



# Developing Java™ Applications with the Enterprise ADF

*Divesh Goyal*

*Eric Bader*

# Key Objectives of This Presentation

- **Help You Answer:**

- **What, Why, and How on ADF's EJB™ components**
- **How does ADF support development of Geospatial Java Enterprise (EE) applications?**
- **What are the workflow steps for building Geospatial Java EE applications?**
- **Are there any nuances? If so, what are they?**
- **What are the development patterns for programming to the Geospatial EJBs?**
- **What resources are recommended and available for a Java developer who is new to ESRI technology?**

# Agenda

- **Introduction**
- **Inside the Enterprise Application Developer Framework**
- **Architecture**
- **Workflow**
- **Programming models and scenarios**
- **Application Deployment**
- **Learning resources**
- **Summary**

# Overview



**ArcGIS Server (Java) includes framework to build and deploy GIS-enabled J2EE applications using Enterprise JavaBeans technology (EJB)**

- **Out-of-the-box EJBs that perform GIS tasks**
  - Mapping, Geocoding, Geodata Querying, Geoprocessing, Network solving
- **Deployed across a wide variety of J2EE 1.4 certified application servers -- such as BEA, Oracle, IBM, JBoss, and Sun**
- **Integrated into Manager Application for simple point click ease**
- **Focus on generated applications and not coded applications**
- **High Performance**

# Understanding EJBs

- **Enterprise JavaBeans (EJB )**
  - An architecture for developing secure transactional applications as distributed components in Java
- **Server-side Components**
- **Clients can be:**
  - Standalone desktop applications (Client-server)
  - Web Applications (Servlets/JSPs/Web Services)
  - Other EJBs (remote or local)

# EJB Advantages

- **Advantages**

- **One source for business logic**
- **Accessible by many types of clients**
- **Integrates well with other systems**
- **Clear separation of business from client tier**
- **Easily distributed**

# What's inside the Enterprise ADF?

- **EJBs: Pre-packaged, ready for configuring**
  - Mapping
  - Geocoding
  - Geoprocessing
  - Network Analysis
- **API: a pure Java library**
  - Packaged in: **com.esri.arcgisws.\***
- **Java EE standard Resource Adaptor**
  - Optional
  - Provided for strict J2EE compliancy when connecting to ArcGIS Server
- **SDK: Documentation and Sample client applications**
  - Demonstrate best practices and recommended development patterns
- **Eclipse IDE Integration**
- **Manager Integration**

# What functionalities do the EJBs provide?

- **Map EJB**

- Get maps (URLs, byte[])
- Query data (Find, Identify, get Feature recordsets, etc)
- Coordinate transformations (screen-to-map, map-to-screen)
- Calculate distances

- **Geocode EJB**

- Geocode an address, lists of addresses
- ReverseGeocode
- Obtain Locator properties, etc.

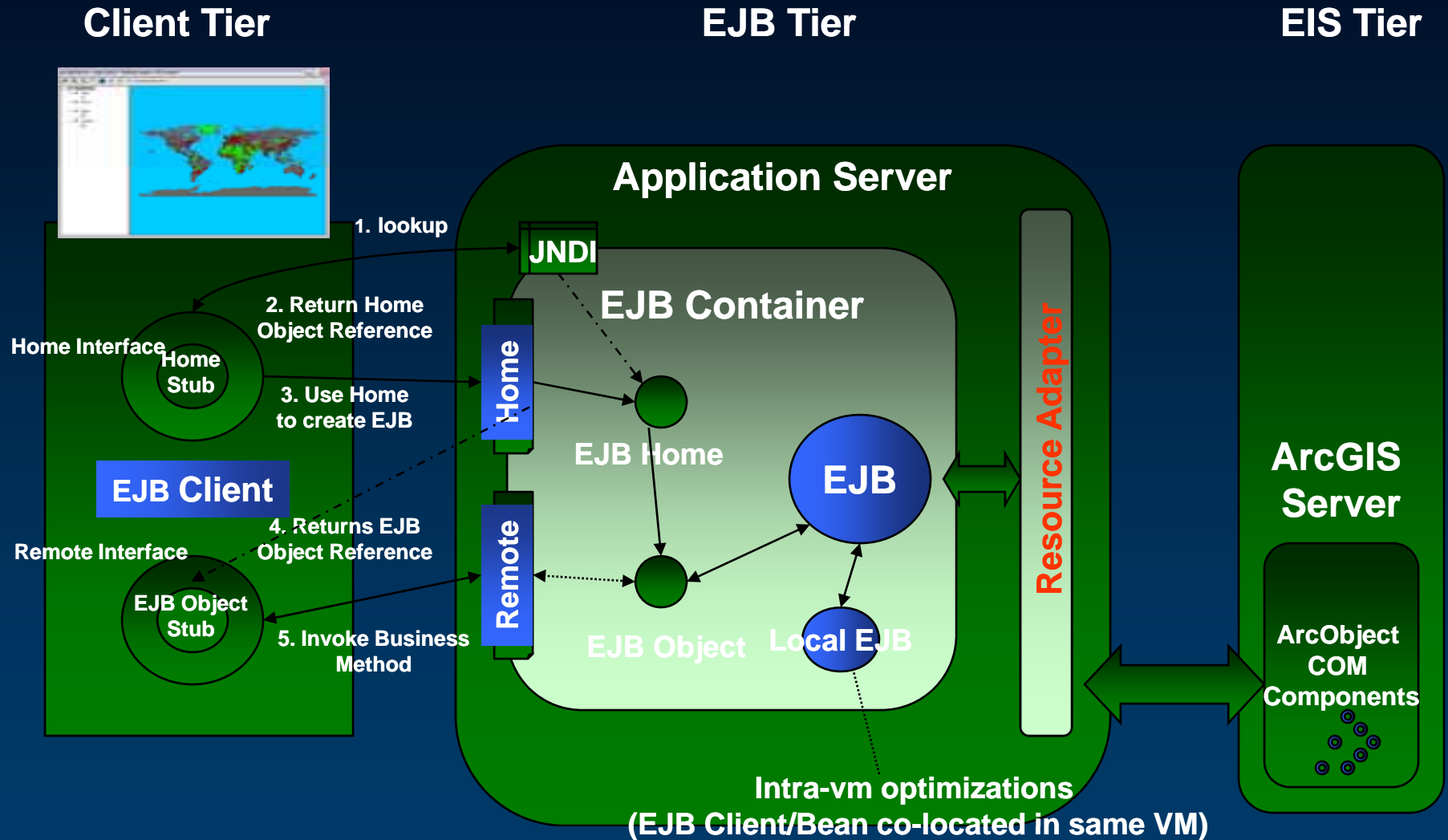
- **Geoprocessing EJB**

- Execute a tool/model remotely, asynchronously
- Get tool/model information, parameters

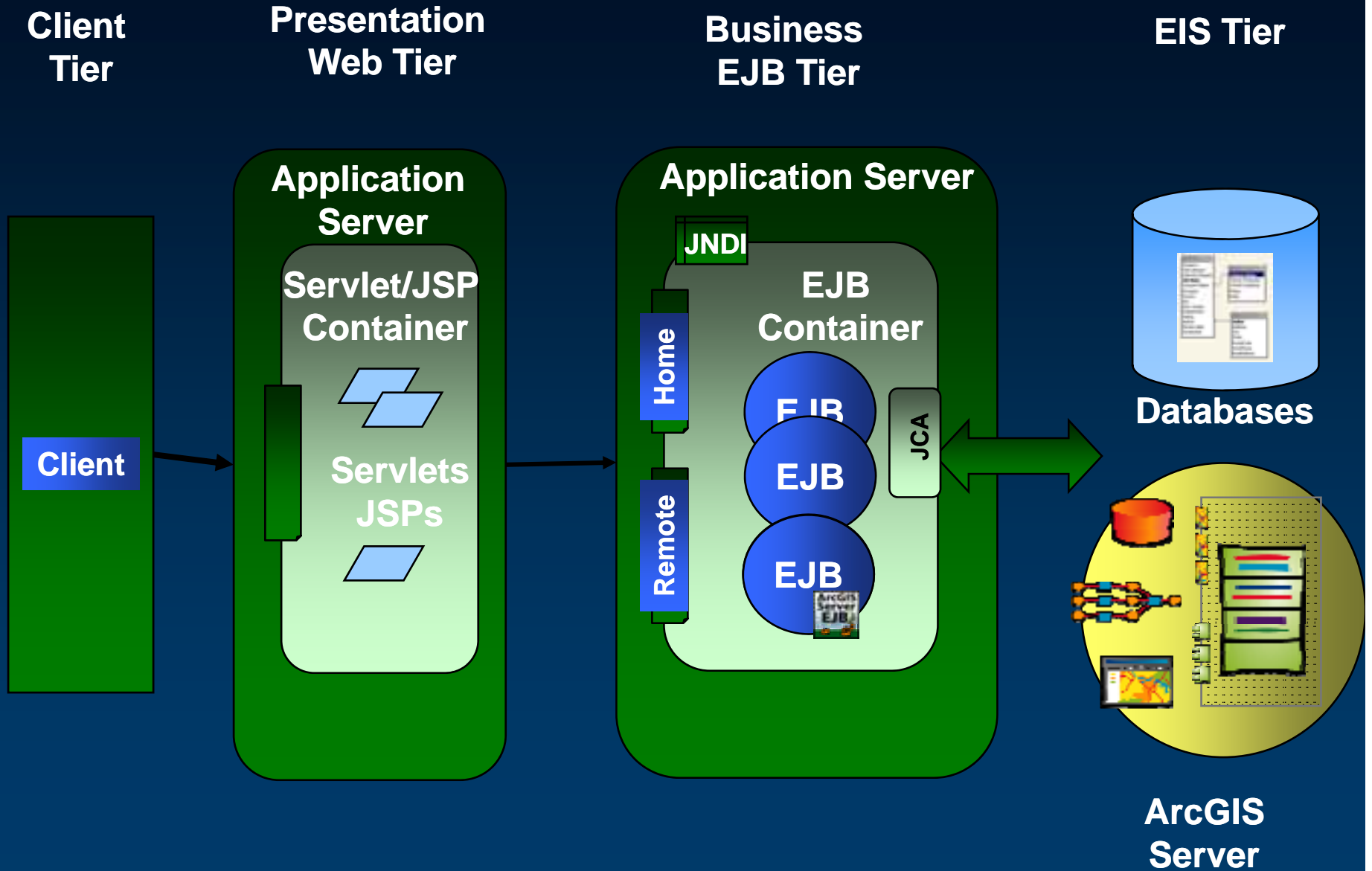
- **Network Analyst EJB**

- Solve ClosestFacility, point-to-point or multiple stop routing, Service Areas with time windows, directions, etc.

# Architecture



# Typical J2EE Use Case Scenario in GIS space

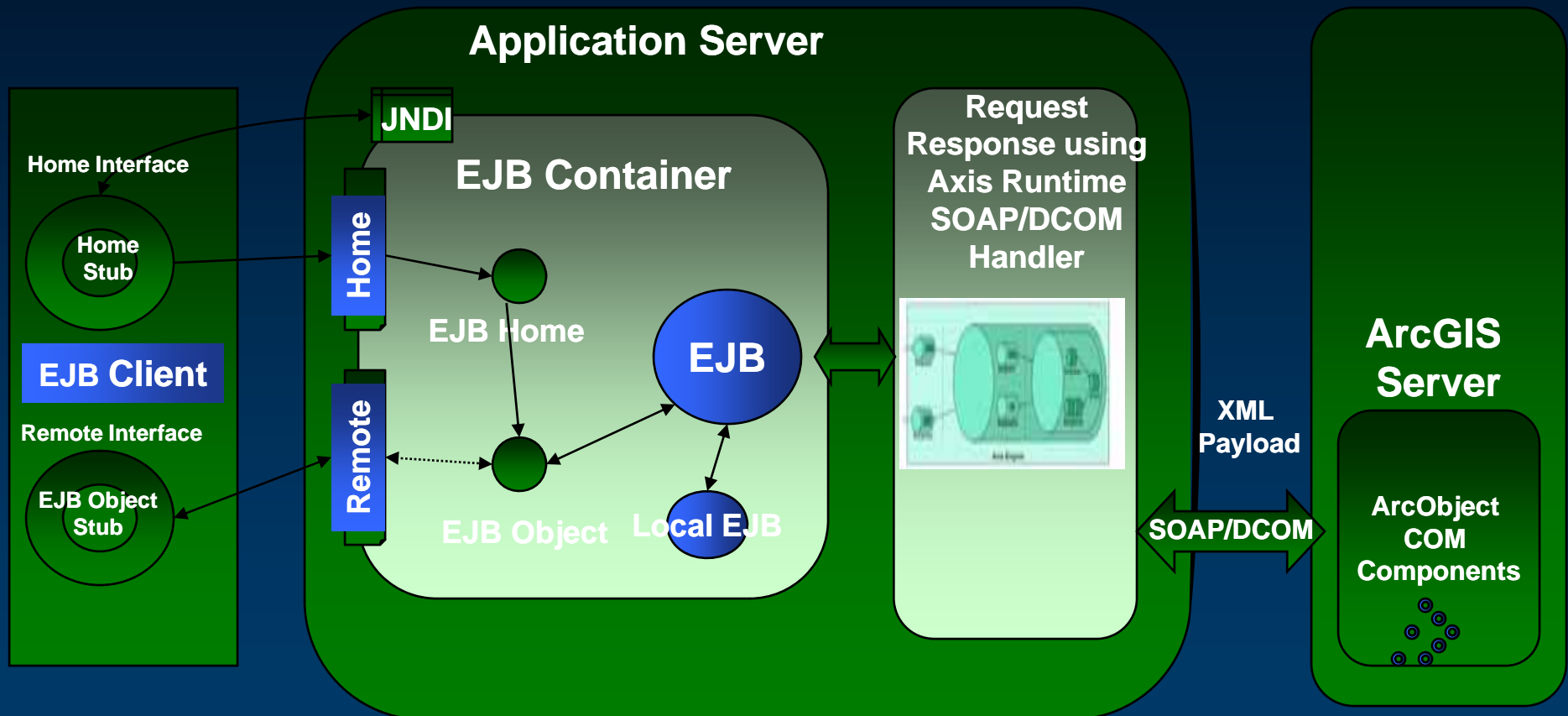


# Invoking GIS operations

Client Tier

EJB Tier

EIS Tier



# Certified J2EE 1.4 Containers



BEA WebLogic8.1 SP4



JBoss 4.0.2



OracleAS10.1.2 (10g Release 2)



IBM WebSphere 5.1



Sun (SJSAS) 8.1

# EJB Workflow

## ArcGIS Server 9.2

## EJB Container



Web Clients



3.

Configuration and deployment

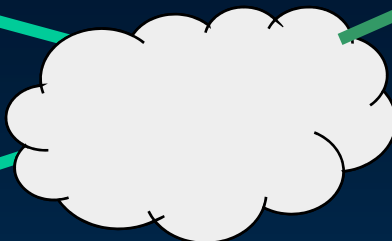


## ArcGIS Server

4.

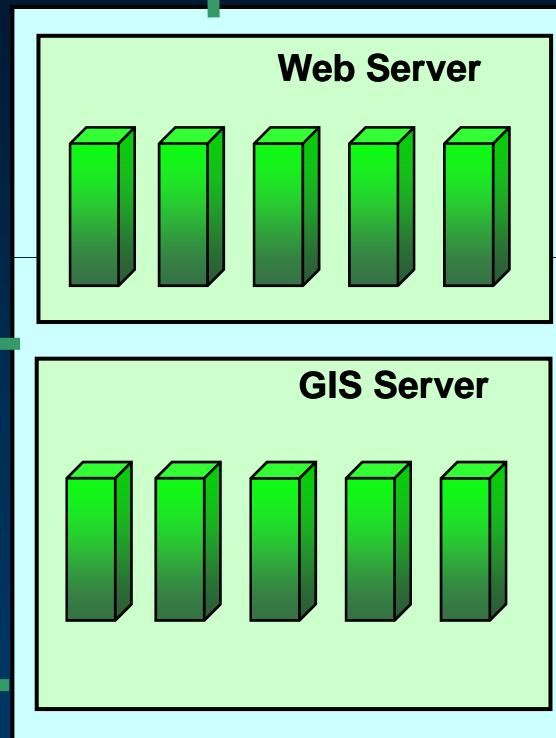
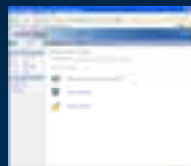


Rich Clients



2.

Manager or ArcCatalog Administration

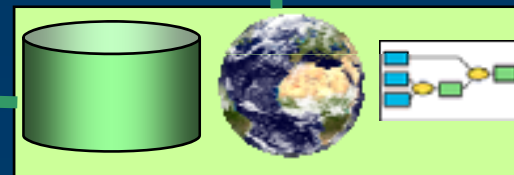


1.

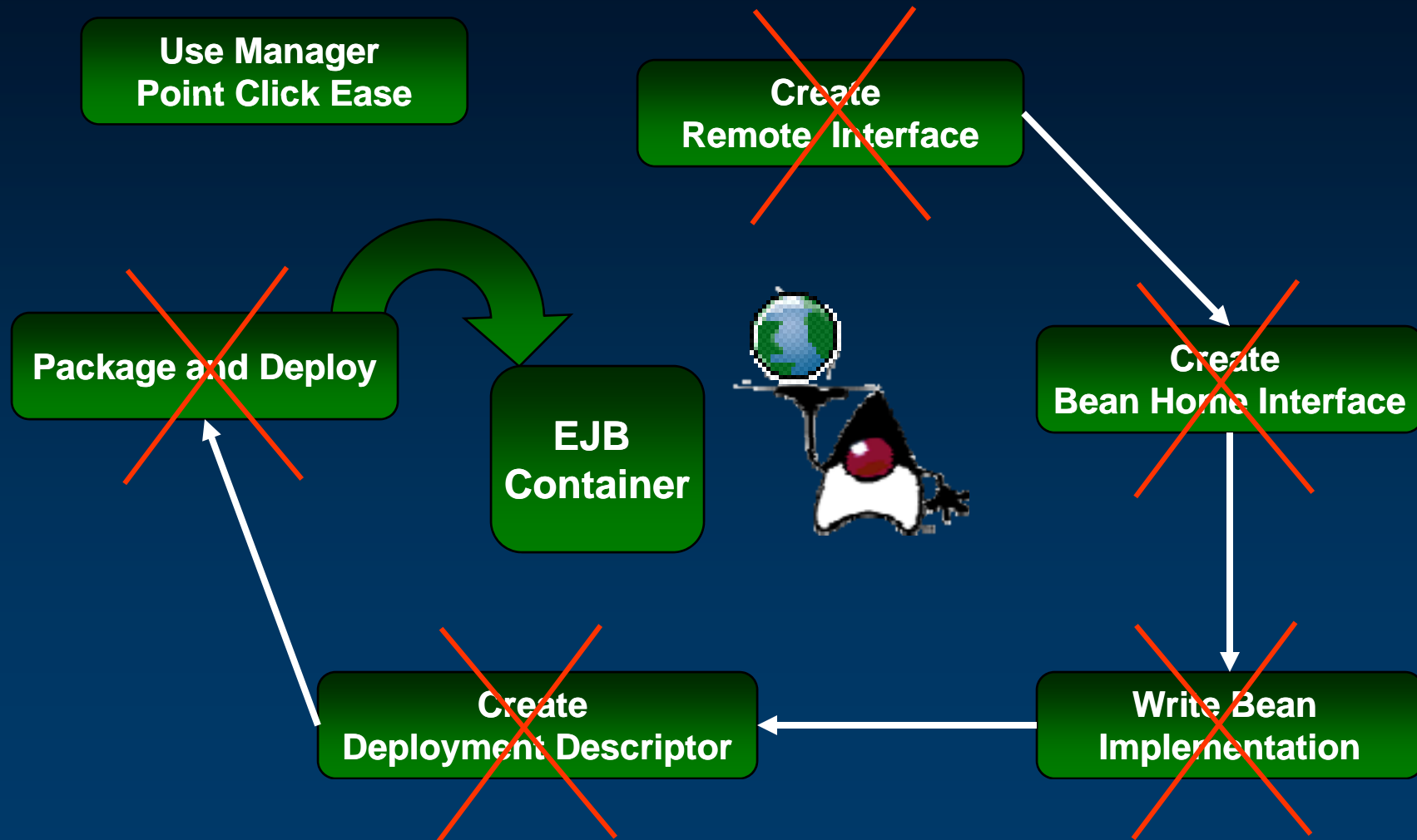
ArcMap, ArcGlobe  
Service Content  
Author



## Geodata



# Steps to Creating and Deploying EJBs (2.1)





# Building GeoSpatial EJBs with Manager



# Building GeoSpatial EJBs with Eclipse

*Using the ArcGIS Server Plugins*



# Programming Patterns and Scenarios

# EJB Business Case #1: **Getting Maps**

## Client code development tasks...

1. Get a reference to the **Map EJB** through standard JNDI lookup
2. Get the **MapDescription** object from the Map EJB
3. Create the **ImageDescription** and **ImageDisplay** objects and set the appropriate properties
4. Apply the **ImageDescription** and **ImageDisplay** objects to **MapDescription**
5. Get the **MapImage** object from the **Map EJB** using **MapDescription** object

# EJB Business Case #1: Getting Maps

```
// Get the Map EJB
mapContext = new InitialContext();
Object ref = mapContext.lookup(mapJNDIName);
MapEJBHome home = (MapEJBHome) PortableRemoteObject.narrow(ref,
    MapEJBHome.class);
mapEJB = home.create();
```

# EJB Business Case #1: Getting Maps

```
..
// Get the MapDescription
String defaultMapName = mapEJB.getDefaultMapName();
mapDescription = mapEJB.getServerInfo(defaultMapName).getDefaultMapDescription();

// Create and prepare the ImageDescription, ImageType and ImageDisplay
imageDescription = new ImageDescription();

ImageType type = new ImageType();
type.setImageFormat(EsriImageFormat.esriImagePNG);
type.setImageReturnType(EsriImageReturnType.esriImageReturnMimeData);

ImageDisplay display = new ImageDisplay(600, 600, 96);
imageDescription.setImageDisplay(display);
imageDescription.setImageType(type);

// Get the map image to be displayed in your client application..
MapImage mapImage = mapEJB.exportMapImage(mapDescription, imageDescription);
byte[] mimeTypeData = mapImage.getImageData();
..
```

## EJB Business Case #2: Querying Features within a FeatureLayer

### Client code development tasks...

1. Get a reference to the **Map EJB** through JNDI lookup
2. Get the **LayerDescription** object for the Layer to be queried. Use **MapDescription** to obtain it.
3. Define a **SpatialFilter** to select Features with. Use **PolygonN** and **PointN** objects.
4. Call **setWhereClause(String)** on the **SpatialFilter**
5. Define a selection symbol and set it to the Layer to be queried.
6. Export the MapImage

## EJB Business Case #2: Querying Features within a FeatureLayer

```
...  
  
// Create and set the appropriate values for the SpatialFilter  
  
SpatialFilter sf = new SpatialFilter();  
sf.setFilterGeometry(polygon);  
sf.setGeometryFieldName("SHAPE");  
sf.setWhereClause(""); //selects ALL features in the Layer  
sf.setSearchOrder(EsriSearchOrder.esriSearchOrderAttribute);  
sf.setSpatialRelDescription("");  
sf.setSpatialRel(EsriSpatialRelEnum.esriSpatialRelIntersects);  
  
...  
  
// #1: Query layer and set selected features  
FIDSet ifid =  
    mapEJB.queryFeatureIDs(mapdesc.getName(), layer0.getLayerID(), sf);  
layer0.setSelectionFeatures(ifid.getFIDArray());  
MapImage mapimg = mapEJB.exportMapImage(mapDesc, imageDesc);
```

## EJB Business case #3: **Reverse Geocode**

### Client code development tasks...

1. Define and set input parameters using ***PropertySet*** and ***PropertySetProperty[]***
2. Create the point to geocode (***PointN***). Make sure the correct ***SpatialReference*** is applied
3. Invoke the method call to the ***Geocode EJB*** using the ***PropertySet***
4. Process the results

# EJB Business case #3: Reverse Geocode

```
// Get a reference to the Geocode EJB..
InitialContext context = new InitialContext();
Object ref = context.lookup("ejb/esri/GCEjb/Geocode");
GeocodeEJBHome home =
    (GeocodeEJBHome) PortableRemoteObject.narrow(ref, GeocodeEJBHome.class);

geoEJB = home.create();
```

## EJB Business case #3: Reverse Geocode

```
// Define reverse GC inputs
PropertySet geocodePropSet = new PropertySet();
PropertySetProperty[] propArray = new PropertySetProperty[2];

PropertySetProperty geocodeProp = new PropertySetProperty();
geocodeProp.setKey("ReverseDistanceUnits");
geocodeProp.setValue("Meters");
propArray[0] = geocodeProp;

PropertySetProperty geocodeProp1 = new PropertySetProperty();
geocodeProp1.setKey("ReverseDistance");
geocodeProp1.setValue("100");
propArray[1] = geocodeProp1;

geocodePropSet.setPropertyArray(propArray);
```

# EJB Business case #3: Reverse Geocode

```
// Create point to reverse geocode
PointN inputPoint = new PointN();

inputPoint.setX(7649794.0);
inputPoint.setY(677525.0);

// Point must be in the same coordinate system as locator

SpatialReference resultsr = null;
Fields resultflds = geoEJB.getResultFields(geocodePropSet);

for(int i = 0;i < resultflds.getFieldArray().length;i++){
    Field fld = resultflds.getFieldArray()[i];
    if(fld.getType().getValue().equals("esriFieldTypeGeometry")){
        resultsr = fld.getGeometryDef().getSpatialReference();
    }
}

inputPoint.setSpatialReference(resultsr);
```

# EJB Business case #3: Reverse Geocode

```
// Do the Reverse Geocode and iterate through the results

PropertySet results =
    geoEJB.reverseGeocode(inputPoint, false, geocodePropSet);

PropertySetProperty[] resultsArray = results.getPropertyArray();

for (int i = 0; i < resultsArray.length; i++) {
    PropertySetProperty result = resultsArray[i];
    System.out.println(result.getKey().toString());
    System.out.println(result.getValue().toString());
    if (result.getValue() instanceof PointN) {
        PointN geocodePoint = (PointN) result.getValue();
        double x = geocodePoint.getX();
        double y = geocodePoint.getY();
        System.out.println(x + ", " + y);
    }
}
```

## EJB Business Case #4: Buffering Features by calling a GP Model

### Client code development tasks...

1. Get the tool names from the **GP EJB**
2. For each tool, get the **GPToolInfo** from the GP EJB
3. For each tool, get its **GPParameterInfos**
4. Each parameter requires a specific **GPValue** type. For each parameter, find out what the valid data type is
5. Set the appropriate values to each parameter for the tool to be executed
6. Add each newly populated GPParameterInfo value to a **GPValue[]** array
7. Submit the “Job” by calling the GP EJB to execute, passing the name of the tool and the GPValue[]. This can happen asynchronously
8. When the job finishes, do something with the result

# Get the EJB references

```
// Get the GP Bean reference
gpContext = new InitialContext();
Object ref = gpContext.lookup(gpJNDIName);
GeoprocessingEJBHome gpEJBHome =
    (GeoprocessingEJBHome)
    PortableRemoteObject.narrow(ref, GeoprocessingEJBHome.class);

gpEJB = gpEJBHome.create();

// Get the Map Bean reference
mapContext = new InitialContext();
Object ref = mapContext.lookup(mapJNDIName);
MapEJBHome home = (MapEJBHome) PortableRemoteObject.narrow(ref,
    MapEJBHome.class);
mapEJB = home.create();
```

# Set the parameters for the buffer model

```
// get the tool names
String[] toolNames = gpEJB.getToolNames();
String executeTool = toolNames[1]; // this is the buffer tool

// get toolinfo, parameter info and initialize the parameters
GPToolInfo toolInfo = gpEJB.getToolInfo(executeTool);
GPParameterInfo[] paramInfo = toolInfo.getParameterInfo();

// There are 2 parameters for this tool
GPValue[] values = new GPValue[paramInfo.length];
..
..
if (paramInfo[i].getDataType().equalsIgnoreCase("GPFeatureRecordSetLayer")) {
    GPFeatureRecordSetLayer gpFeatureRS = new GPFeatureRecordSetLayer();

    // this is the recordset of features to buffer...
    gpFeatureRS.setRecordSet(recordSet);

    // Set value for the FIRST parameter
    paramInfo[i].setValue(gpFeatureRS);
}
values[i] = paramInfo[i].getValue();
```

# Set the parameters for the buffer model

```
..  
..  
// Deal with the second parameter..  
if (paramInfo[i].getDataType().equalsIgnoreCase("GPLinearUnit")) {  
    GPLinearUnit lu = new GPLinearUnit();  
    lu.setUnits(EsriUnits.esriMiles);  
    lu.setValue(distance);  
  
    // Set the value for the SECOND parameter  
    paramInfo[i].setValue(lu);  
}  
  
values[i] = paramInfo[i].getValue();
```

# Execute the tool

```
// execute job and retrieve status
String jobID = gpEJB.submitJob(executeTool, values);

EsriJobStatus jobStatus = gpEJB.getJobStatus(jobID);

String status = jobStatus.getValue();

// wait till job execution succeeds
while (!status.equalsIgnoreCase(EsriJobStatus._esriJobSucceeded)) {
    Thread.sleep(1000);
    jobStatus = gpEJB.getJobStatus(jobID);
    status = jobStatus.getValue();

    if (status.equalsIgnoreCase(EsriJobStatus._esriJobFailed)) {
        break;
    }
}

if (!status.equalsIgnoreCase(EsriJobStatus._esriJobSucceeded)) {
    jobID = null;
}

return jobID;
```

## EJB Business Case #5:

### Solve a simple route between 2 stops and generate directions

Client code development tasks...

1. Get references to **Geocode** and **NA** EJBs
2. Geocode “start” and “end” addresses
3. Get the **PointN** for each (to be used as the “stops”)
4. Create **NAServerRouteParams** and
  - add the “stops” to it
  - Set “Return Directions” to be TRUE
  - Set “Direction Distance Units”
  - Set “Impedance” (Minutes? Meters?)
5. Call **solve (NAServerRouteParams)** on NA EJB
6. Process the results (**NAServerRouteResults** is returned from Solve)

# Geocode the “start” and “end” addresses

```
//Geocode STARTING address
Field[] addressFields = geoEJB.getAddressFields().getFieldArray();
PropertySetProperty[] addressComponents = new PropertySetProperty[]{new
    PropertySetProperty(), new PropertySetProperty()};

addressComponents[0].setKey(addressFields[0].getName());
addressComponents[0].setValue(startAddress);
addressComponents[1].setKey(addressFields[1].getName());
addressComponents[1].setValue(startZone);
PropertySet address = new PropertySet(addressComponents);

PointN startPoint = null;
PointN endPoint = null;

PropertySet geocodedAddress = geoEJB.geocodeAddress(address, null);
PropertySetProperty[] geocodedComponents = geocodedAddress.getPropertyArray();

for(int i=0; i<geocodedComponents.length; i++){
    if(geocodedComponents[i].getKey().equals("Shape")){
        startPoint = (PointN)geocodedComponents[i].getValue();
        break;
    }
}
// DO THE SAME FOR THE ENDING ADDRESS
```

# Prepare the Route Parameters...

```
//Find route. First, set the Route Parameters
NAServerRouteParams routeParams =
    (NAServerRouteParams)naEJB.getNASolverParameters(routeLayer);

NAServerPropertySets stops = new NAServerPropertySets();
PropertySet[] pitStops =
    new PropertySet[] {new PropertySet(), new PropertySet()};

pitStops[0].setPropertyArray(new PropertySetProperty[] {
    new PropertySetProperty("Name", startAddress),
    new PropertySetProperty("X",
        String.valueOf(startPoint.getX())),
    new PropertySetProperty("Y", String.valueOf(startPoint.getY()))});

pitStops[1].setPropertyArray(new PropertySetProperty[] {
    new PropertySetProperty("Name", endAddress),
    new PropertySetProperty("X",
        String.valueOf(endPoint.getX())),
    new PropertySetProperty("Y", String.valueOf(endPoint.getY()))});

stops.setPropertySets(pitStops);
routeParams.setStops(stops);
routeParams.setReturnDirections(true);
```

# Solve the Route...

```
...
```

```
// Solve Route
```

```
NAServerRouteResults results =  
    (NAServerRouteResults)naEJB.solve (routeParams);
```

```
// Iterate through the resulting direction components (Text directions, etc...
```

```
NAStrreetDirections[] directions = results.getDirections();
```



# Network Analyst EJB Demo

# Integration with Web ADF

- The Java Web ADF API includes:
  - **EJBMapResource**
  - **EJBGeocodeResource**
- To add an EJB as a Map or Geocode resource to your ADF Web Application:
  - Modify the **WEB-INF/faces-config.xml**
  - Include a “managed-bean” for each EJB
  - Define “managed properties” for jndiNames and any RMI specific key-values (specific for each different application server vendor)

# WEB-INF/faces-config.xml

```
<!-- Managed bean for ArcGIS Server EJB resource [START] -->
<managed-bean>
  <managed-bean-name>agsejb2</managed-bean-name>
  <managed-bean-class>com.esri.adf.web.ejb.data.EJBMapResource</managed-bean-class>
  <managed-bean-scope>none</managed-bean-scope>
  <managed-property>
    <property-name>jndiNames</property-name>
    <map-entry>
      <key>map</key>
      <value>ejb/esri/Enterprise/Frisco/Map</value>
    </map-entry>
    ...
  </managed-property>
  <managed-property>
    <property-name>jndiConnectionEnvProperties</property-name>
    <!-- AGSEJB APPSERVER SPECIFIC ENTRIES [START] -->
    ...
  </managed-property>
  ...

```

# Application Deployment

- **EAR deployments**
  - **Most common deployment mechanism for all application servers**

# EJB Security

- **The Geospatial EJBs in the Enterprise ADF can take advantage of J2EE standard security mechanism**
- **2 Security aspects:**
  - **Authentication** – is the requestor really who they say they are?
  - **Authorization** – is the requestor really allowed to perform the task they are asking to perform?
- **Authorization policies are made declaratively within application deployment descriptors**
  - Bean developers are not required to implement in code
  - Some vendors provide tools to create these

# EJB Security

## Example: Securing a Geocode EJB at the method level

### Steps:

1. Define a security **realm** for the EJB
2. Define security **roles**
3. Define method **permissions**
4. Map the roles to users/groups
5. Edit the `ejb-jar.xml` for your EJB module – Declare the security role for your specific methods
6. Using your application server's specific administrator, set up the security realm, roles for that realm, and users belonging to the roles.

# EJB Security

## Edits to the `ejb-jar.xml`...

```
<!-- Define an abstract security role, reverse-geocoders: ->
<ejb-jar>
  <enterprise-beans>
    ...
  </enterprise-beans>
  <assembly-descriptor>
    <security-role>
      <description>Users allowed to reverse geocode</description>
      <role-name>reverse-geocoders</role-name>
    </security-role>
  <!-- Restrict access to reverseGeocode method: ->

    <method-permission>
      <description>Restricting access to reverseGeocode() </description>
      <role-name>reverse-geocoders</role-name>
      <method>
        <ejb-name>GeocodeEJB</ejb-name>
        <method-name>reverseGeocode</method-name>
      </method>
    </method-permission>
  </assembly-descriptor>
</ejb-jar>
```

# Summary

## **ArcGIS Server Application Development Framework**

- ✓ **supports development of J2EE applications powered by EJB technology**
- ✓ **ships with out-of-the-box Geospatial EJBs ready to use, and ready to deploy across a wide variety of J2EE 1.4 certified application servers**
- ✓ **supports EJB development within Manager application for simple point click ease**
- ✓ **integrates with other application developer framework components such as Web ADF, and Eclipse IDE**
- ✓ **ships with Advanced Edition of ArcGIS Server 9.2**

# Developer Resources

- <http://edn.esri.com/java> - “JavaCentral”