# Performance of EJB Applications

Siddhartha Rai

## What is EJB?

- A component model for scaleable, reusable, portable, transactional, and distributed enterprise applications.
- EJB are server-side components that encapsulate business logic and are designed to be run in an EJB *container*.
- Individual EJB can be combined to create an enterprise application.

## EJB

- Enterprise Java beans
  - Asynchronous
    - Session
      - Stateless
      - Stateful
    - Entity Beans
      - Bean Managed Persistence
      - Container Managed Persistence
  - Message Driven Beans

## Enterprise Computing Challenges

- Building distributed applications is complex
  - Transactions
  - State management
  - Multi-threading
- Complexities of different operating system calls, interoperability between different communication protocols.
- Locating servers easily and transparently
- Solutions – EJB, JNDI, RMI-IIOP

## Related Technologies - JNDI

- Naming service – mapping of names to object bindings so clients can access objects by name.
  - (Associated terms – Naming service, contexts, sub contexts, naming conventions, namespaces.
- Directory Service
  - A directory is a connected set of directory objects.
  - Directory service organizes directory in a hierarchical manner.
  - Naming service uses a directory service( Such as Lighweight Directory Access Protocol- LDAP) to provide association of names to objects.

## Related technologies - Servlets

- Java code that run in a server application
- provide a component-based, platform-independent method for building Web-based applications.
- have access to the entire family of Java APIs
- Typical uses
  - Processing and/or storing data submitted by an HTML form.
  - Providing dynamic content, e.g. returning the results of a database query to the client.
  - Managing state information on top of the stateless HTTP

## Related Technologies – RMI-IIOP

- RMI enables distributed computing in Java
  - RMI client uses a remote interface to execute methods on RMI server.
  - RMI client uses a stub to marshal a request and unmarshal the response, RMI server uses a skeleton to unmarshal this request and to marshal the response.
  - RMI uses Java Remote Method Protocol (JRMP) to hide the low-level networking and data translation details.
- RMI-IIOP enables interoperability with non-Java and CORBA clients.

## J2EE containers

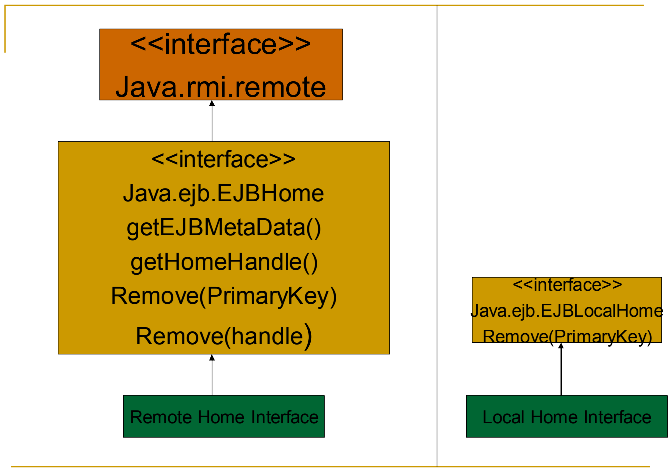
- EJB instances execute within EJB containers.
- Container provides the runtime environment --such as JVM- along with Java standard class libraries and other libraries to support specific components
- Container interacts with the components by way of standard API's and it implicitly manages security, transactions, and life cycle of component

## EJB client view

- Client may be a servlet, JSP, standalone Java or CORBA app, or another EJB.
- Entity and session beans exist only within their respective containers and client can only access them via their interfaces.
- Interfaces
  - Two kinds – Remote and Local
  - Two more kinds – home and component, within each category
    - Home interface used to manage life cycle of bean instance – create, remove, find…
    - Component interface - business methods

## Remote and Local Interfaces

- Remote interfaces
  - provide location independent view of EJB's.
  - Implement RMI-IIOP interfaces
  - Arguments and results are passed by value
- Local Interfaces ( EJB 2.0 )
  - May be used if clients are co-located in the same JVM as bean instance
  - Arguments and results passed by value.
- Before EJB 2.0 vendors optimized local calls by passing objects by reference JBoss(optimized) and JonAs(Jeremie).



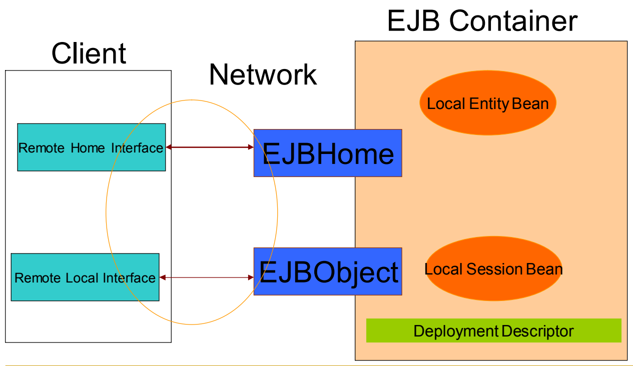
## EJB client view (contd.)

- When client invokes a business method –
  - the remote component interface's stub marshals the client's parameters
  - sends it to *EJBObject* whose skeleton unmarshals the request.
  - container performs security, transaction and life-cycle services
  - passes the parameters to the matching method in the respective bean instance, where it is executed.
  - Result is passed back to the *EJBObject*, whose skeleton marshals the results before sending it to client.

## Container Design

- client never references an EJB bean instance directly, but rather references the *EJBObject* and *EJBHome* which implement the beans remote and home interfaces respectively.
- Question - Who generates these classes?
  - Answer 1 – Container generates them (JOnAS)
  - Answer 2 – They are never physically generated, but realized using dynamic proxies (JBoss)
- Approach in answer 1 is faster when number of classes is small, while answer 2 is good when large number of classes have to be generated.

## Remote and Local Client views



## Session Beans

- Model "verbs" e.g. *buy, sell, approve*
- Exist only as long as client session
- Can not be shared
- Can be transactional
- Do not survive a container/server crash
- Two flavors – stateless – for processes that can be performed in a single request, and stateful for maintaining info between requests (e.g. shopping cart)

## Entity Beans

- Model nouns such as cars, books, invoices
- Provide an in memory view of persistent data stored in an enterprise system
- Represents persistent data along with data access logic that business processes can manipulate
- Transactional, survive server crashes
- Manage persistence

## Entity Beans (contd.)

- Container and entity bean instance work together to synchronize the in-memory data with the database.
- In case the EJB container or server crashes, the container is able to recreate the entity bean instance from data saved in the database.
- Entity beans are identified by primary keys.
- Multiple clients can access an entity bean instance

## Entity Bean's life cycle

- Three states
  - Does not exist - bean instance is not available, until container sets a context. (Container can not access bean instance)
  - Pooled - beans are not associated with an object identity or primary key, invocation of ejbCreate (by client) or ejbActivate (by container) changes state to pooled. Container may unset a bean's context to move it back to Does not exist state.
  - Ready – bean is associated with a primary key and can execute business methods for clients.
- Ps : Session beans do not have primary keys.

## Entity Contexts

- Allows entity beans to access its primary key, and local/remote home/component interfaces.
- Allow a bean instance to access transaction related methods such as getUserTransaction, getRollbackOnly and setRollBackOnly which it uses to control its transaction.
- Allow a bean instance to access security related methods such as getCallerPrincipal and isCallerInRole which can be used to implement EJB security .

## Managing persistence in Entity Beans

- Bean Managed Persistence
  - EJB developer is responsible for writing the necessary database access logic to manage the persistence in the entity bean class.
- Container Managed Persistence
  - Container is responsible for generating the code necessary for data access and management.
  - Bean developer is still responsible for specifying the container managed persistence fields in the bean class and declaring abstract persistence schema in the deployment descriptor.

## CMP 2.0 Persistence Model

- CMP 2.0 entity beans are associated with an abstract persistence schema, which is a logical persistence view.
- The abstract persistence schema is used by EJB QL ( a query language) allowing bean developers to focus at the object level without having an intimate knowledge of the physical schema of underlying database.
- EJB QL statements are used to describe the behavior of custom finder and select methods.

## EJB-QL Examples

SELECT OBJECT(p) FROM **Player p**

*Data retrieved:* All players.

*Finder method:* findall()

*Description:* The FROM clause declares an identification variable named p, omitting the optional keyword AS.

The Player element is the abstract schema name of the PlayerEJB entity bean. Because the bean defines the findall method in the LocalPlayerHome interface, the objects returned by the query have the LocalPlayer type

SELECT DISTINCT OBJECT(p) FROM Player p,
**IN (p.teams) AS t**
WHERE **t.city** = ?1

*Data retrieved:* The players whose teams belong to the specified city.

*Finder method:* findByCity(String city)

*Description:* The FROM clause declares two identification variables: p and t.
The p variable represents the PlayerEJB entity bean, and the t variable represents the related TeamEJB beans. The declaration for t references the previously declared p variable. The IN keyword signifies that teams is a collection of related beans.
The p.teams expression navigates from a PlayerEJB bean to its related TeamEJB beans.
The period in the p.teams expression is the navigation operator.

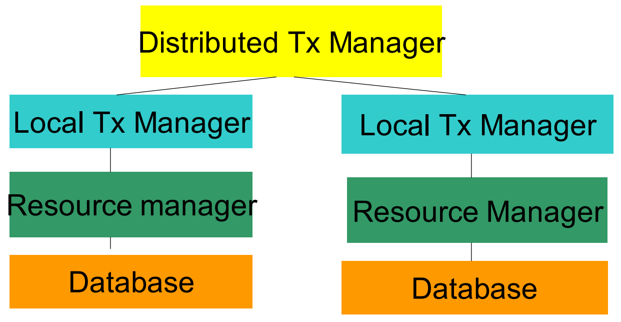
## Transactions

- Transactions may be declared programmatically (*bean managed*) or declaratively (*container managed*.)
- Every EJB method has following transaction attributes associated with it
  - NotSupported, Required(default), Supports, RequiresNew, Mandatory, Never.
- Bean Managed Transactions
  - - Using the EJB context ( either EntityContext or SessionContext ) the bean methods can access the Tx.
  - The Tx is available to EJBMethods using EJBContext.
  - Only session and message driven beans support bean managed Tx.
  - Only flat Tx supported.

## Transactions (contd.)

- Container Managed Transactions
  - Entity beans support only container managed Tx.
  - Can also be used for session and MDB.
  - EJB container demarcates the transaction scope by calling the appropriate begin(), commit() or rollback() methods based on the Tx attributes specified in the deployment descriptor.
  - Allows bean developers to force a transaction to rollback.

## Distributed EJB Transactions



## EJB Design Patterns – Value Object

- Motivation
  - Increased frequency of *get/set* methods impacts the performance of applications, reducing number of remote method calls will reduce overall network traffic.
- Solution
  - A ValueObject encapsulates data so it can be transferred in a single call
    - populate this object with attributes that the client is interested
    - Include accessor methods for these attributes
    - Serialize and send it to the client

## EJB Design Patterns – Session Facade

- Motivating reasons
  - Tight coupling between clients and business objects makes applications *brittle*.
  - Fine-grained method calls made by the client on the participating business objects increase network traffic.
- Solution
  - Use a session bean to encapsulate and hide the interactions between business objects participating in a workflow.
  - Coarse grained access layer responsible for locating, creating and executing business logic in the business objects.
  - handles interactions between business objects.
- PS : Based on the façade design pattern from OOP.

## EJB Design Patterns – Data Access Object

- Motivations
  - The API's that access the data should not be vendor specific.
    - Affects portability and flexibility in replacing data store.
  - Data Access Object design pattern abstracts and encapsulates all data-access implementations.
  - Manages connection and data access to the data store.
  - Exposed interface can remain same even if underlying data store is replaced.
- May be used only with BMP entity beans.

## EJB Design Patterns - Business Delegate

- Motivation
  - Reduce the coupling between clients on the presentation tier and business components in business tier.
- Use a BusinessDelegate object as a client side abstraction
  - Shields clients from changes in business service API's.
  - BusinessDelegate handles naming and lookup services.
  - Provides result caching .
  - Use with SessionFacade – (1-1 relationship.)

## EJB Design Patterns – Value List Handler

- Motivation
  - Clients need to retrieve lists of item and display them to the end user in a scrollable manner.
- Solution
  - Use a ValueListHandler, implemented as a session bean.
    - ValueList handler maintains a ValueList ( a list of ValueObjects.)
    - Return a collection of appropriate size to client .
    - Client may scroll through ValueList.

## EJB Performance Studies

- Application
  - Auction system .
  - Comparisons between
    - Servlets .
    - Session beans.
    - Entity beans – CMP .
    - Entity beans – BMP.
    - Session façade .
    - Session façade with EJB 2.0 local interfaces.

## Results

- Servlets only gives best performance (at peak load.)
- Followed by
  - Session beans
  - Session façade with local interfaces
  - Session façade
  - Entity beans (BMP and CMP)

## Conclusions

- Were the experiments well designed?
  - Can entity beans (nouns) be modeled by session beans (verbs) and vice versa?
- What other options do current technologies have to offer?