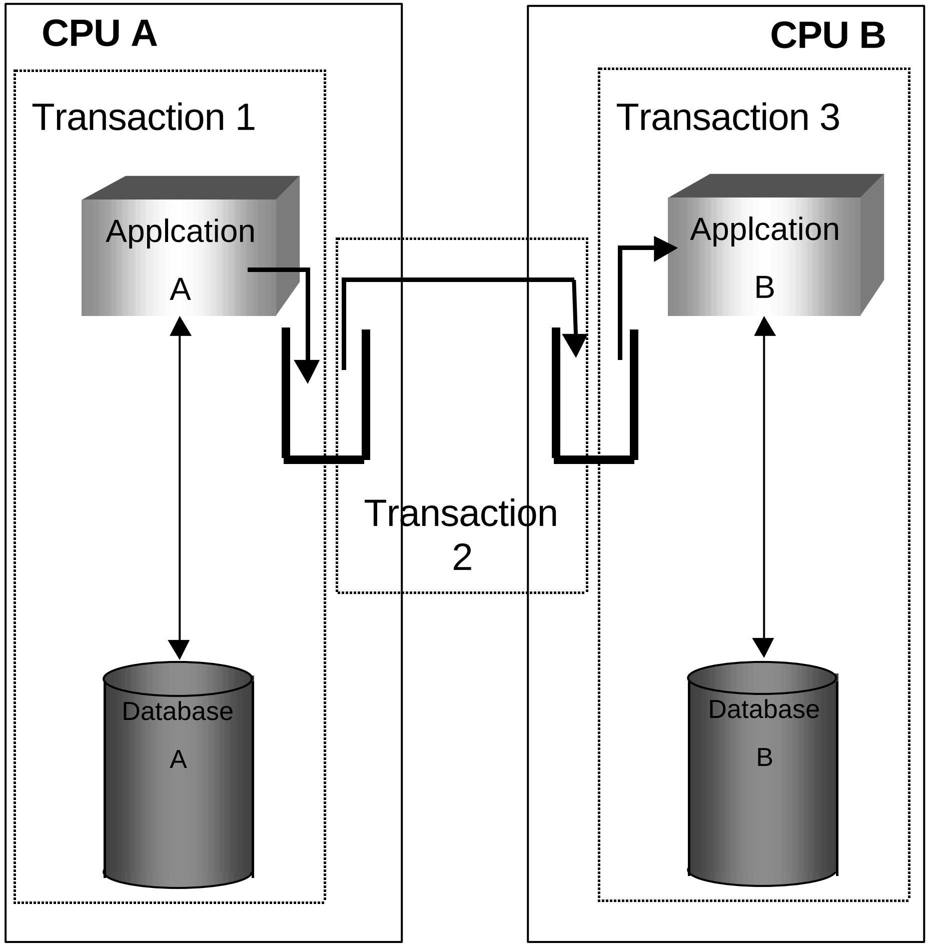
- **Summary: Synchronous vs. Asynchronous Messages**
  - Most Distributed OLTP (e.g., EJB containers/RMI) require synchronous messaging
  - Synchronous messaging increases client latency
  - Synchronous messaging does not scale well
  - Synchronous messaging tightly couples the client to the server
    - E.g.: Legacy batch systems are difficult to couple with On-Line Transaction Processing (OLTP) systems unless you use asynchronous messaging
  - J2EE **JMS** provides asynchronous messaging to overcome these problems

MOM & Transactions



Distributed Transaction

Transactional Messaging



Distributed Computing History…

CORBA vs. Java/JPE

Historically, then:

CORBA and Java have developed from a similar communications model:

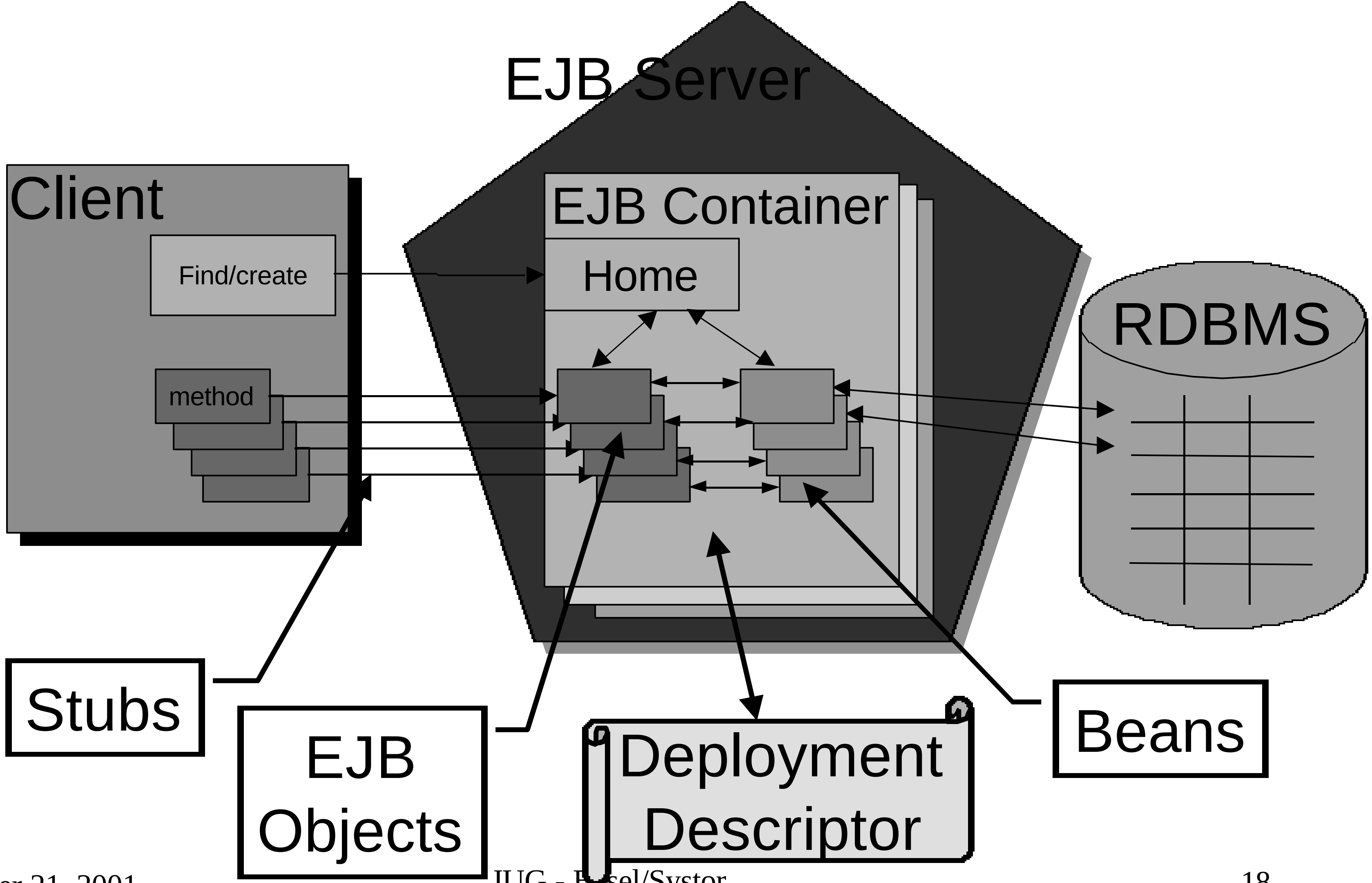
- **Objects are the unit of distribution**
- **Messages are sent synchronously**

- **Both have components or *beans*:**
  - More flexible than pure inheritance: uses delegation/aggregation
  - J2EE these components are EJB - Enterprise Java Beans

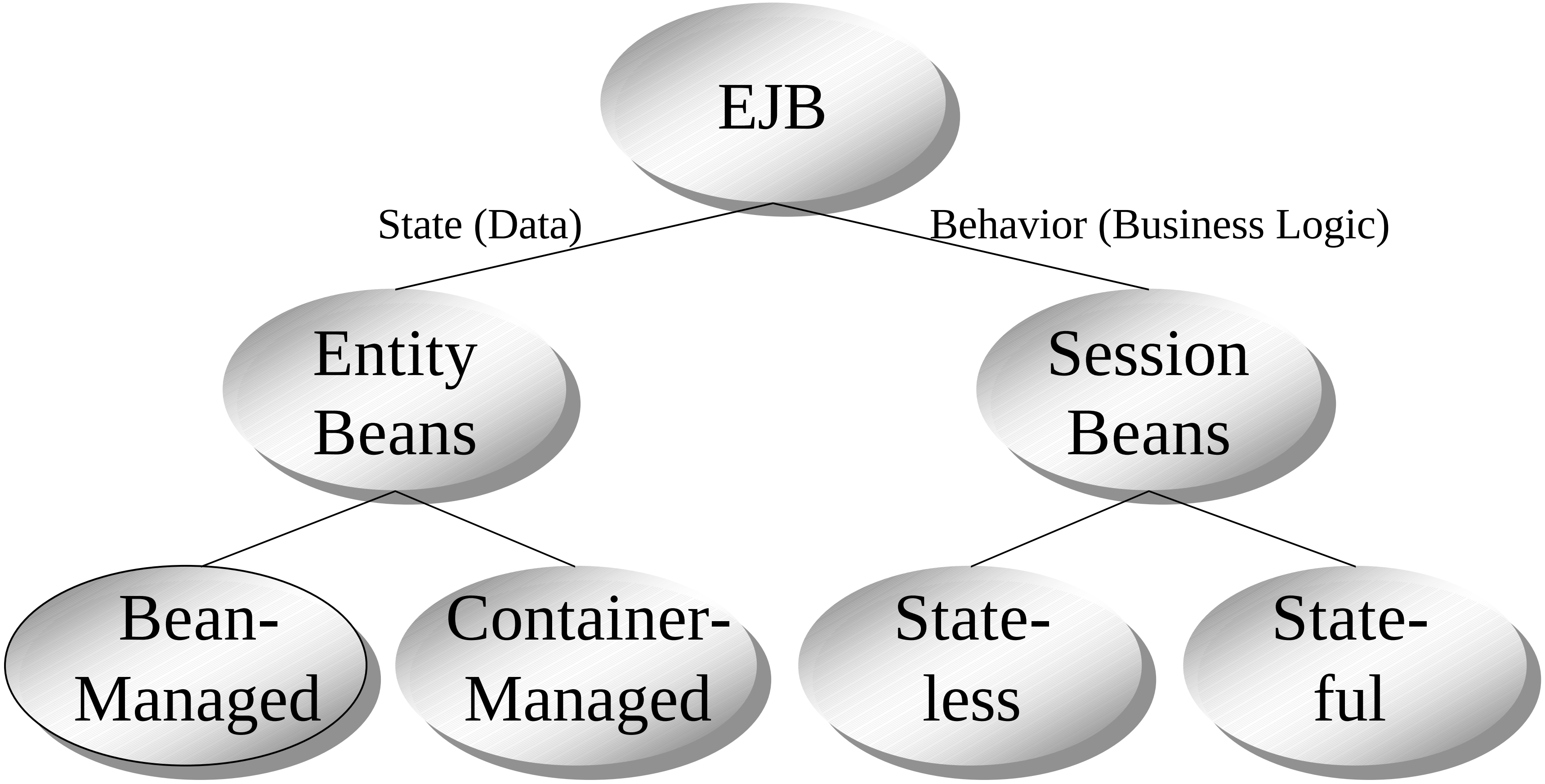
J2EE, EJB, & JMS

- **Java 2 Platform Enterprise Edition**
  - Component architecture:
    - Container: Framework
    - Beans: Components that use container services – Plug ‘N Work
- **EJBObject - wrapper for remote access**
  - Specifies the remote interface implemented by the Bean
- **Bean - business logic**
  - Implements EntityBean or SessionBean interface
- **Home - factory and finder**
  - Implements Home interface
- **Deployment Descriptor**
  - XML file that describes the bean’s properties

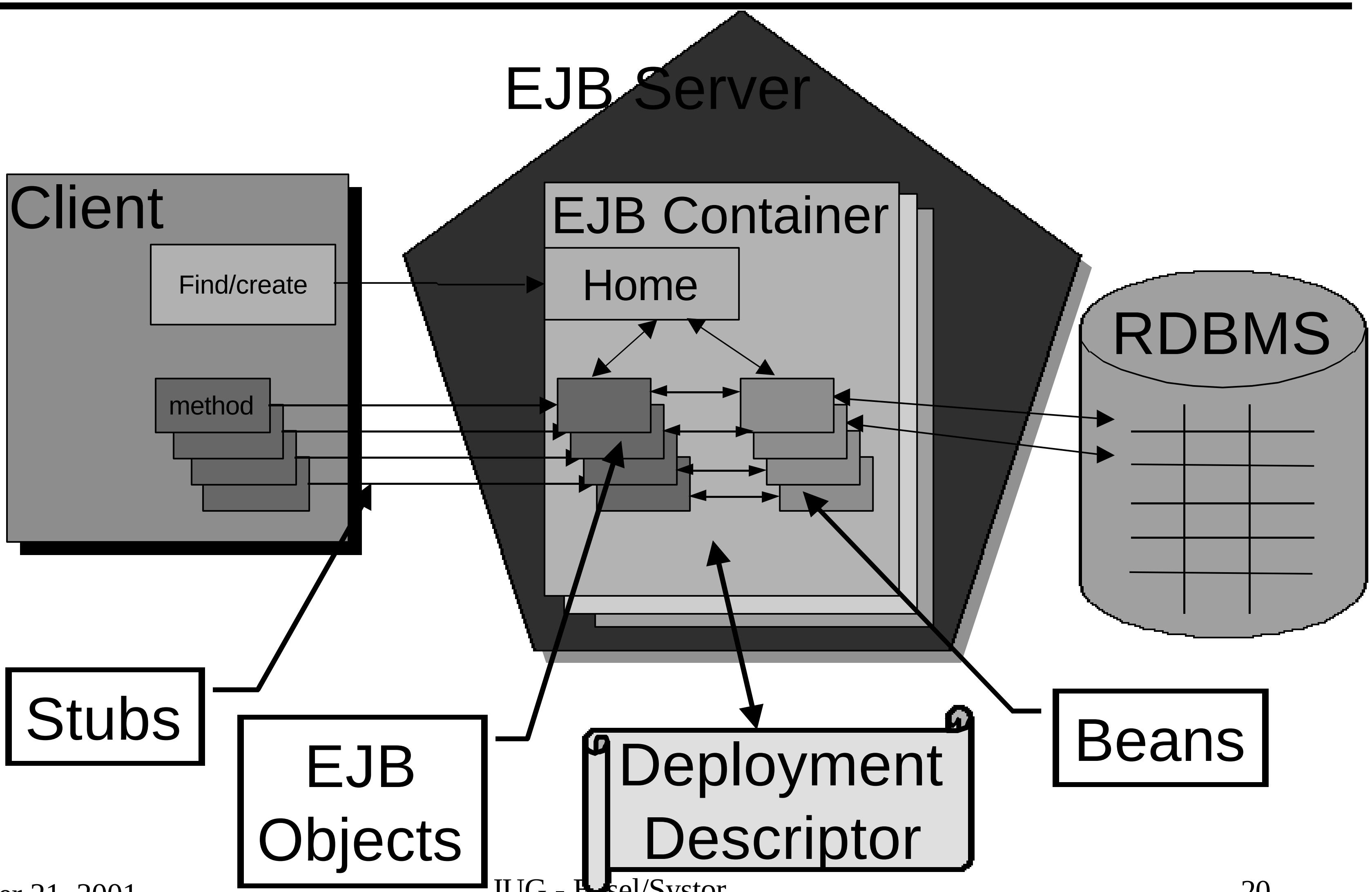
EJB Design


EJB Server
Client
Find/create
method
EJB Container
Home
RDBMS
Stubs
EJB
Objects
Deployment
Descriptor
Beans

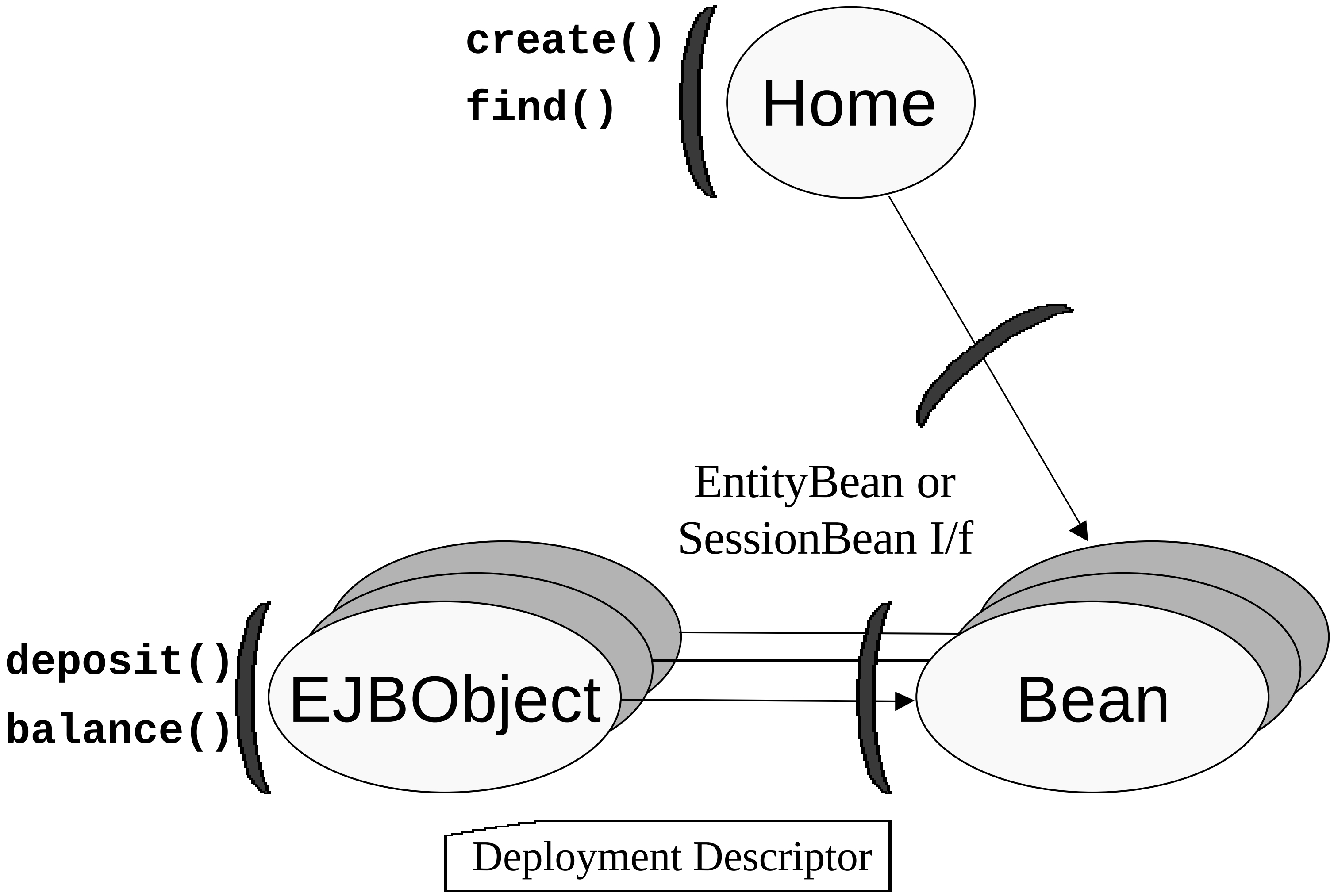
EJB Taxonomy



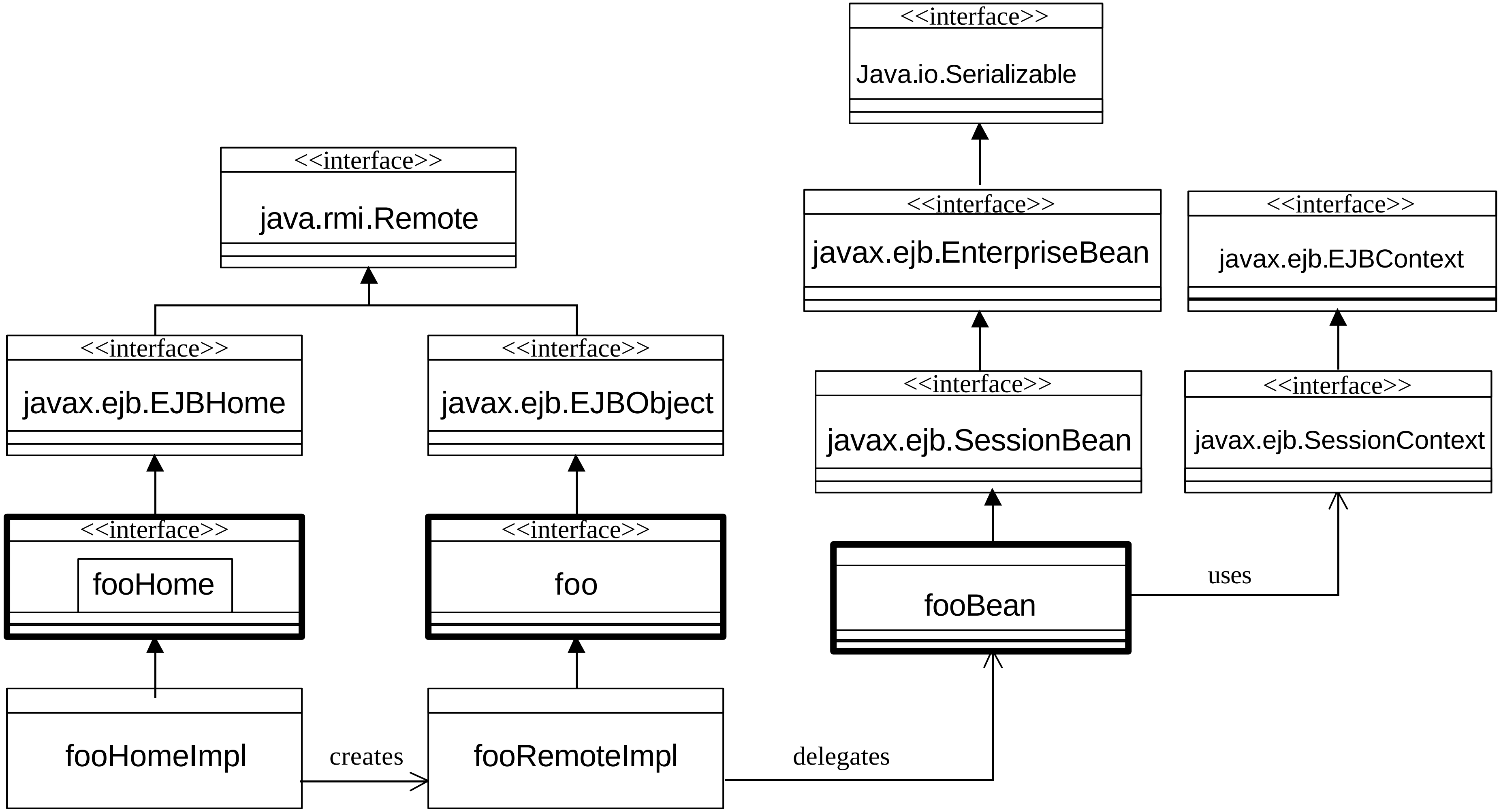
EJB Design


EJB Server
Client
Find/create
method
EJB Container
Home
RDBMS
Stubs
EJB Objects
Deployment Descriptor
Beans

Closeup view



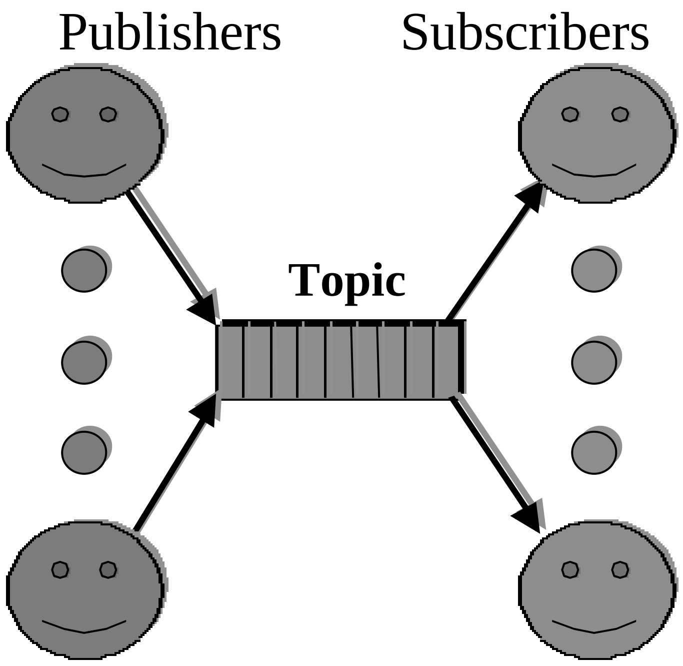
EJB Delegation / Inheritance - Session Bean



JMS – Java Messaging Service

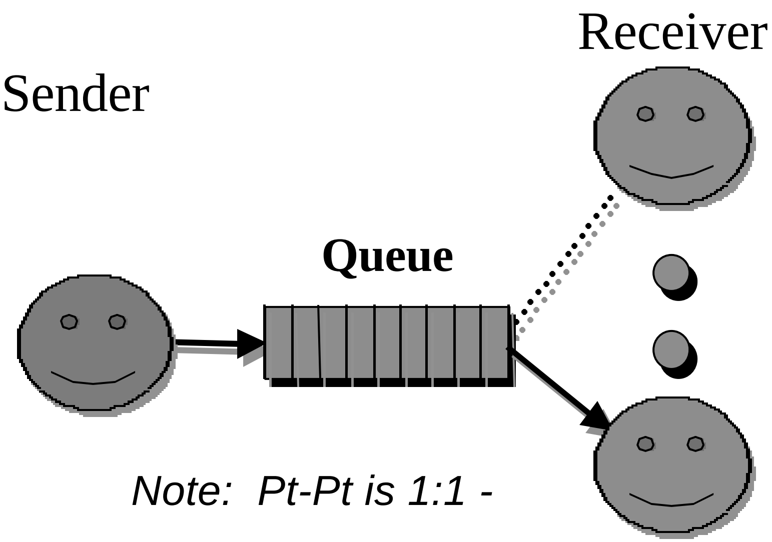
2 Messaging Models / Domains:

- Publish & Subscribe: **Topics**



Note: Pub/Sub is M:N

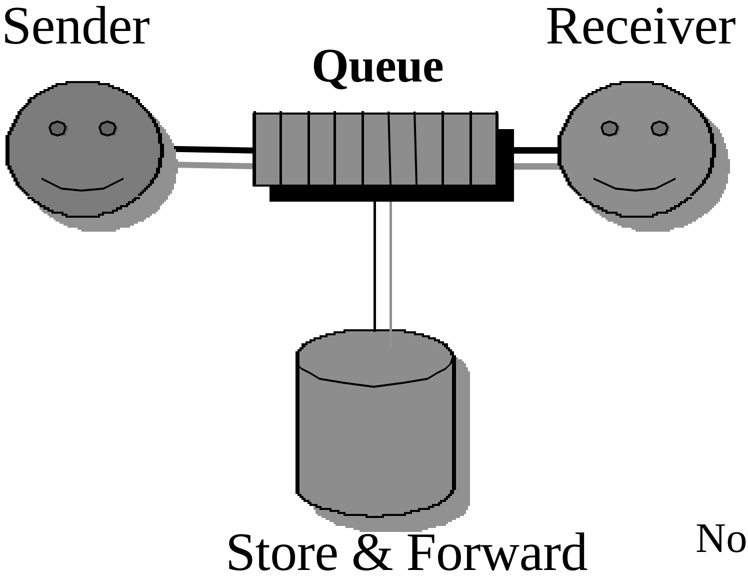
- Point-to-Point: **Queues**

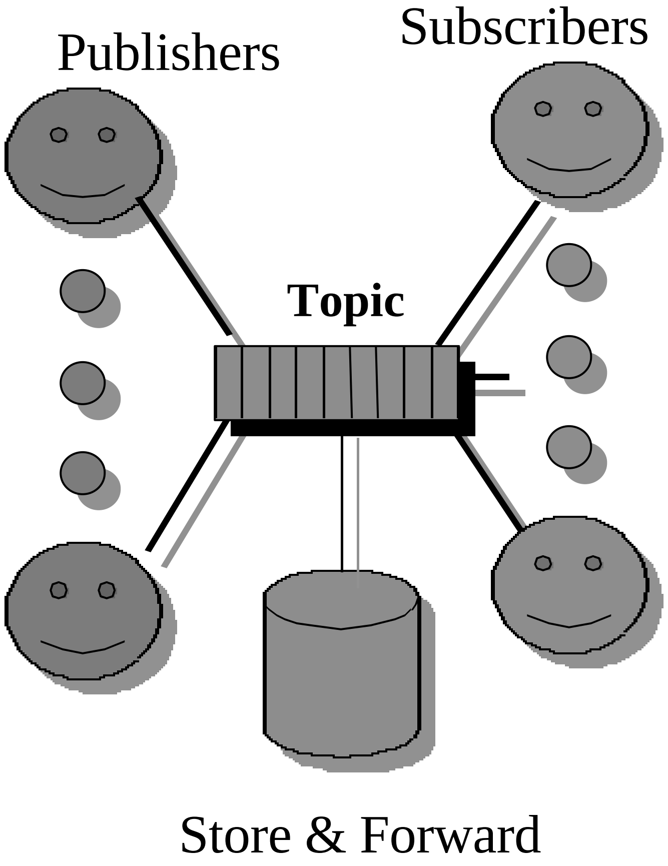


Note: Pt-Pt is 1:1 - – only one receiver may actually receive a message. For performance, the receiver may be chosen from a pool.

JMS - Quality of Service - QoS

- **At Most Once**
- **Once and only Once**
  - Via Store & Forward





Note: Guaranteed Ordering of messages

JMS – Message Structure

- **Header – message delivery fields**
  - Delivery Mode (Persistent, Non-Persistent)
  - Priority, MessageID, CorrelationID, Type, Destination, Time-stamp, Replyto, Redelivered, Expiration
- **Properties - application defined property values**
- **Body – data payload can be of 5 types:**
  - Bytes - Opaque, arbitrary data
  - Objects - Serialized Java objects
  - Map - Name-Value pairs
  - Stream - Sequences of typed data primitives
  - Text - Text (optionally including XML) string data

JMS API

- **ConnectionFactory – Connection objects**
  - TopicConnectionFactory – topic Connection objects
  - QueueConnectionFactory – queue Connection objects
- **Session objects provide a context for sender/receiver**
- **send()**
- **receive() - (proactive, blocking, polling)**
- **onMessage - (passive, as a registered callback)**

Let’s look at an example in detail…

JMS – Example: publisher

```
Properties env = new Properties();
…                                    //Specify JNDI props for your specific JNDI service provider
jndi = new InitialContext(env);
//  get a connection factory for Publish & Subscribe
factory = (TopicConnectionFactory)jndi.lookup(“TopicConnectionFactory”);
//  create a connection
connect = factory.createTopicConnection (userName, password);
//  create a session for publishing and one for subscriptions, non-transacted
pubSession = connect.createTopicSession(false, Session.AUTO_ACKNOWLEDGE);
subSession = connect.createTopicSession(false, Session.AUTO_ACKNOWLEDGE);
aTopic = (Topic)jndi.lookup(“A Topic”);
//  create  publisher and subscriber
publisher = pubSession.createPublisher(aTopic);
subscriber = subSession.createSubcriber(aTopic); //  Note: no filter specified for simplicity
//  associate onMessage() handler with this subscriber
subscriber = setMessageListener(this);
Connect.start();                                              //  start your messages!
TextMessage textMsg = pubSession.createTextMessage();
textMsg.setText(“Hello World”);
publisher.publish(textMsg, javax.jms.DeliveryMode.PERSISTENT,
                             javax.jms.Message.DEFAULT_PRIORITY, 1800000);  // TTL = 30 minutes
```

JMS – Example: subscriber

```
public void onMessage(javax.jms.Message message) {

   TextMessage textMsg = (TextMessage) message;

   System.out.println(textMessage.getText());
}
```

JMS
Problems & Issues

- Developer typically calls JMS from a stateless session bean
  - No support for component development
- Developer must create a **MessageListener** class, whose instances come from a server-wide session pool
  - Topic/queue classes tightly tied to available server resources
  - No container management for scalability, transactions, etc
  - Developer must develop queue/topic pooling classes
  - “Start-up” classes must be coded to initialize JMS destinations

EJB 2.0 – Message-Driven EJB

- EJB 2.0 Final Draft Standard establishes a new kind of bean: ***Message-Driven Bean (MDB)***
- Unlike other EJBs:
  - MDBs have **no Home/Remote Interfaces**
  - MDBs only indirectly interact with clients
    - MDBs cannot be created/removed by client actions
    - MDBs are managed solely by the container
  - MDBs process messages asynchronously
  - MDBs support concurrent processing of Topics & Queues
    - (in JMS this would have to be specially coded by the developer)
  - Note: one MDB can only interact with one Queue / Topic

MDB Example:

```
public class myMessageBean implements MessageDrivenBean, MessageListener {
    private MessageDrivenContext mdbContext;
    public myMessageBean( ) { }                              //  no-arg default constructor
    public void ejbActivate( ) { }                           //  EJB spec lifecycle control
    public void ejbRemove( ) { mdbContext = null;  }         //  EJB spec lifecycle control
    public void ejbPassivate( ) { }                          //  EJB spec lifecycle control
    //  set the MDB context
    public void setMessageDrivenContext ( MessageDrivenContext ctx ) { mdbContext = ctx; }
    ejbCreate ( )  throws CreateException { }                //  EJB spec lifecycle control
//  implement MessageListener
    public void onMessage ( Message msg ) {
            try {                               //  ensure no Exceptions escape to container
                    TextMessage tmsg = (TextMessage) msg;
                    String text = tmsg.getText ( );
                    System.out.println ( “myMessageBean : “ + text );
            }
            catch ( Exception e ) {             //  catch ALL exceptions
                            e.printStackTrace ( );
            }    // catch
        }        // onMessage
    }            // class myMessageBean
```

MDB Example: Deployment Descriptor

ejb-jar.xml - excerpt

```
<message-driven>
    <ejb-name>myMessageBean</ejb-name>
    <ejb-class>com.bea.myMessageBean</ejb-class>
    <transaction-type>Container </transaction-type>
     <message-driven-destination>
           <jms-destination-type>javax.jms.Topic </jms-destination-type>
     </ message-driven-destination >
</message-driven>
```

WebLogic Server config.xml JMS Entry

```
<JMSServer Name=“myJMSServer” Targets=“myServer”>
    <JMSTopic JNDIName=“myTopic” Name=“myTopic”/>
</JMSServer>
```

EJB 2.0 – Free Software Evaluation

WebLogic Server 6.0 provides an evaluation version of this new technology today – available for download from BEA’s Developer Center site -

http://developer.bea.com

&

http://commerce.beasys.com/downloads/weblogic_server.jsp

How business becomes e-business™

www.beasys.com