J2EE Architecture & Design
2° session

Angela Fogarolli

Angela.Fogarolli@apss.tn.it

J2EE Patterns in dept

- Presentation Tier
  - Front Controller
  - Intercepting Filter

  Business Tier
  - Session Facade
  - Business Delegate Pattern
  - Service Locator Pattern
- Persistence Tier
  - Data Access Object Pattern
  - Version Number (Transaction Patterns)

Patterns….

- Singleton
- Service Locator

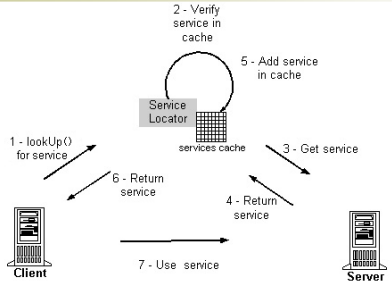
Singleton Pattern

```
public class MySingleton {
      private static  MySingleton _instance;

      private MySingleton() {
      // construct object . . .
      }

      // For lazy initialization
      public static synchronized MySingleton getInstance() {
      if (_instance==null) {
      _instance = new MySingleton();
      }
      return _instance;
      }
      // Remainder of  class definition . . .
      }
```

Cache to itself

if the cache is empty
it creates an istance

Service Locator Pattern



…to acquire an EJB Home object for the first time, Service Locator first creates a JNDI initial context object and performs a lookup on the EJB Home object.

But not only for EJB Home also for other kinds of services…an enhanced Service Locator can have a verifier mechanism that checks the cached services' validity by monitoring them during a cycle of a specified frequency.

Service Locator Pattern

Forces

- Service lookup and creation involves complex interfaces and network operations
- Service lookup and creation operations are resource intensive and redundant

Service Locator Pattern

Solution

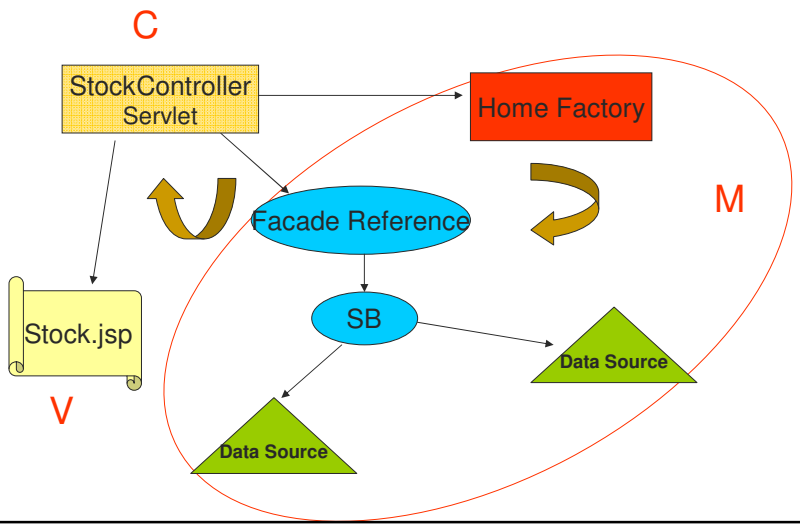
- Use a Service Locator to
  - Abstract naming service usage
  - Shield complexity of service lookup and creation
  - Promote reuse
  - Enable optimize service lookup and creation functions
- Usually called within Business Delegate or Session Facade object

Review Example

- Web Tier
  - MVC
  - Front Controller
  - Service Locator
  - Home Factory + singleton
  - DTO
- EJB Tier
  - Session facade
  - Service Locator
  - Home Factory + singleton
  - DTO
  - Data Access Object Pattern

The applied patterns can be almost the same in both layers.

Review Example



Bean Manager

Model (MVC)

Singleton + Home Factory

```
public class BeanManager {
   private  LoginManager lm;
   private  StockManager sm;
   private static BeanManager beanManagerSingleton;
   public BeanManager() {
     if (lm == null) {
        try {
           LoginManagerHome lmHome = EJBGetter.getLoginManagerHome();
           lm = (LoginManager) lmHome.create();
        } catch (RemoteException ex) {
           System.out.println("Couldn't create loginManager bean." +
                       ex.getMessage());
        } catch (CreateException ex) {
           System.out.println("Couldn't create customer bean." +
                       ex.getMessage());
        } catch (NamingException ex) {
           System.out.println("Unable to lookup home: " +
                       CodedNames.LOGIN_MANAGER_EJBHOME +
                       ex.getMessage());
        }
     }
```

Calling the service locator

Bean Manager

Calling the service locator

```
if (sm == null) {
        try {
          //lookUP Home Interface
           StockManagerHome smHome = EJBGetter.getStockManagerHome();
           sm = (StockManager) smHome.create();
        } catch (RemoteException ex) {
           System.out.println("Couldn't create stockManager bean." +
                        ex.getMessage());
        } catch (CreateException ex) {
           System.out.println("Couldn't create stockManager bean." +
                        ex.getMessage());
        } catch (NamingException ex) {
           System.out.println("Unable to lookup home: " +
                        CodedNames.STOCK_MANAGER_EJBHOME +
                        ex.getMessage());
        }
     }
    }
```

Bean Manager

```
private static BeanManager getFactory(){
    if (beanManagerSingleton==null){
       BeanManager.beanManagerSingleton= new BeanManager();
    }
    return beanManagerSingleton;
  }

  public LoginManager getLoginManager() {
     return lm;
  }

  public StockManager getStockManager() {
     return sm;
  }

  public void destroy() {
     sm = null;
     lm = null;
  }
}
```

Model (MVC)

EJBGetter: Service Locator

```
public final class EJBGetter {

   public static LoginManagerHome getLoginManagerHome()
      throws NamingException {

      InitialContext initial = new InitialContext();
      // lookup of the remote Session Bean
      Object objref = initial.lookup(CodedNames.LOGIN_MANAGER_EJBHOME);
      return (LoginManagerHome)
         PortableRemoteObject.narrow(objref, LoginManagerHome.class);
   }

   public static StockManagerHome getStockManagerHome()
      throws NamingException {
      InitialContext initial = new InitialContext();
      // lookup of the remote Session Bean
      Object objref = initial.lookup(CodedNames.STOCK_MANAGER_EJBHOME);
      return (StockManagerHome)
         PortableRemoteObject.narrow(objref,StockManagerHome.class);
   }
}
```

Controller (MVC)

Controller

```
public class StocksController extends HttpServlet {

public void doGet(HttpServletRequest request,HttpServletResponse response)
      throws ServletException, IOException
   {
      doPost(request, response);
   }
 public void doPost(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
      HttpSession session = request.getSession();
// checking the URL
      String selectedScreen = request.getServletPath();
      request.setAttribute("selectedScreen", selectedScreen);
      Integer userInt=(Integer)getServletContext().getAttribute("userID");
      int userId = 0;
      if(userInt != null){
       userId= userInt.intValue();
      }
      if (userId == 0){
         try {
            request.getRequestDispatcher("/index.jsp").forward(request, response);
         } catch (Exception e) {}
      }
      BeanManager bm=  BeanManager.getFactory();
      StockManager sm=bm.getStockManager();
      LoginManager lm=bm.getLoginManager();
```

Calling the JSP

Controller


```
if (selectedScreen.equals("/stocks")) {
        try {
        //calling the MODEL: StockManager
        List stocklist= sm.getStocks();
        request.setAttribute("quotations", stocklist);
         request.getRequestDispatcher("/stocks.jsp").forward(request, response);
        }catch (Exception e) {
         }
    }

    if (selectedScreen.equals("/portfolio")) {
        try {
        //calling the MODEL: StockManager
        List stocklist= sm.getPortfolio(userId);
        request.setAttribute("portfolio", stocklist);
         request.getRequestDispatcher("/portfolio.jsp").forward(request,
  response);
        }catch (Exception e) {
    }

    }
```

Calling the JSP

Calling the JSP

StockManager
Session Facade

```
public class StockManagerBean implements SessionBean
{
    private  SessionContext sessionContext = null;
     public List getStocks()
     {
       List stockList=null;
        try
        {                          //Trader is a session bean
                TraderLocalHome traderHome=
EJBGetter.getTraderLocalHome();
                TraderLocal traderObj = (TraderLocal)
traderHome.create();
                stockList=traderObj.getStocks();

        } catch (NamingException re)
        {
        } catch (Exception exception)
        {
                exception.printStackTrace();
        }finally{
                //Return a List of DTO objects
                return stockList;
        }
 }
```

Calling a Service Locator

Use another Locator for the Local interfaces … …you could also use factory + singleton pattern…

Trader

```
public class TraderBean implements SessionBean {

   private SessionContext sessionContext = null;

   public void ejbCreate() throws javax.ejb.CreateException {
   }

   public List getStocks() throws TraderDAOExcep {
      TraderDAOFactory traderDAOF = new TraderDAOFactory();
      TraderDAO traderDAO = traderDAOF.getDAO();
      return traderDAO.getStocks();
   }
}
```

call to data source
Using DAO Pattern

Stocks.jsp: View

View (MVC)

```
<FORM METHOD='POST' ACTION="buystock">
     <TABLE ALIGN=CENTER BORDER=0>
     <TR bgColor=#ffffff>
          <TD colspan=5 height=20 style="MARGIN-LEFT: 1px" bgColor=#425a9c
     class=TitoloTabella2>    STOCK'S QUOTATIONS</TD>
     </TR>
     <TR bgColor=#ffffff height=5>
     </TR>
     <TR bgcolor=#e5e9f5>
          <TD>Id</TD><TD>Stock</TD><TD>Value</TD> <TD>MinimumPackage</TD><TD>Buy</TD></TR>
<%
      List list = (List) request.getAttribute("quotations");
      Iterator i = list.iterator();
      int j = 0;
      String color = "w";
      while (i.hasNext()) {
        StocksTrader stock = (StocksTrader) i.next();
        String idstock = stock.getId();      %>
        <TD align=left class=testote>
          <input type="hidden" name="stockId" value>
<%=idstock%>        </TD>
      <%String desc = idstock + "description";      %>
        <TD align=left class=testote>
          <input type="hidden" name=desc value>
<%=stock.getDescription()%>        </TD>
      <%String val = idstock + "value";      %>
        <TD align=left class=testote>
          <input type="hidden" name=val value>
<%=stock.getValue()%>        </TD>
```

Map in web.xml

DTO

getting returned value

Implementing MVC: web.xml

```
<servlet-mapping>
    <servlet-name>StocksController</servlet-name>
    <url-pattern>/buystock</url-pattern>
</servlet-mapping>

JSP code:
<FORM METHOD='POST' ACTION="buystock">
```

Presentation patterns….

- Front Controller
- Intercepting Filter

Front Controller

Forces

- Multiple views are used to handle a similar business request
  - – View is required to handle logic for content retrieval and navigation
- Common system services are rendered to each request
  - – Example: Authentication, authorization, Logging

Front Controller

Solution

- Use a controller as an centralized point of contact for all requests
  - - Promote code reuse for invoking system services
- Can have multiple front controllers, each mapping to a set of distinct services
- Works with other patterns Command, Dispatcher, View Helper

Intercepting Filter

Forces

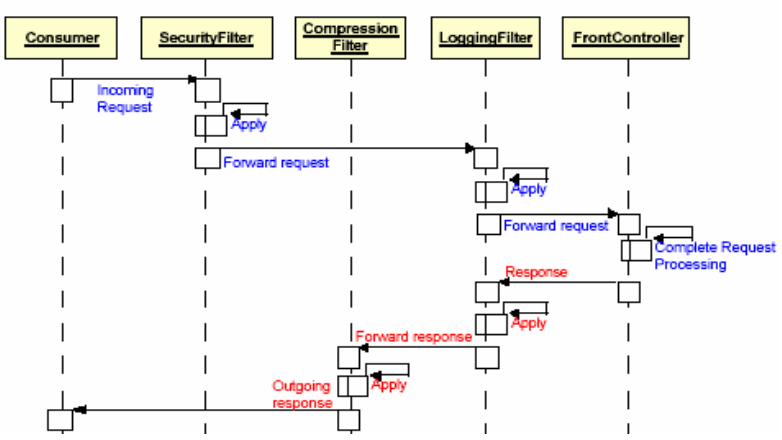
- Each service request and response requires common pre-processing and post-processing
  - – logging, authentication, caching, compression,data transformation
- Adding and removing these “pre” and “post” processing components should be flexible
  - – deployment time installation/configuration

Intercepting Filter

Solution

- Create pluggable and chainable filters to process common services such that
  - – Filters intercept incoming and outgoing requests and responses
  - – Flexible to be added and removed without requiring changes to ther processing code
- Examples
  - – Servlet filters for HTTP requests/responses
  - – Message handlers for SOAP requests/responses

Intercepting Filter Pattern



Web Application Frameworks

- Based on MVC Model architecture
- Web-tier applications share common set of functionality
  - Dispatching HTTP requests
  - Invoking model methods
  - Selecting and assembling views
- Provide classes and interfaces that can be used/extended by developers

Why Web Application Framework?

- De-coupling of presentation tier and business logic into separate components
- Provides a central point of control
- Provides rich set of features
- Facilitates unit-testing and maintenance
- Availability of compatible tools
- Provides stability
- Enjoys community-supports
- Simplifies internationalization
- Simplifies input validation

Web Application Frameworks

- Apache Struts
- Sun ONE Application Framework
- JavaServer Faces (JSR-127)
  - Standard-based Web application framework

More EJB Tier patterns

- Business Delegate
- DAO
- Version Number

Business Delegate Pattern (similar Command Pattern)

Forces

- Business service interface (Business service APIs) change as business requirements evolve
- Coupling between the presentation tier components and business service tier (business services) should be kept to minimum
- It is desirable to reduce network traffic between client and business services

Business Delegate Pattern

Solution

- Use a Business Delegate to
  - Reduce coupling between presentation-tier and business service components
  - Hide the underlying implementation details of the business service components
  - Cache references to business services components
  - Cache data
  - Translate low level exceptions to application level exceptions
  - Transparently retry failed transactions
  - Can create dummy data for clients

Business Delegate is a plain java class

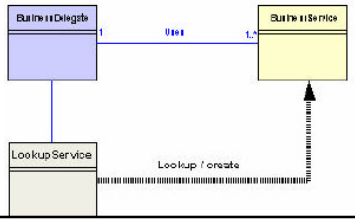
Service Locator Pattern

A Business Delegate uses a component called the Lookup Service.

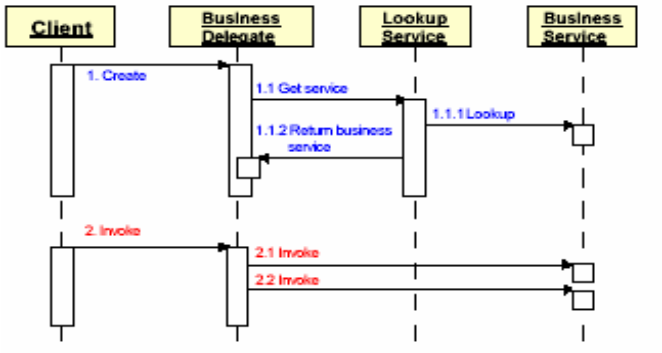
The Lookup Service is responsible for hiding the underlying implementation details of looking up a business service.

The lookup service can be written as part of business delegate however it is recommended to create a separate component.

If a facade exists, there will be a 1-1 relationship between business delegate and session facade.



Business Delegate & Service Locator Pattern

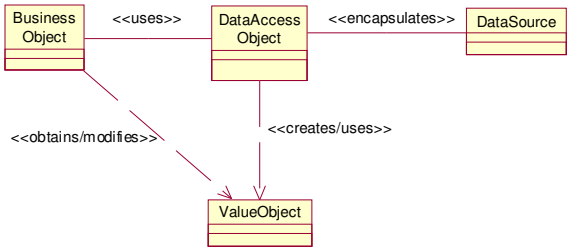


Business Delegate Implementation Strategies

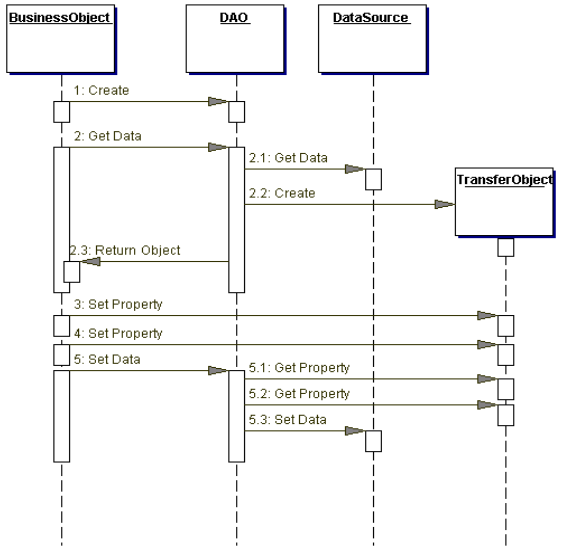
- Delegate Adapter Strategy
  - Integrating two disparate systems require an Adapter
  - Adaptor changes XML request to native request
  - Perfectly fits for B2B environments that use XML
- Delegate Proxy Strategy
  - Business Delegate proxy to the Session bean it is encapsulating
  - May cache necessary data such as home or remote object handles to improve performance

DAO Pattern

A pattern for encapsulating access to a data source can be used as the persistence component for BMP entity beans or directly from a session facade



DAO Pattern

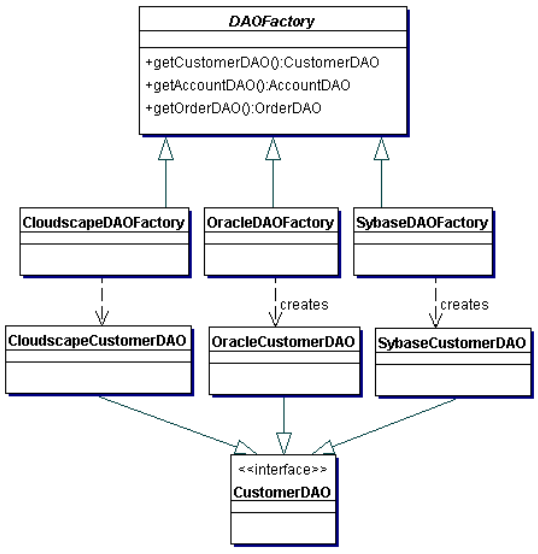


DAO Pattern

- **Enables Transparency**
- **Enables Easier Migration**
- **Centralizes All Data Access into a Separate Layer**
- **Adds Extra Layer**

The benefit realized by choosing this approach pays off for the additional effort.

DAO Example



DAOFactory
using abstract Factory

```
public abstract class DAOFactory {

  // List of DAO types supported by the factory
  public static final int CLOUDSCAPE = 1;
  public static final int ORACLE = 2;
  public static final int SYBASE = 3;

  // There will be a method for each DAO that can be created. The concrete factories
will have to implement these methods.
  public abstract CustomerDAO getCustomerDAO();
  public abstract AccountDAO getAccountDAO();
  public abstract OrderDAO getOrderDAO();

public static DAOFactory getDAOFactory(
      int whichFactory) {

    switch (whichFactory) {
      case CLOUDSCAPE:
          return new CloudscapeDAOFactory();
      case ORACLE    :
          return new OracleDAOFactory();
      case SYBASE    :
          return new SybaseDAOFactory();
      ...
      default        :
          return null;
    }}}
```

Concrete DAOFactory
Implementation for Cloudscape

```
public class CloudscapeDAOFactory extends DAOFactory {
  public static final String DRIVER="COM.cloudscape.core.RmiJdbcDriver";
  public static final String DBURL="jdbc:cloudscape:rmi://localhost:1099/CoreJ2EEDB";

  // method to create Cloudscape connections
  public static Connection createConnection() {
    // Use DRIVER and DBURL to create a connection and a connection pool
}
  public CustomerDAO getCustomerDAO() {
    // CloudscapeCustomerDAO implements CustomerDAO
    return new CloudscapeCustomerDAO();
  }
  public AccountDAO getAccountDAO() {
    // CloudscapeAccountDAO implements AccountDAO
    return new CloudscapeAccountDAO();
  }
  public OrderDAO getOrderDAO() {
    // CloudscapeOrderDAO implements OrderDAO
    return new CloudscapeOrderDAO();
  }}
```

Cloudscape DAO

Implementation for Customer

```
public class CloudscapeCustomerDAO implements CustomerDAO {

  public CloudscapeCustomerDAO() {
    // initialization
  }

  // The following methods can use CloudscapeDAOFactory.createConnection()  to get a
//connection as required

  public int insertCustomer(...) {
    // Implement insert customer here.
    // Return newly created customer number
    // or a -1 on error
  }

    public Customer findCustomer(...) {
    // Implement find a customer here using supplied argument values as search criteria
    // Return a Transfer Object if found, return null on error or if not found
  }

  }
```

This class can contain all Cloudscape specific code and SQL statements. The client is thus shielded from knowing these implementation details.

Customer Transfer Object

```
public class Customer implements java.io.Serializable {
  // member variables

  int CustomerNumber;
  String name;
  String streetAddress;
  String city;
  ...

  // getter and setter methods...
  ...
}
```

Client Code

Using a DAO and DAO Factory

```
// create the required DAO Factory
DAOFactory cloudscapeFactory =  DAOFactory.getDAOFactory(DAOFactory.DAOCLOUDSCAPE);

// Create a DAO
CustomerDAO custDAO =  cloudscapeFactory.getCustomerDAO();

// create a new customer
int newCustNo = custDAO.insertCustomer(...);

// Find a customer object. Get the Transfer Object.
Customer cust = custDAO.findCustomer(...);

// modify the values in the Transfer Object.
cust.setAddress(...);
cust.setEmail(...);

// update the customer object using the DAO
custDAO.updateCustomer(cust);

// delete a customer object
custDAO.deleteCustomer(...);
```

Version Number:

Optimistic locking

How can you determine if the data used to update the server is stale?

Problem:

- A reads Message X in a transaction
- B reads Message X in a transaction
- A performs local updates on his copy of the Message
- B performs local updates on her copy of the Message
- A updates Message X in one transaction
- B updates Message X in one transaction

Once step 6 occurs all updates willl be overwrittern by those changes made by B.

After step 5 any copies of the message help by other clients is said to be stale since it no longer reflects the current state of the Message entity bean

Version Number:
Optimistic locking

Solution:

Use version numbers to implement your own staleness checks in entity beans

- Carry the version number along with any other data read from an entity bean during read transaction.
- Send the version number back to the entity bean along with any updated data.
- Increment the entity bean's verion number when performing an update.
- Reject the update if the version numbers do not match.

JDBC for Reading

When should a session facade perform direct database access instead of going through the entity bean layer?

- Bulk loading.
  - The n + 1 BMP database calls problem (1 finder + n ejbLoad());
  - Performance in navigating EJB relationships.

Trade Offs:

- Tight coupling between business and persistence logic.
- Bug prone and less maintainable.

DAO Pattern can be used to alleviate this problem.

Conclusion

" There are many decisions that must be made for a successful J2EE implementation "

Key patterns for J2EE:

- session façade
- message façade
- service locator
- value object / data transfer object
- data access object

Use a combination of time tested patterns and best practices