Work groups meeting - 2

INF5040 (Open Distributed Systems)

Sabita Maharjan
sabita@simula.no

Department of Informatics
University of Oslo
September 04, 2009

Outline

- Features of Distributed Systems
- Enterprise Java Beans (EJB)
- EJB 3.0
- J2EE Application Servers
  - JBoss

Key Requirements for Distributed Systems

- Remote Method Invocation
- Load Balancing
- Transparent failover
- Integration to legacy systems
- Transactions
- Clustering (state replication across servers)
- Resource pooling
- Security …..

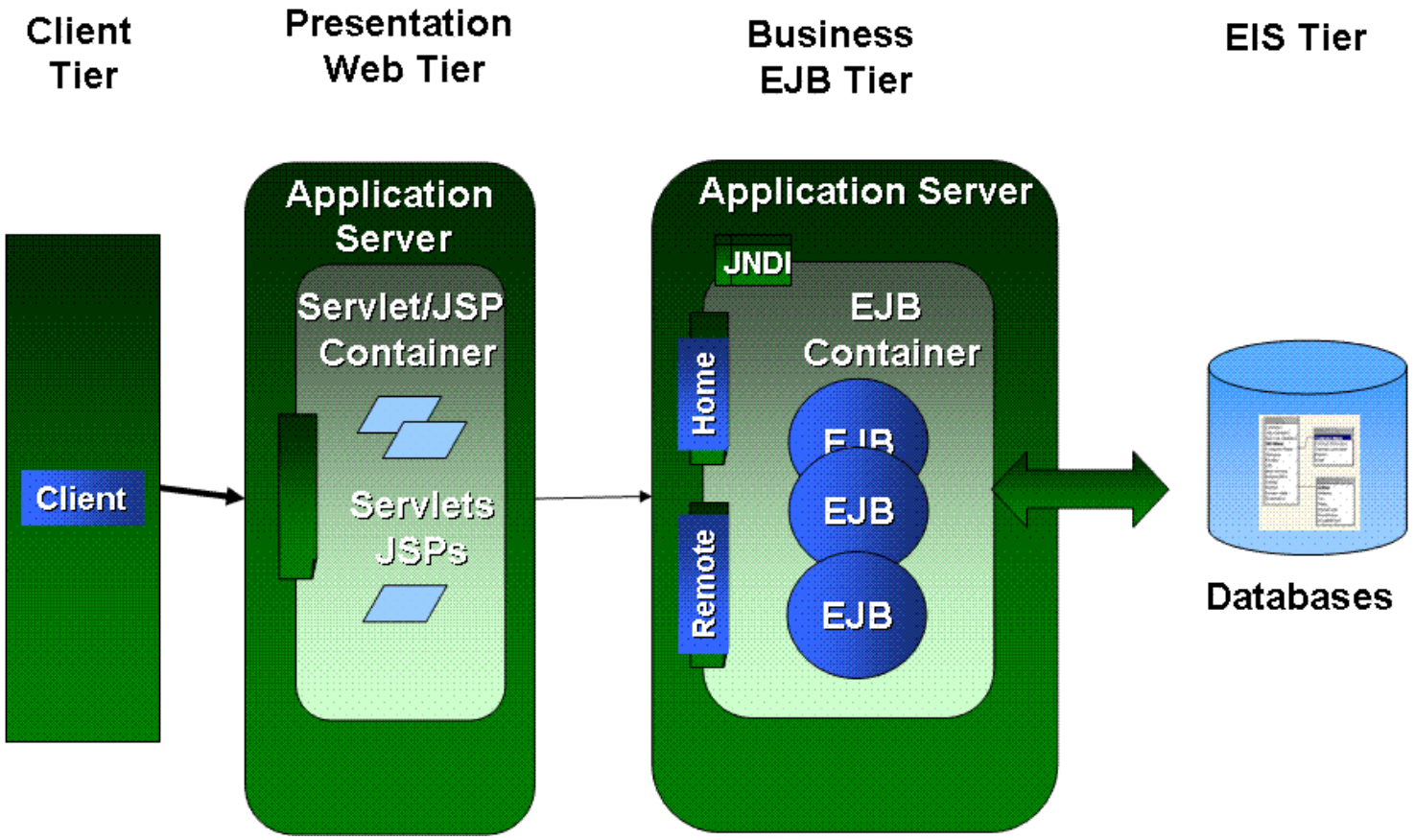
Key Requirements for Distributed Systems

- **Middleware services** provides those features
- Goal: To separate those issues from the developers of distributed app
  - Business logic is more important for developers
  - Too costly to build from scratch
  - Distributed app should be portable across middleware service providers

J2EE as a middleware

- *"The Java 2 platform, Enterprise Edition reduces the cost and complexity of developing ... multi-tier services, resulting in services that can be rapidly deployed and easily enhanced“*
- Introduced in June, 1999

J2EE Architecture



J2EE API Services

- Primary technologies
  - Servlets

    Server side components that provide component-based, platform-independent methods for building Web-based applications
  - Java Server Pages (JSPs)

    A technology for controlling the content or appearance of Web pages through the use of servlets
  - **Enterprise JavaBeans (EJB)**

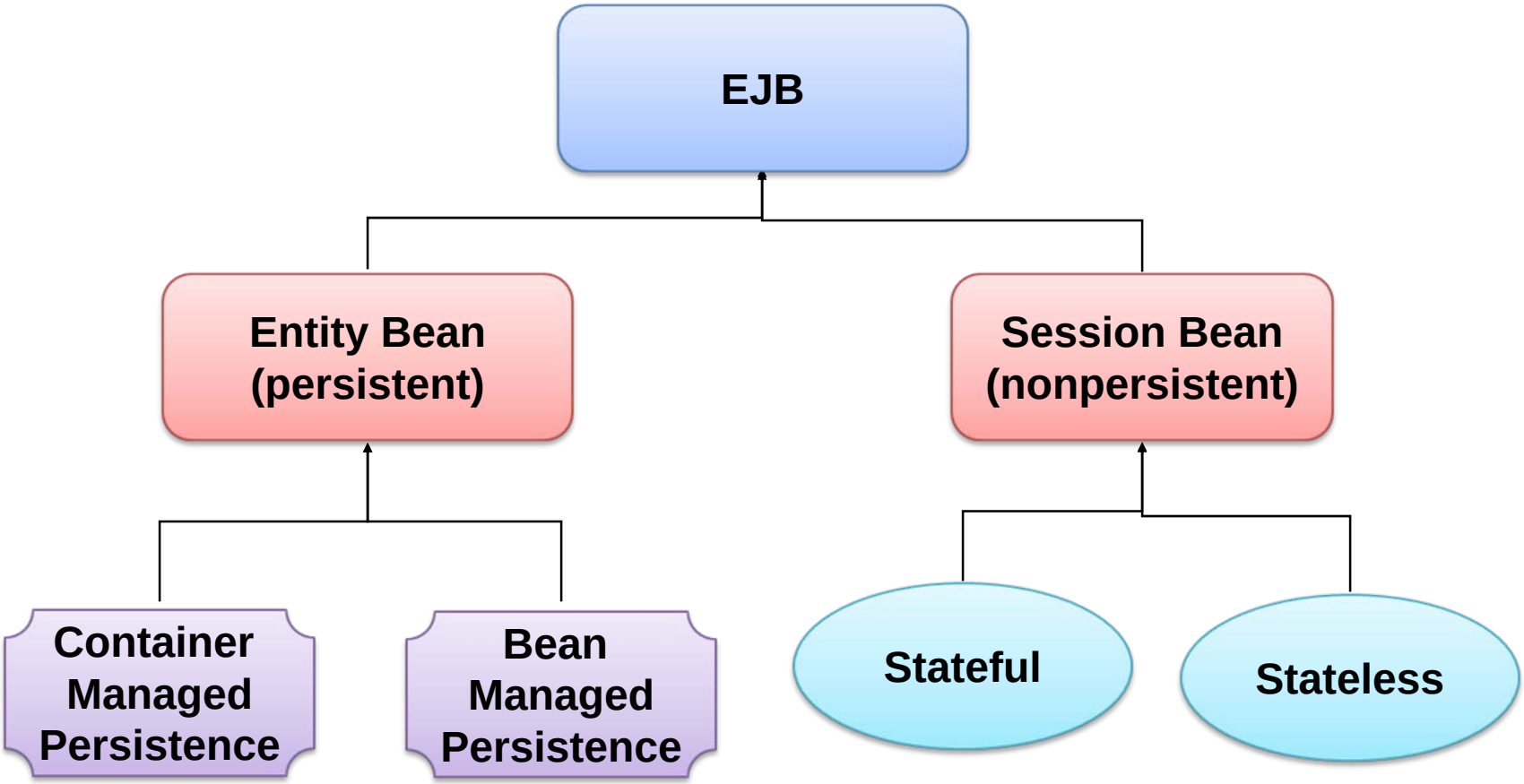
J2EE API Services

- JNDI – Java Naming and Directory Interface
- RMI - Remote Method Invocations
- JTA – Java Transaction API
- JDBC – Java Database Connectivity
- JMS – Java Messaging Service
- JavaMail – Java Mail API
- JAXP – Java API for XML Parsing
- Connector – standard for connecting to enterprise applications such as ERPs
- JAAS – Java Authentication and Authorization Service

Enterprise Java Beans (EJB)

- A server-side software component model for distributed enterprise applications
- Core of the J2EE technology
- Goal: To provide a standard way to manage distribution issues in enterprise applications through **reusable code**
  - By implementing the back-end 'business' code typically to enterprise applications
  - Addressing the same types of problems through the same code

EJB Classification



EJB Classification

- Entity Beans
  - Represent persistent business Entity
  - Persist in storage system (usually Database)

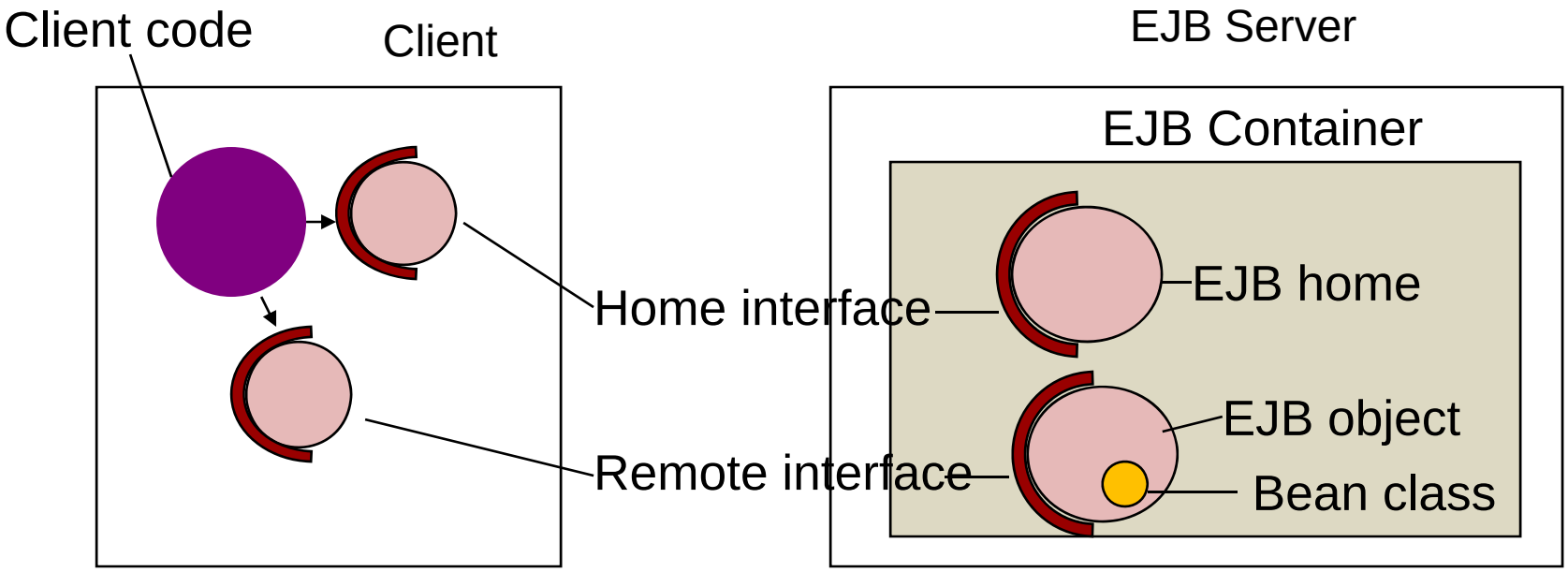
- Entity beans can have
  - Bean-managed persistence – The entity bean code that the developer writes contains the calls that access the database
  - Container managed persistence – The EJB container automatically generates the necessary database access calls.

EJB Classification

- Session Beans
  - Perform work for individual clients on the server
  - Encapsulate complex business logic
  - Can coordinate transactional work on multiple entity beans
- Session beans can be
  - Stateless- The SBs belong to the client for duration of the method call only
  - Stateful- The SBs belong to the client for duration of client lifetime

EJB Architecture

EJB home implements all the methods defined by the home interface

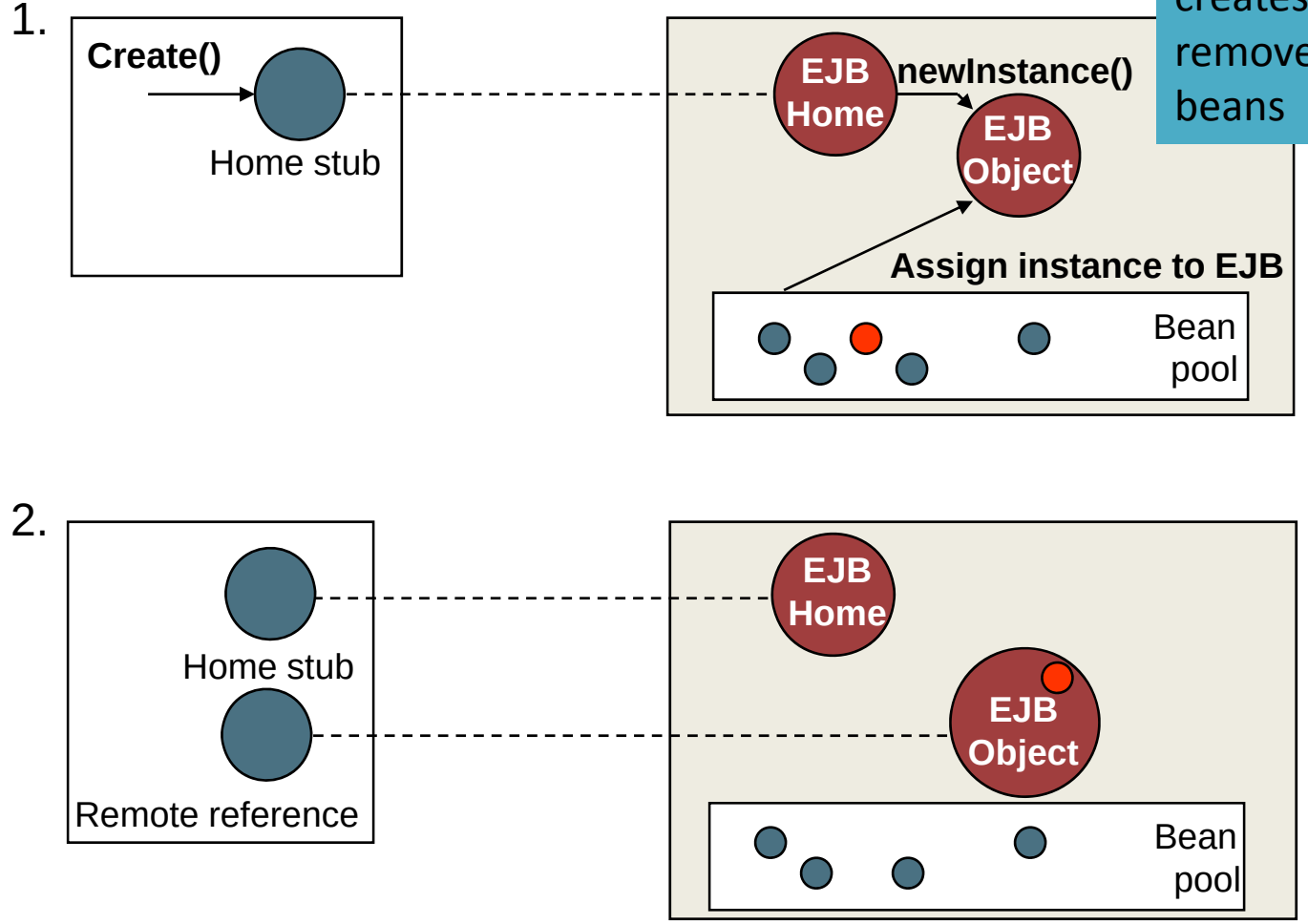


The EJB object is generated by the utilities provided by the vendor of the EJB container

The EJB object works with the container to apply transactions, security, and other system-level operations to the bean at runtime.

EJB Architecture

EJB home locates, creates, as well as removes enterprise beans



Advantages of EJB

- Industry standard
  - Vendors design containers to EJB specs
- Portability possible across containers
- RAD ability because of the services provided by container

The Other Side …..

- Rarely necessary for web-based applications
- *Spring*, a lightweight container, can achieve most of the EJB services
  - makes it easy to use many different services and frameworks
  - accepts any Java bean instead of specific components
- Complexity
  - Specifications are getting more complex
  - EJB 3.0 tries to address this issue
- Difficulty in Unit testing
  - If anything goes wrong, is it container related or app related?

EJB 3.0

- Too many interfaces to implement in EJB 2.1
- Complex deployment descriptors in EJB 2.1
- Focuses on “ease of use”
- EJB 3.0 specification has made the xml deployment descriptors optional

SessionBean in EJB 3.0

- Remove unnecessary interfaces
- Remove unnecessary callbacks
- Make deployment descriptors optional
- Make beans pojo-like (Plain Old Java Object)

SB - Stateless

- Homeless
- Methods don't throw RemoteException

```
@Remote public interface Calculator  {
        public int add(int x, int y);
        public int subtract(int x, int y);
}
@Stateless public class CalculatorBean implements Calculator {
        public int add(int x, int y) {
                return x + y;
        }
        public int subtract(int x, int y) {
                Return x – y;
        }
}
```

SB - Stateful

- Homeless
  - Created as they are looked up
- Stateful bean is removed after the method is called
- @Remove is used instead of explicitly calling EJBObject.remove

```
@Remote public interface ShoppingCart  {
        public void addItem(int prodId, int quantity);
        public void checkout();
}

@Stateful public class ShoppingCartBean implements ShoppingCart {

          @Remove
        public void checkout()  {
            ...
        }
}
```

SB - Transactions and Security

```
@Stateful public class ShoppingCartBean implements ShoppingCart {

        @Remove
        @TransactionAttribute(REQUIRED)
        @MethodPermission({“valid_customer”})
        public void checkout()  {
            …
        }
}
```

Entity Bean

- "Make it simple"
  - Fewer interfaces, etc.
  - Plain Java based
- Provide full Object/Relational mapping
- Supports Inheritance
- Expanded EJBQL
  - Fully featured
  - Parallel SQL
  - Polymorphic Queries

Entity Bean

O/R Mapping Metadata as annotations

- Table mappings, @Table, @SecondaryTable
- Column mappings, @Column, @JoinColumn
- Relationships, @ManyToOne, @OneToOne, @OneToMany, @ManyToMany
- Multi-Table mappings, @SecondaryTable
- Inheritance, @Inheritance, @DiscriminatorColumn
- Identifier + Version properties, @Id, @Version

Entity Annotations

```
@Entity
@Table(name="AUCTION_ITEM")
public class Item {
    private long id;
    private String description;
    private String productName;
    private Set<Bid> bids = new HashSet();
    private User seller;

   @Id(generate=GeneratorType.AUTO)
   @Column(name="ITEM_ID")
    public long getId() {
        return id;
    }
    public void setId(long id) {
        this.id = id;
    }
   @Column(name="DESC", length=500)
   public String getDescription() {
        return description;
    }
    public void setDescription(String desc) {
        this.description = desc;
    }
}
```

```
create table AUCTION_ITEM
(
ITEM_ID Number,
DESC varchar(500),
ProductName varchar(255),
OWNER_ID Number
);
```

Entity Relationships

```
@OneToOne(fetch=LAZY)
@JoinColumn(name="OWNER_ID")
public Owner getOwner() {
    return owner;
}

public void setOwner(Owner owner) {
    this.owner = owner;
}

@OneToMany(cascade=ALL)
@JoinColumn(name="ITEM_ID")
public Set<Bid> getBids() {
    return bids;
}
public void setBids(Set<Bid> bids) {
    this.bids = bids;
}
public void addBid(Bid bid) {
   getBids().add(bid);
   bid.setItem(this);
}
```

```
create table AUCTION_ITEM
(
ITEM_ID Number,
DESC varchar(255),
ProductName varchar(255),
OWNER_ID Number
);

create table BID
(
…
ITEM_ID Number
…
);
```

Multi-table Mappings

```
@Entity
@Table(name="OWNER")
@SecondaryTables({(name="ADDRESS"
        join={@JoinColumn(name="ADDR_ID")})
public class Owner {
   private long id;
   private String name;
   private String street;
   private String city;
   private String state;

  @Id(generate=GeneratorType.AUTO)
  @Column(name="OWNER_ID")
   public long getId() {
      return id;
   }
   public void setId(long id) {
      this.id = id;
   }

  @Column(name="STREET", secondaryTable="ADDRESS")
  public String getStreet() {
      return street;
   }
   .....
```

```
create table OWNER
(
OWNER_ID Number,
NAME varchar(255),
);

create table ADDRESS
(
ADDR_ID Number,
STREET varchar(255),
CITY varchar(255),
STATE varchar(255)
);
```

Query API

- Queries may be expressed as EJBQL strings
- Invoke via `Query` interface

```
@Session public class ItemDAOImpl {
   ...
   public List findByDescription(String description, int page) {
            return em.createQuery("from Item i where i.description like :d")
               .setParameter("d", description)
               .setMaxResults(50)
               .setFirstResult(page*50)
               .listResults();
         }
```

Application Servers

Definition

A set of software frameworks, components, utilities, functionality that enables you to develop and deliver n-tiered web applications

J2EE App. Server

A software package that implements all of the J2EE APIs and frameworks that allow you to develop and deliver J2EE applications

App. Servers Characterisitics

- Connectivity to various applications and DBs on various operating environments and hardware
- Provides an integrated IDE for all aspects
- Support for reusable distributed components (CORBA, COM, EJB)
- Performance management (load balancing, caching, monitoring)
- Robust and reliable software – redundancy, backup/recovery
- User-friendly administrative, diagnostic tools
- Strong security framework

J2EE Application Server

- A J2EE Application Server provides (at a minimum):
  - Servlet engine/container
  - EJB engine/container
  - Web Server (or ability to integrate with a web server)
  - Concrete implementations of all the J2EE APIs
    - Java Mail
    - Messaging
    - RMI etc.
- A vendor-bought App Server will have additional functionalities

J2EE Application Server

- Products
  - Full J2EE App Servers
    - BEA Web Logic
    - IBM Websphere
    - Oracle App Server
    - **JBoss (FREE / open source)**
  - Others
    - Tomcat (servlet only - FREE)

JBoss

- Introduced in 1999
- The product of an OpenSource developer community dedicated to developing the best J2EE-compliant application server in the market
- With 1000 developers worldwide and a steadily growing number of downloads per month, reaching 72,000 for October ’01 (per independent www.sourceforge.net), JBoss is arguably the most downloaded application server in the world today
- Distributed under an LGPL license, JBoss is absolutely FREE for use.

JBoss Properties

- JBoss 4.0 is compliant to the J2EE 1.4 specification. So J2EE components such as JBs and EJBs can be reused even if they were developed on other application servers
- It also supports J2EE Web Services and Service Oriented Architecture
- Since JBoss is java based, it can be used on any operating system that supports java
- JBoss requires smaller memory than other application servers, so it is faster
- It has a very powerful documentation on its web page

Jboss Application

- Jboss is also free but needs more knowledge about J2EE and have to manually create deployment packages.
- Runs on any Java platform and is robust.
- For applications that are not extremely complex, JBoss is the optimal choice.

JBoss

Source code can be found at

http://sourceforge.net/projects/jboss

Jboss EJB tutorials

http://www.jboss.org/file-access/default/members/jbossejb3/freezone/docs/tutorial/1.0.7/html/index.html

Resources

- http://java.sun.com/products/ejb/
- http://java.sun.com/javaee/
- http://www.jboss.org
- Master Enterprise JavaBeans 3.0, 4th Ed, Sriganesh et. Al., 2006
- http://oreilly.com/catalog/entjbeans/chapter/ch02.html

- Preferences for group composition
  - Please email to: sabita@simula.no
- Sept. 14 – Oct. 3
  - Amir Taherkodi will be available to help you
  - amirhost@ifi.uio.no
- Sept. 14, Group meeting at IFI 3B