# Design & Implementation Guide by Jeff Zhuk

JavaSchool.com

Purpose: Improve quality and governance of *web applications*

The guide promotes best practices in architecture, design, and coding. The focus is on design patterns and practical samples of data and service layers. Extracts from the book, multiple projects, and training sessions, as well as reader's feedback notes are used below.

Structure: The document consists of topics related to web application development in Java. Each topic describes a problem and a recommended solution in a manner similar to design pattern descriptions looking at a problem from integration and SOA perspectives.
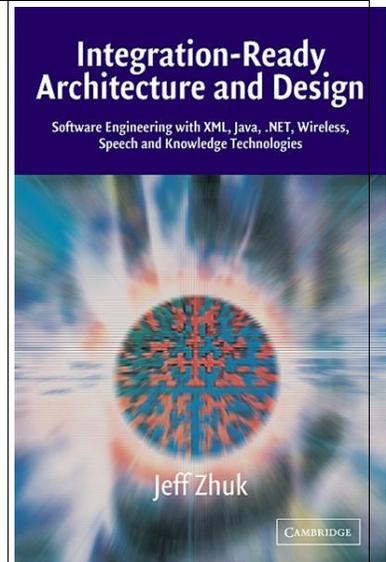
Related subjects: Best practices in Software and Knowledge engineering

Contents

Authentication, Identification and Authorization Solutions for Web and Client-Server applications

Problem

The title tasks look very common although for different development units and individuals they might mean a little bit different things.
This is just one small problem, which become bigger for bigger and faster growing organizations.

On the top of this problem, there are tactical choices of centralized and decentralized solutions for these tasks and on the top of that one there are issues of multiple data sources and related maintenance problems.

When multiple units in a big organization provide different types of design and implementation solutions to these problems, the problems tend to rapidly grow.

Solution

There is no silver bullet, which works for all in this area, except to stick to a single design and implementation policy based on best practices.

Let's start with definitions and then describe best practices.
1. **Authentication** – confirms claimed identity of a person or an application.
1.1 The Authentication Service takes, for example, a persona credentials, like login/email and password and validates these credentials.
1.2 **Best Practices**: for big organizations with multiple applications and multiple data sources it's crucial to minimize code pieces (best have just one!) and related maintenance efforts dedicated to the Authentication Service. Ideally a single service should be used by all applications. This single service will be able to work with all necessary data sources, so applications will no need to know about data sources details. This would also allow change data sources details without changing application details related to authentication.

2. **Identification** – recognizes a person or an application with their role (appearance) in the process.
2.1 The Identification Service takes, for example, user's login or email and application name and returns user's role for this application.
2.2 **Best Practices**: for big organizations with multiple applications and multiple data sources it's crucial to minimize code pieces (best have just one!) and related maintenance efforts dedicated to the Identification Service. Ideally a single service should be used by all applications. This single service will be able to work with all necessary data sources, so applications will no need to know about data sources details. This would also allow change data sources details without changing application details related to identification.

2.3 **Best Practices**: The Identification Service can inherit some functionality of the Authentication Service, especially if authentication data sources include application roles and user profiles. In this cases it is beneficial to integrate both services providing authentication and identification functionality at once. For example such integrated service can provide the following functions:

2.3.1   Validate a user (Authentication)

2.3.2   Get user's role for an application

2.3.3   Change user's role for an application (Create and Delete User for this application might be also done via the "Change user's role" function)

2.3.4   Get user's profile

2.3.5   Usually functions like "Change user's password", "Create and Change user's profile" are under a strict HR control and implemented by separate efforts. It's still desirable to add these functions to authentication and identification services.

3. **Authorization** - allows access to resources only to those permitted to use them. There could be different types of access based on different application roles.

3.1 **Centralized and Decentralized Authorization** – both implementation types are valid solutions.

3.1.1   **The Centralized Authorization**, for example, is recommended to authorize access to enterprise services and applications.

The Centralized Authorization, also known as Service Access Security, is described in the **Service Access Security** section of this design guide. The Centralized Authorization solution is usually based on an XML configuration file which is located somewhere in a central repository and includes a table of roles and related services and applications. To prevent hacking into this file, the file should include some dynamic information

3.1.2   **The Decentralized Authorization** is recommended to address privilege-based access inside services and applications.

For example, the service A must offer one set of functions to the role X and another set of functions to the role Y. We'll build in the service A (a decentralized solution!) ability to present two different set of functions based on the role passed as a parameter. We'll implement this ability by calling a proper class (AX or AY) for a server action or passing control to a proper (AX.jsp or AY.jsp) web page.

One important issue is to **escape "if/else" conditional code** in the invocation section. We don't want the invocation code to change each time when a set of roles is changed. Instead we'll use the **Factory Method Design Pattern** which allows us to add or delete new sets of functions without changing the invocation code.

Server side example:

Bad invocation code	Good invocation code

if(role.equals("X")) {
       AX ax = new AX();
       ax.run();
} else if(role.equals("Y")) {
	AX extends A; // A is an abstract base class
	….
	AY extends A; // A has common methods for AX and AY

AY ay = new AY();
    ay.run();
} else if(role.equals("Z")) {
    AZ az = new AZ();
    az.run();
}
// … etc ….. code will grow with more roles
	….
	// Invocation code
	A a = (A) Class.forName("A" + role).newInstance();
	
	a.run();
	
	// no more code, no more changes, stable API for all current and future roles!

Example for web applications with STRUTS frameworks where invocation code will pass control to a proper JSP page

In both we pass the role as a parameter and we'll need to map proper JSP pages in the struts-config.xml file, but only the second example allows us to add or delete proper pages without changing the central invocation place.

Bad invocation code inside a STRUTS action	Good invocation code inside a STRUTS action

if(role.equals("X")) {
    return mapping.findForward("AX");
} else if(role.equals("Y")) {
    return mapping.findForward("AY");
} else if(role.equals("Z")) {
    return mapping.findForward("AZ");
}
// … etc ….. code will grow with more roles
	return mapping.findForward("A" + role);
	
	// no more code, no more changes, for all current and future roles!

Security Guidelines: Service Access Security

Problem

Service-Oriented Architecture shift development focus from applications to services. Multiple applications can call the same services instead of copy/paste/modify their code. The problem is that exposed services can be called not only by legitimate applications. Being outside of application umbrella, exposed services need secure protection.

Solution

Service request must include secure identification of an application requested a service. Similar to a way as we secure user access to applications, we must secure application access to services. It is recommended to provide a Security Guard that can protect multiple services. Each application (a service consumer) will register with the Security Guard and receive back its secure time-stamped application ticket and dynamic association with the IP address of the requestor.

Service Security

Validate user session ticket

Register (when start) providing appName and appPassword pair

Application

Security Guard

Receive a secure application ticket

Validate application ticket and invoke the service

Request service using the application ticket as an argument (to identify/secure application access) besides other arguments

Service

Request application service (providing user session ticket with embedded roles). Application will check the session ticket with security layer and check role-based access before sending service request. The service request will include the application ticket.

Login to a multiple-apps system with name/password via SSL

User

Receive a secure user session ticket with (embedded) user roles info

Security Guidelines: Web Access Security

Problem

Web applications are exposed to internal and external users. There are multiple ways for hackers to try to access corporate data. They use software that can track information coming in and out of the web site. There are software tools that try to mimic user authentication behavior, and etc.

Solution

Multiple layers of security provide better security. Using SSL over HTTP we ensure that all messages are encrypted. Then, all web users will access applications with the URL that starts with "HTTPS:// ". This URL is served by a specific port that supports SSL communications. (See References)

Another layer is to protect users from authentication fraud by establishing rules for password encryption and change password functionality. Working in Java environment, it is recommended to use Java encryption library and proven encryption mechanisms, versus homegrown encryption algorithms.

One more layer is to ensure that only necessary data are coming out of the corporate web site and periodically analyze network traffic.

It is recommended to use hidden fields instead of long URLs and keep most of session data on the server side exposing minimum information related to session authentication to the web page.

For more information on web application security please look at The Open Web Application Security Project (OWASP) http://www.owasp.org/

Data Layer Frameworks and Governance of Data Services

Problem: data handling takes essential space in source code. This code is not only time consuming during development. It often requires major troubleshooting efforts and might cause problems in testing and production.

Here is the example of the message we can often see in the JBoss log files:

```
09:38:06,304 INFO  [CachedConnectionManager] Closing a connection for you.
Please close them yourself:
org.jboss.resource.adapter.jdbc.WrappedConnection@1cf630a
java.lang.Exception: STACKTRACE
```

Similar log lines we can find in the log files in the D&A and other applications.
Writing data handling code consumes significant time with far from perfect results.

Solution:
There are several frameworks, including Hybernate, Spring, and DataService, that recognize and solve common data processing issues and allow us to minimize our coding efforts and focus on specific business cases.

The Hybernate and Spring frameworks provide object-relational mappings and offer creating SQL statements on-the-fly. Developers don't need to be familiar with the database structures; they just create initial configuration files instead.

The DataService framework is the smallest one. The framework requires developers to know data structure and use this knowledge to create good SQL statements. The framework allows developers keep SQL statements in separate files that can be directly used for testing SQL with the TOAD or SQLPlus tools. The DataService takes care of data source initializations, data connections, and processing result sets.

The DataService class is the façade of the com.util.jar library. This is a single jar file with a very small footprint.

The library includes the `ItsBaseServlet` class that reads initialization parameters for a web application from the **web.xml** file, and makes data sources available. The initialization procedure is called automatically when application is started.

There are several pieces of data that we pass to the framework via the **web.xml** file

1. The application will need to work with the file system. Let the application know where it is deployed to make this work a bit easier. Note two values below separated by comma: one for Windows deployment and another for Linux deployment, for example in the AWS environment.
   DataService will recognize Operating System and will use a proper pathToDeploy value.

```xml
<context-param>

 <param-name>pathToDeploy</param-name>

 <param-value>c:/aws/itofthefuture/,/home/ec2-user/itofthefuture/</param-value>

</context-param>
```

2. Provide the description of the data sources in the file located in the deploy directory named as applicationName-ds.xml, for example MyApp-ds.xml.
   Map data sources to the application in the web.xml file using **context-param** elements. Use the names for data source that end with "DataSource", for example, "MyAppDataSource".

```xml
 <context-param>
     <param-name>MyAppDataSource</param-name>
     <param-value>MyAppDataSource</param-value>
  </context-param>
```

```
   In the case when the number of databases and tables is not very
significant, directly describe in the web.xml all metadata.
For example, if you have just one database, list the tables as comma
separated parameters.
```

```xml
   <context-param>

    <param-name>tables</param-name>

    <param-value>UserProfile,Accounts,Roles,Teams,Articles,Comments</param-value>

   </context-param>
```

```
For each table above provide the column names (table keys).
```

```xml
<context-param>

 <param-name>UserProfileTableKeys</param-name>

 <param-value>userID,firstName,lastName,mobileNumber,lastUpdate,domain</param-
value>

</context-param>
```

```
DataService framework will create necessary tables if do not exist and
provide default data types, such as varchar(60).
```

It is easy to specify data types as in the example below with the Roles table.

```xml
<context-param>
 <param-name>RolesTableKeys</param-name>
 <param-value>roleID,roleName,lastUpdate</param-value>
</context-param>
<context-param>
 <param-name>RolesDataTypes</param-name>
 <param-value>number,varchar(60),varchar(120)</param-value>
</context-param>
```

3. Use the **listener-class** element to point to the `ItsBaseServlet` class in the web.xml file.
```xml
<listener>
 <listener-class>com.its.util.ItsBaseServlet</listener-class>
</listener>
```

4. Similar to Struts or Spring frameworks there is a single entry in the **web.xml** for the servlet class and servlet mapping.

```xml
<servlet>
  <servlet-name>com.its.util.ItsBaseServlet</servlet-name>
  <servlet-class> com.its.util.ItsBaseServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>com.its.util.ItsBaseServlet</servlet-name>
  <url-pattern>/Lookup</url-pattern>
</servlet-mapping>
```

It is expected that the request will always include url-pattern **/Lookup** and the following parameters will point to a real action or a page.

Briefly about the `ItsBaseServlet` class. This class extends HttpServlet and implements the following interfaces:
- HttpSessionListener, // allows to listen to session events, such as createSession and destroySession and insert in these methods a specific behavior
- ServletContextListener // allows to listen to the servlet context events, such as initialization. Implement the initialize() method to provide specific to your application initialization procedures not related to database sources.

The ItsBaseServlet class includes the doPost() and doGet() methods that respond to a request from a browser. Similar to the Dispatcher class in the Spring this class will delegate control to an action class or to a Java Server Page.

This can be done with the simple URL, such as:
/BASE/Lookup?action=content
Or in a more formal REST presentation
/BASE/Lookup/content

How to use the Data Service framework :

a) Copy the **com.util.jar** library to the WEB-INF/lib directory and make it a part of the Java Build Path.
b) Handle data by using the DataService API (see References at http://itofthefuture.com/BASE/Lookup?action=content&issue=175)

You can use Statement or PreparedStatement in the DataService framework
Prefered way is using PreparedStatement, which besides performance benefits also provides more security and has embedded work around symbols, such as a single quote and more, which may appear in SQL statements.

Example with the Statement

getUser.sql – file in the Eclipse project in WebContent/config/sql – directory

Select * from users where loginName = ':loginName'

Provide this code in your action class to execute this statement in the MyAppDataSource.

// See recommendations on the action class and find more details on the HashMap keys

```
keys.put("loginName", form.getLoginName()); // common HashMap keys
List records = DataService.getData("getUser", keys, User.class, "MyAppDataSource");
If(records.size() == 0) {
   // no users found
   return "error";  // forward to the Error page
}
User user = (User) records.get(0);
request.getSession().setAttribute("user", user);
return "success"; // forward to the next page
```

For PreparedStatement:

insertUser.sql – file in the Eclipse project in WebContent/config/sql – directory

INSERT into User values(? , ?)

/* parameters:
- name of the sqlFile - insertUser refers to insertUser.sql file

- String array of run-time parameters to replace "?"
- the name of data source - in this case "myDB"
*/
DataService.setPrepDataBySqlName("insertUser", new String[] {login, psw}, "myDB");

Data Handling: DataService with a pool connections

Problem: There are cases when it is beneficial to keep a single connection for multiple operations.

Solution: The DataService framework includes multiple methods that allow developers to obtain a connection from the pool and reuse the same connection.

Here are several lines of code that demonstrate this feature.

```
Connection con = DataService.getDataSourceConnection(dsName);

Date[] dates = SQLHelper.getDateArray(con,
  sql, // String of SQL statement
  dsName, //String of data source name
  false); // boolean closeConnectionFlag=false (do not close connection)

int[] indexes = SQLHelper.getIntArray(
  con, sql2, dsName, false); // do not close connection

long[] indexes = SQLHelper.getLongArray(
  con, sql2, dsName, false); // do not close connection

long index = SQLHelper.getLong(con, sql, dsName, false);

int nAffectedRows = SQLHelper.setData(sql, con, dsName, false);


Note, that the connection has never been closed at this point.

Don't forget to close this connection with the last operation.
For example:

int nAffectedRows = SQLHelper.setData(sql, con, dsName, true);

or

int nAffectedRows = SQLHelper.setData(sql, con, dsName); // default closing
```

Custom actions in the Struts framework

Problem: Struts provides developers with highly reusable solution based on Action and Form concepts. The form bean reflects the screen structure and keeps data in place similar to the PostBack concept in .NET. There are cases when screen is created on-the-fly and screen structure is not permanent, so it cannot be provided in the Struts form bean. There are also cases when programmer needs access to the attributes on the page and same time to the session attributes.

Solution: The DataService framework includes the DataAction class that extends the Struts Action and performs generic action operations that include collecting page attributes into the HashMap keys. Then, the execute() method calls the perform() method that is implemented in a custom subclass that extends the DataAction.

Here is the example of a custom action below.
Note, that the HashMap keys is already filled with the data (loginName). The keys were collected by the DataAction class that provides generic functionality.
The UserAction only performs a specific function, in this case, getting user data.

```
/**
 * The UserAction class implements the perform() method
 */
public class UserAction extends DataAction {
    public String perform(HttpServletRequest request) throws Exception {
        List records = DataService.getData("getUser", keys, User.class,
          "MyAppDataSource");
        if(records.size() == 0) {
          // no users found
          return "error";  // forward to the Error page
        }
        User user = (User) records.get(0);
        request.getSession().setAttribute("user", user);
        return "success"; // forward to the next page
    }
}
```

Content Management in Web Applications

Content Management (CM) with multiple content types was described [1] and implemented in many CM systems and then standardized with JSR 170 that provides a set of interfaces for CM implementation.
Here is the basic concept and implementation example provided in the DataService framework.

Problem: With multiple content types, (like messages, pictures, documents, products, etc.) the systems that describe content management for each and all different types might grow unmanageably big.

Solution: Define all different content types in the same manner while focusing on attributes that allow the system to consistently change, review, approve, and display any content type for a selected audience. Then, a complicated task of content management described once will be applicable to all content types.

For example, messages, pictures, documents, products, and etc. can have different attributes and behavior in some specific applications, but in the content management applications they can be unified with simple object model and definitions similar to the example below.

All content types can be located in the **Content_Elements and Content** tables.

All attributes of any specific type are described in the **Content_Elements** table with generic parent-child relationships. The **Content_Elements** table serves as the table of indexes and can easily scale to multiple million records.
The type of content and the order fields allow the system (if necessary) to present and assemble the content into visible pages.

With growing number of records the data from the **Content** table can be distributed to multiple type-tables, where each type-table would have the same columns as the **Content** table and the table name that reflects the content type.

The content is usually related to a specific business domain. Example of domain can be "Services", "Products", "For Approval", "For Review", and etc.

Specific business domains are used by specific audiences or groups. A group can be a department of development group, or a user group, and etc.

The **Content_Elements** includes the following columns:

Content_Element_ID	Content_Element_PID	Content_Element _ORDER
Number	Number	Number

Content_Element_ID – unique identifier of the content element – Primary Key
Content_Element_PID – Parent ID of the content element
Content_Element _ORDER – Order in which elements placed inside a parent element

During the visualization process, the system retrieves all children of the parent and places them in the proper order on the page.

The **Content** table includes the following columns:

Content_Element_ID	Content_Text	Domain_ID	Content_Type_ID

Number	Text	Number	Number

Content_Element_ID – The unique identifier for each element – Primary Key
Content_Text – Text or URL
Domain_ID – Data domain allows the system to group data for privileged access, versioning, and group operations on data
Content_Type_ID – The Content type is used by the system to properly visualize different content types.
With growing number of records the data from the **Content** table can be distributed to multiple type-tables, where each type-table would have just three first columns from the **Content** table and the table name will reflect the content type.

For example:
Message

Content_Element_ID	Content_Text	Domain_ID

Product

Content_Element_ID	Content_Text	Domain_ID

Text

Content_Element_ID	Content_Text	Domain_ID

Etc.

This change (from a single to multiple content tables) can be done at any time without changing any code except one SQL query that assembles content elements (children) that belong to the same parent.

SQL for a single table
select * from **content** where id in :listOfIDs and content_type = ':content_type'

SQL for multiple tables
select * from **':content_type'** where id in :listOfIDs

Make sure that DataService is used and SQL is kept outside of Java code to make this change painless.
In both cases the listOfIDs is selected and if necessary split in pages by the DataService methods.

The benefit of this unification is drastic code reduction.
For example, the most common task of visualizing the content would require assembling all different elements for a particular page. Implementation of this task is usually takes many thousands lines of code with multiple DB queries.

Here is the example of code that will do the same with several code lines:

```java
/**
 * The testDisplayContent() method demonstrates
 * visualization of a content element with ID = 1
 * The method will retrieve all IDs first (using the CONTENT_ELEMENTS table)
 * and then retrieve the CONTENT table records that belong to these IDs
 *
 * The method will assemble all content elements that belong to this parent
ID
 * Each line in the list will be provided with custom Edit/Delete controls
 * The method will automatically provide the controls with specific keys
 * that point to selected items
 * At the end of the list a set of custom Submit buttons will be provided
 * @return html
 * @throws Exception
 */
public static String testDisplayContent() throws Exception {
       // Provide DS name mapped in the web.xml
       String dataSourceName = "myDataSource";
       // The content element ID = 1 is selected in the example
       int idOfTheSelectedItem = 1;
       String content_type = "*"; // all content types to be displayed
       // Instructions to create edit/delete controls for each row
       String controls = "<a href=/Edit.do?key=[0]>EDIT</a>;" +
       "<a href=/Delete.do?key=[0]>DELETE</a>";
       // Instructions to create a set of Submit buttons
       String submit = "/Process.do;Submit;Cancel";
       String html = getContentHTML(idOfTheSelectedItem, content_type,
                   dataSourceName, controls, submit);
       return html;
}
```

Workflow and web action reuse

Problem: Workflow (in the Struts world) is often defined by the forward line at the end of an action, for example:

```java
 public ActionForward upload(…) {
….
       // struts-config.xml will bind "success" of this action to a specific jsp page
       return mapping.findForward("success");
}
```

In this example the "success" of this action will be linked by the struts-config.xml to a specific JSP page. It is hard to reuse the "upload" action because this action is always linked to the same JSP.

Solution: Pass the "next" attribute or hidden field parameter to the request object to point to a specific JSP before the call to your reusable action.

Here are the code samples:

```
public ActionForward beforeUpload(…) {
….
        request.setAttribute("next", "esb");
        return mapping.findForward("upload");
}
public ActionForward upload(…) {
….
        String nextPage = (String) request.getAttribute("next");
        if(nextPage == null) {
                nextPage = request.getParameter("next");
        }
        if(nextPage == null) {
                nextPage = "next"; // default value
        }
        return mapping.findForward(nextPage);
}
```

The struts-config.xml should support all values of the next page.

Best Practices in Web Application Configuration

Problem: Developers often use web.xml and property files to configure application environment. Sometimes these files include URLs or names specific to test or production environment. If so, developers need to change source of these files and rebuild the deployment WAR or EAR files each time when they move from development to test and production. For example, EMARSConfig.properties file is a part of the EAR, and must be changed for each specific environment

Solution: create EAR and WAR files transparent to Development, Test, and Production, and avoid changing lines inside EAR/WAR files.

Provide all environment variables that are sensitive to Development/Test/Production in the separate files distributed to **deploy** and **conf** directories.

Provide data source variables in the file named as applicationName-ds.xml located in the deploy directory.
Provide URL pointers and other environment dependencies that are different for Development, Test, and Production, in the file named as applicationName.properties located in the **conf** directory.

Use the **web.xml** and similar files inside EAR or WAR space to set the variables that are the same for Development, Test, and Production.

Remember, that the DataService will bind all Data Sources named in the **web.xml** to their environment variables provided in the **deploy/applicationName-ds.xml.** The initialization is provided automatically when application started. (See Data Handling)
No extra code or efforts is needed.

Project
- - **Deploy**
- - - store WAR, EAR, and other files that are the same for Dev/Test/Production
- - -- **Development**
- - ----- store files specific for development
- - -- **Test**
- - ---- store files specific for test environment
- - -- **Production**
- - ---- store files specific for test production
- - -- **Docs**
- - ---- store installation instructions, DB scripts, and related documents
- - ---- store requirements and design documents

CVS is the perfect place to keep track of deployment files and important documents with their versions

---

Stateful Beans and Memory Leaks

Problem

Developing web applications, we often need to accumulate data from multiple user requests in a single object that implements the Value Object Assembler Design Pattern.
Working with EJBs, some developers use Stateful Session Beans for this purpose.
However, Stateful EJBs are the often source for memory leaks.

Solution

Avoid using stateful beans, use stateless everywhere it is possible. There are rare occasions when you really need stateful beans and in such cases you'll need to provide special code to destroy them.

In the web applications the role of stateful beans can play the Session object for each user or the Application object for all users.

What about the Application object?
It does exist in the .NET world, but in the J2EE web applications it is very easy to create one.

Why do we need such an object?
Here is the example:
One application (no names :-) demonstrates the code where each user session starts with reading 6 tables that include common for all users information that we can name ApplicationInfo.

For each login the code hits again these 6 tables because each user needs to know these data.

This can be done differently using the Singleton design pattern.

The MyServletContextListener class (that is called once upon application init) initializes this object.
The object is static and can be viewed by all users.
It has private constructor and static getInstance() method that returns a single instance of the object to any user.

Handling huge data volumes – auto-paging with DataService

Problem: Some queries can return unpredictably big Result Set that can crash memory management systems. This is especially true when we deal with big multiple-field records.

Solution: In most cases we need to display this data for a user and the user will only observe a portion of data at once.
One approach is to add one pre-processing select upfront to select only reference indexes instead of full records, and then use a subset of this array of indexes to select a desired page of data records at once. Here is the example of implementation using accelerated paging methods of DataService frameworks. The methods manage array of unique indexes to select a page of data records at once. User will be able to navigate from page to page and re-define the page size.

```
String getIndexes = "select uniqueID from aTable where conditions";
int[] arrayOfIndexes = DataService.getIntArray(getIndexes, dataSourceName);

String query = "select field1, field2, etc from aTable where uniqueID in (:arrayOfIndexes)";
int currentPage = 0;
int recordsPerPage = 50;
String pageQuery = DataService.getPageQuery(query, arrayOfIndexes, currentPage, recordsPerPage);
// create a query to retrieve a page of records

List records = DataService.getData(pageQuery, dataSourceName, RecordBean.class);
```

It is possible to do even more pre-processing work if we know specifics related to data structure and query requests. We can group data on arrival or/and periodically based on some criteria, for example, date, or business domain. Then a user ad-hock query will be performed much faster because it will work against smaller data volume. Some filtering work is done upfront.

Governance of Data Services: Providing Data Performance Status

Problem: Significant time is spent by developers to identify problems. In most cases developers report such problems in the log files. Retrieving truth from these files is quite difficult because of non-existing standards to writing and reading log files.

Solution: The DataService framework uses the java.util.logging.Logger and includes the LogWriter and Stats classes, which report run-time events into a log file. The Stats class also helps to sort out and visualize reports.

Here are some of the methods of the Stats class:

- setAppName – register application at its init time
- setDataSourceDetails – register data source
- setAppDetails(appName, eventName, eventDetails) – see more in API

The DataService framework interacts to the application server (Jboss or WebLogic) working with the connection pools provided by these containers. This interaction enables application self-diagnostics features, helps to collect and use important information on data status and query performance, and etc.


Benefits: Data Status

- NumberOfChanges:
- NumberOfQueries:
- FirstConnectionTime:
- LastConnectionTime:
- NumberOfErrors:
- Errors:
- MaxQueryTime:
- LongestQuery:

Web Container and Application Server — Works with

ServletContextListener interface

ITSServletContextListener
contextInitialized() – init data
initialize() – to be overriden by a subclass

MyServletContextListener
initialize() – custom initialization

Reports run-time status of data sources

DataService — Uses
dataSource DataSource
dataSources Hashtable
getData(sqlName, map, bean.class) : List
setData(sqlName, map)
setDataSource(DS)

Mosel/Data Status   Local/Data Status

Legend: ___ **Java library**
___ Custom Library in the **com.util.jar**
___ **Java classes we need to code**
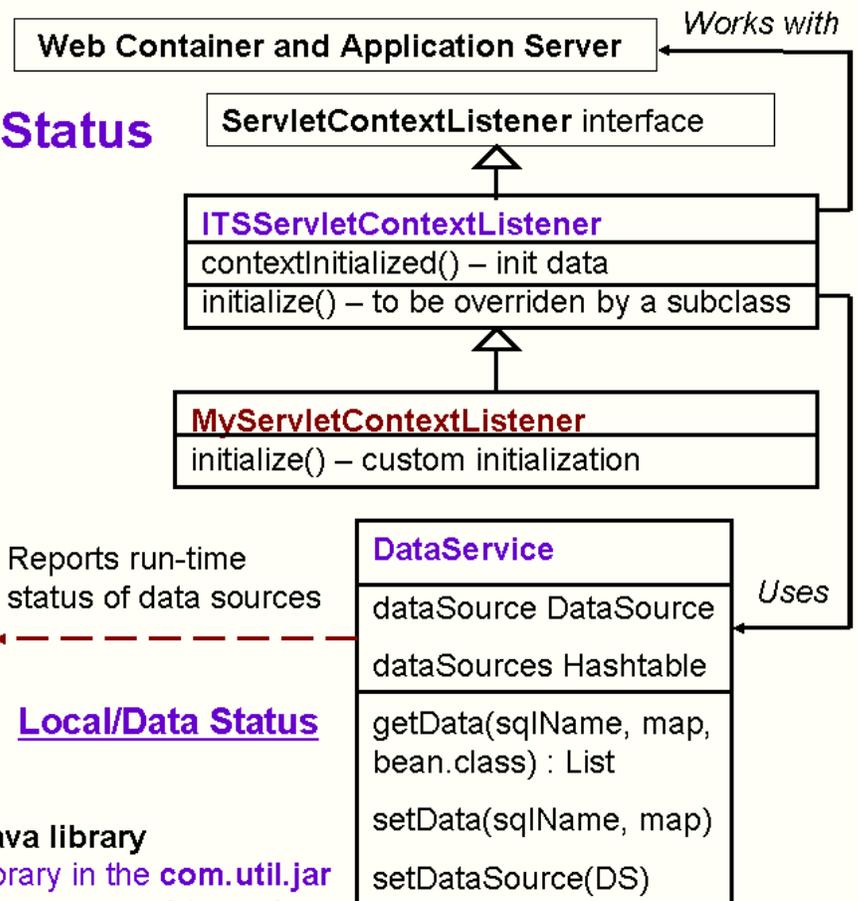
Here is the screen shot of the data status:

Status of data sources @ 10.152.12.208(10.152.12.208)

ServerDataSource

NumberOfUpdates: 2
LastConnectionTime: Wed Oct 12 16:10:47 MDT 2005
MaxQueryTime: 0
NumberOfErrors: 0
NumberOfQueries: 6
FirstConnectionTime: Tue Oct 11 19:11:33 MDT 2005
Errors:
LongestQuery: select * from NEPS_FAILED_LOGINS where upper(userid) = upper('jzhuk')

LegalDemandDataSource

NumberOfUpdates: 0
LastConnectionTime: Wed Oct 12 16:11:11 MDT 2005
MaxQueryTime: 0
NumberOfErrors: 0
NumberOfQueries: 1
FirstConnectionTime: Wed Oct 12 16:11:10 MDT 2005
Errors:
LongestQuery: select RECORD_ID, orig_record_id call_type , STRUCTURE_CODE, to_char(CALL_DATE,'DD-MON-YYYY') call_date, …. etc
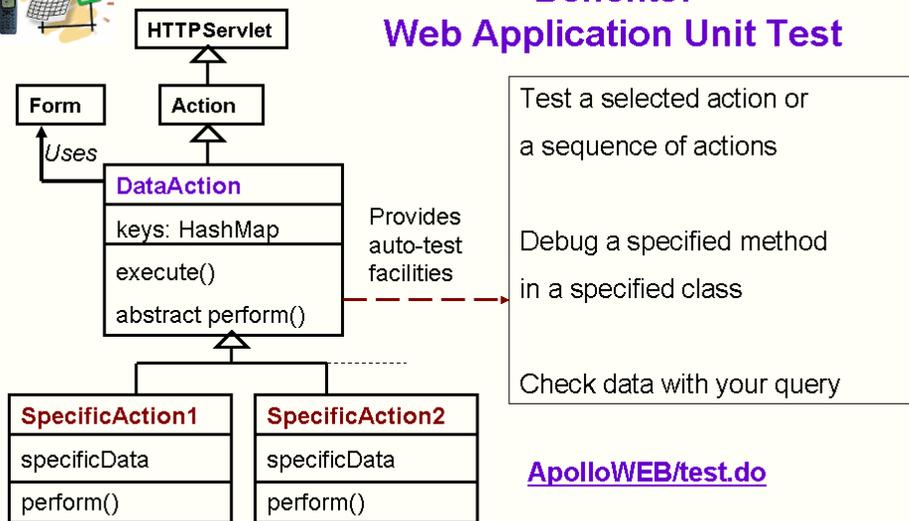
Automated unit testing for web application

Problem: Eclipse includes Junit to test selected classes creating on-the-fly the main() method for test cases. Web applications require complex communications with the web container that is not covered by Eclipse. The challenging part in the testing web applications is simulation of user interactions with the application screens.

Solution: Struts frameworks encourage developers to create form beans that reflect and store data associated with the user screens. The DataService framework provides direct access to these form classes and allows us to create form instances that simulate data input by users. The DataAction class (included in the DataService frameworks) extends the Struts Action and performs generic action operations that include collecting page attributes into the HashMap keys. This infrastructure provides facilities for automated testing of all actions that extend the DataAction. Initial data for each action are defined as default values in the form classes related to these actions.

HTTPServlet

Form

Action

Uses

DataAction

keys: HashMap

execute()

abstract perform()

Provides
auto-test
facilities

SpecificAction1

specificData

perform()

SpecificAction2

specificData

perform()

Legend: ___ **Java library**
___ Custom Library in the **com.util.jar**
___ **Java classes we need to code**

Test a selected action or

a sequence of actions

Debug a specified method

in a specified class

Check data with your query

ApolloWEB/test.do

Local/test.do

Local/Test All Actions

Here is the screen shot of the automated unit test:

Automated Test Report: MyAppWEB @ 10.152.12.208(10.152.12.208)
2005-10-12 16:08:20,780 ERROR [STDERR] Oct 12, 2005 4:08:20 PM
com.its.actions.DataAction testAllActionsINFO:

Start Action Testing: path=/ChangePasswordUpdate
action=com.comcast.MyApp.presentation.actions.ChangePasswordUpdateAction
form=ChangePasswordForm2005-10-12 16:08:20,780 ERROR [STDERR] Oct 12, 2005 4:08:20
PM com.its.actions.DataAction testAllActions
WARNING: e=java.lang.NullPointerException2005-10-12 16:08:20,781 ERROR [STDERR]
Oct 12, 2005 4:08:20 PM com.its.actions.DataAction testAllActionsINFO: End of testing the
action: /ChangePasswordUpdate2005-10-12 16:08:20,784 ERROR [STDERR] Oct 12, 2005
4:08:20 PM com.its.actions.DataAction testAllActionsINFO:

Start Action Testing: path=/UserManagement
action=com.comcast.MyApp.presentation.actions.UserManagementAction
form=UserManagementForm2005-10-12 16:08:20,784 INFO
[com.comcast.MyApp.presentation.actions.UserManagementAction] In execute method of
UserManagementAction2005-10-12 16:08:20,785 INFO
[com.comcast.MyApp.presentation.actions.UserManagementAction] In execute method of
UserManagementAction letter = null2005-10-12 16:08:20,785 ERROR [STDERR] Oct 12, 2005
4:08:20 PM com.its.actions.DataAction testAllActionsINFO:
action result=success
2005-10-12 16:08:20,786 ERROR [STDERR] Oct 12, 2005 4:08:20 PM
com.its.actions.DataAction testAllActionsINFO: End of testing the action

Automated System Testing for Applications

Problem

While unit test is commonly used in development community the system test is often only available to a QA team.
Growing complexity of enterprise applications increases the necessity to manage bigger parts of applications. Developers often concatenate several unit tests by creating a special test method just for this purpose.

Solution

Describe test scenario in the scenario.xml file using action names as test units. Armed with action unit tests the same framework will connect units into more complicated test cases according to a scenario.

Automated System Test Scenario

```xml
<Scenario name="Test User Management Functions" >
  <Action name="/LoginAction">
      <!-- Replace default form data -->
    <initData name1="value1" />
      <!-- Set attributes for Request and Session objects -->
    <requestAttributes name1="value1" />
    <sessionAttributes name1="value1" />
      <!-- Set expectations -->
    <expectedResults location=... name=.... Value=.... />
  </Action>

  <Action name="/CreateUserAction">
  .............
  </Action>
</Scenario>
```

JVM working with multiple processors and Clusters

Problem

Can a single JVM take advantage of multiple CPUs?
When it would be beneficial to cluster applications?

Discussion

A single JVM distributes its threads over multiple processors.
Each JVM will automatically take advantage of multiple processors on the machine.
For example on a 16-CPU machine each JVM can use all processors.

> As the number of processors is increased to resolve the issue of scalability, more and more threads are being simultaneously processed within the system.

All processors share the same memory and shared memory often becomes a bottleneck for J2EE applications. Multiple CPUs running a single JVM sometimes need to synchronize threads fighting for the same object. In this regard running different apps on multiple processor machines is more efficient. Clustering application or adding JVMs to the same machine to run the same process will only increase usage of system resources as it requires more overhead for each JVM plus intercommunications between them.
Clusters on the same machine might make sense if we have enough horsepower on the machine and would like to provide fail over solution for the application.

Scaling out or creating clusters via more servers to run the application is a valid solution if we just want to scale the application without changing the logics or/and removing possible overheads.

Graceful Exit and Recovery for Stand-Alone and Web Apps

Problem

Web applications are usually hosted by an application server like WebLogic, JBoss, or WebSphere. They live happily together in a single JVM process. But when the process is unexpectedly killed, each application might have its own last wish, for example saving the current application state. Application can gracefully die if the last wish is granted.

Solution

1. How to grant the last wish in every application?
Intercept JDK exit event. This is possible since JDK1.4

```
Runtime.getRuntime().addShutdownHook(new Thread() {
    public void run() {
        // grant the last wish
        execOnExit();
    }
});
```

2. Where to place this code?
I recommend placing this code in the class and method that is performed once when the application is initiated.

1.1 **For Web Applications,** it is recommended to place the last execution method in the servlet context listener and implement the **execOnExit**() method in the MyServletContextListener that extends this class.
1.2 **In stand-alone applications,** it is recommended to place the code (that intercepts JVM exit) in the main() method.

Graceful Recovery

Problem

Recovering applications might need to restore its state saved during the last exit, especially if the exit was a result of system shutdown. Regular application initialization procedures don't include this option.

For web applications the concept of Graceful Recovery might also require to restore a user session state and allow each user continue from the point where s/he left or was disconnected from the web application.

Solution

Graceful recovery is tightly related to Graceful Exit implementation.
We will consider generic and specific parts of the recovery as well as application and session recovery levels.

Application recovery

1. Graceful Exit facilities should be used to save the application state as a serialized object in a specific directory, for example named "**state**". The directory will be created during the first saving. The object will be saved with its application name.
For example, web applications, managed by an application server, like JBoss, will store their application state in the **server/default/state** – directory.

There are two parts of the saving: generic and specific.
Application saves specific data in the Hashtable named "state". For example, the code below uses common application facilities provided by the DataService to retrieve the table:

Hastable state = (Hashtable) DataService.getAppDetails().get(appName).get("state");

Then specific data can be placed into the table by a specific code provided in the application. Shutdown event calls for a generic mechanism that will serialize the state and save it as an object for later recovery. Generic mechanism will also call for some specific actions, like

saving in a database, implemented for example in the **MyServletContextListener** class by the **execOnExit**() method. This method is automatically called at the end of execution of the generic part of the code.

2. The recovery mechanism will be included in the initialization part of the application. For example, for web applications generic part of the recovery is implemented by the basic servlet context listener class that reads the serialized object and restores this object back into the Hashtable "state".

A specific part of the recovery mechanism will retrieve this table and work with the stored objects according to application needs.

This part of the code can be implemented, for example, in the **MyServletContextListener** class by the **initialize**() method. This method is automatically called at the end of execution of the generic part of the code.

Session recovery

Session recovery is not necessary related to application shutdown events. User's session can be interrupted due to multiple reasons including session expiration, disconnection from the Internet, and etc. Saving and recovering user sessions are expensive procedures that should be well justified by business goals. For example, it might be important to allow users continue working with the shopping cart, and etc.

1. Storing user's sessions.

For example, web applications, managed by an application server, like JBoss, can store serialized user's sessions in the **server/default/state** – directory, with the name that includes application name as well as user's name, for example, in the **myApp.jzhuk.ser** – file.

There are two parts of the saving: generic and specific.
Application saves user's session in the **MyServletContextListener** class by the **sessionDestroyed**() method. Here is the code to save a user's session:

String username = (String) session.getAttribute("username");
saveSession(username);

The **saveSession()** method is a generic part of the code that serializes the object and stores the object with the name, like "**appName.username.ser**".

2. Session Recovery

The recovery mechanism will be included in the initialization part of the session. For example, a specific code in the successful login action can use generic part of the code, for example, implemented in the **retrieveSession(username)** method of the servlet context

listener. The **retrieveSession(username)** method reads and un-marshal (de-serializes) the session. Then it passes the session to a specific code that will continue working with the session objects according to application needs.

Date Conversion with the SimpleDateFormat

Problem: The java.text.SimpleDateFormat class is often used to convert dates to displayable form. A common code snippet is presented below:

```
Date userDate = new Date();
SimpleDateFormat sdf = new SimpleDateFormat("dd-MMM-yyyy");
String date = sdf.format(userDate); // crash?
```

The last line of this code might unexpectedly cause the Parse Exception if a foreign character set is installed on the machine running the software.

Solution: There are the following reasons for the exception:
The second line of the code constructs the instance of the SimpleDateFormat with the default LOCALE (that might be not US English). This instance can be easily confused by mismatch with another character set.

The solution is to construct the instance forcing the US English Locale.

```
SimpleDateFormat sdf = new SimpleDateFormat("dd-MMM-yyyy",
// enforce US char set
java.util.Locale.US); // !!!
String date = sdf.format(userDate); // no crash!
```

Date Calculations – do this right!

Problem: There are cases when dates are calculated with oversimplified code like this:

```
SimpleDateFormat sdf = new SimpleDateFormat( "MMMMM d, yyyy", java.util.Locale.US);
Date startingDate = sdf.parse("October 30, 2005");
Date today = new Date();
for(java.util.Date currentDate = startingDate; currentDate.before(today); )  {
   // code to perform every day

   // calculate next day
   long time = currentDate.getTime() + 24*60*60*1000; // + one day
   currentDate = new Date(time);
}
```

This code looks simple and innocent but it will produce wrong results at least twice a year when daylight savings make days 23 or 25 hours.

Solution: More precise code uses java.util.GregorianCalendar class that knows about daylight savings for each time zone.

```
SimpleDateFormat sdf = new SimpleDateFormat( "MMMMM d, yyyy", java.util.Locale.US);
Date startingDate = sdf.parse("October 30, 2005");
Date today = new Date();
TimeZone tz = TimeZone.getTimeZone("America/Denver");
GregorianCalendar  calendar = new GregorianCalendar(tz);
for(java.util.Date currentDate = startingDate; currentDate.before(today); )  {
   // code to perform every day

   // calculate next day
   calendar.setTime(currentDate);
   calendar.add(Calendar.DATE, 1);
   currentDate = calendar.getTime();
}
```

Exception handling

Exception handling and performance

Problem: Developers often simplify source and cover by try/catch statement a big block of code. Try/catch statement is expensive. Every line inside the try/catch is executed in a try/catch mode that takes significantly more time and resources that usual line of code.

Solution: Provide try/catch statement only around code that requires it. Provide multiple try/catch blocks if necessary. Illustration shows bad source on the left and improved source on the right. The source on the left will be performed slower because every line in the loop is performed in the try/catch mode.

Bad source

```
public static main(String[] args)
{


 try
 {
   int[] numbers = new int[args.length];
   for(int i = 3; i <args.length; i++)
   {
     numbers[i] =
       Integer.parseInt(args[i]);
   }
} catch(Exception e) {
   System.err.println(e);
 }
```

Better source

```
public static main(String[] args)

{
  int[] numbers = new int[args.length];
  for(int i = 3; i < args.length; i++)
  {
    try
    {
      numbers[i] =
             Integer.parseInt(args[i]);
    } catch(Exception e) {

              System.err.println(e);
  }
```

```
}                                    }
                                     }
```

Exception Handling and debugging

Problem: Developers often limit error handling by the method **e.printStackTrace()** and any exception causes application crash with several pages of exception lines produced by this method. This log is hard to read. There are also cases when application could continue running in spite of the exception.

Solution: Declare method that can throw exception with the option "throws Exception". Collect information about the exception in the try/catch statement. It is usually the best place to understand what happens. Provide meaningful message in the log and integrate collected exception data into the **ExceptionDetails** instance. Throw the object of the ExceptionDetails and let the calling method to decide what to do next. The ExceptionDetails class is provided with the DataService framework

Here is the extract from the ExceptionDetails class and the example of its usage in the DataService framework.

```java
public class ExceptionDetails extends Exception {
        private int code = -1;
        private String className = "";
        private String methodName = "";
        private String debugHint = "";
        public static int DATA_SOURCE_EXCEPTION = -1;
        public static int CONNECTION_EXCEPTION = -2;
        public static int SQL_EXCEPTION = -3;
        public static int OTHER_EXCEPTION = -4;

        public ExceptionDetails(int code, String message, String className,
          String methodName, String debugHint){
                super(message);
                this.code = code;
                this.className = className;
                this.methodName = methodName;
                this.debugHint = debugHint;
        }
        // more code
    }
-------------------------------------- Example of usage -------------------------------
/**
 * The getInt() method finds a common or specific data source,
 * makes a connection, and executes a statement that selects a number (int)
 * @param sql
 * @param dataSourceName
 * @return number
 * @throws Exception
```

```
*/
    public static int getInt(String sql, String dataSourceName) throws Exception {
      DataSource ds = dataSource;
      if(dataSourceName != null) {
        ds = (DataSource) dataSources.get(dataSourceName);
        if(ds == null) {
          logger.severe("DataSource "+ dataSourceName + " not found!");
          ExceptionDetails ed = new ExceptionDetails(
            ExceptionDetails.DATA_SOURCE_EXCEPTION,
            "DataSource "+ dataSourceName + " not found!",
            "getInt", "DataService", "sql="+sql);
          throw ed;
        }
      }
      // … more code …
  }
```

Exception handling in web applications

Problem

Even in the professional web applications the error screens often look not very professional and they differ from application to application.

Solution

Handling exceptions in web applications is partially standardized by Struts frameworks.
Struts configuration file can include pointers to error handler actions.
Creating a standard Web Exception Handler can provide a standard look and feel to all error pages across web applications.
It is recommended to include the section below in the struts configuration file to point to a standard handler that generates a standard error screen.

```
<!-- ======== Global Exception Definitions -->
<global-exceptions>
        <exception type="java.lang.Exception" key="none"
        handler="com.its.actions.WebExceptionHandler" >
        </exception>
</global-exceptions>
```

In the Java class you include a log statement that indicates your application name (multiple applications are hosted by the same app servers), a meaningful business reason for the record or a business category (like "User Admin Error" or "Successful login"), and a specific message. Such records will be visualized into meaningful status web pages. In the example below, the com.its.util.Stats class is used to prepare statistics of errors and provide preventive analysis

```
} catch(Exception e) {
        Stats.addAppList(appName,
"User Admin Error", e.getMessage());
}
```

This code will include this error into the application status statistics and produce a log record below (you can also produce the same format with a regular log4j or java.util.loggin package):

"[[your-app-name] User Admin Error]specific-error-message"

See on the right side how such records look on the web page

⊞ WebDisp Errors:1

⊞ Check Status:3

⊟ Successful Login:3

Tue Dec 06 18:02:08 MST 2005 jzhuk

Tue Dec 06 18:02:54 MST 2005 jzhuk

Tue Dec 06 18:03:47 MST 2005 jzhuk

JMS Topics and Queues

Problem

In the multi-component system interactions, one of components might not be ready to reply to the request.
If interaction nature is synchronous, the requestor will wait for the response and crash or provide a nicely formed timeout exception. There are cases when application must tolerate "I am too busy now" behavior and demonstrate flexibility of getting service when it is ready.

Solution

Event-driven or asynchronous messaging is a new way to think of applications

JMS represents an asynchronous way of doing business. From the very beginning, I have to admit that messaging applications are not exactly new in the industry. Think about email service, a well known veteran, and the simplest example of asynchronous communications.

One can send email, and then do other things without waiting for the recipient's response. This is a really great invention that frees our hands and fills our hearts with the happiness of a "done deal". Except for the fact that you are never sure that your mail will be delivered.

Simple Mail Transport Protocol (SMTP) is not reliable by definition.

Messaging services like Message Queue (MQ) from IBM, Inc., or JMS from Sun Microsystems, Inc. guarantee delivery.

Today more and more businesses are linked to each other. Creating new applications and adapting old ones, we include the software that provides these links in the scope of the applications.

The software can be built with a synchronous connection-oriented design, or with an asynchronous paradigm that allows networks to dissolve and appear again. The synchronous design is known for better performance and for tightly coupled parts of applications. The asynchronous design provides better flexibility and scalability in a loosely coupled application environment.

Messaging services used to be expensive and tended to proprietary solutions. These factors prevented most businesses from considering messaging solutions as primary in their application design.

The situation has changed. JMS is a standard, inexpensive solution available for every company. It takes time to re-invent existing applications in new asynchronous terms, but it definitely makes sense to think about new projects as potential messaging solutions.

How does it work?

JMS includes two major mechanisms: point-to-point messaging and publish-subscribe domain.

A point-to-point mechanism allows multiple clients to send their messages into message queues. Message queues serve as mail boxes for receivers.

The most interesting and intriguing point-to-point mechanism is the publish/subscribe mechanism. Clients subscribe to topics and publishers post their messages to the topics.

You can think of topics as mail boxes, but keep in mind that each topic has many subscribers (serves as a box for multiple clients) and each message may have multiple recipients.

A client must subscribe to the topic and be an "active" subscriber in order to receive messages. A single client can be a publisher as well as a subscriber.
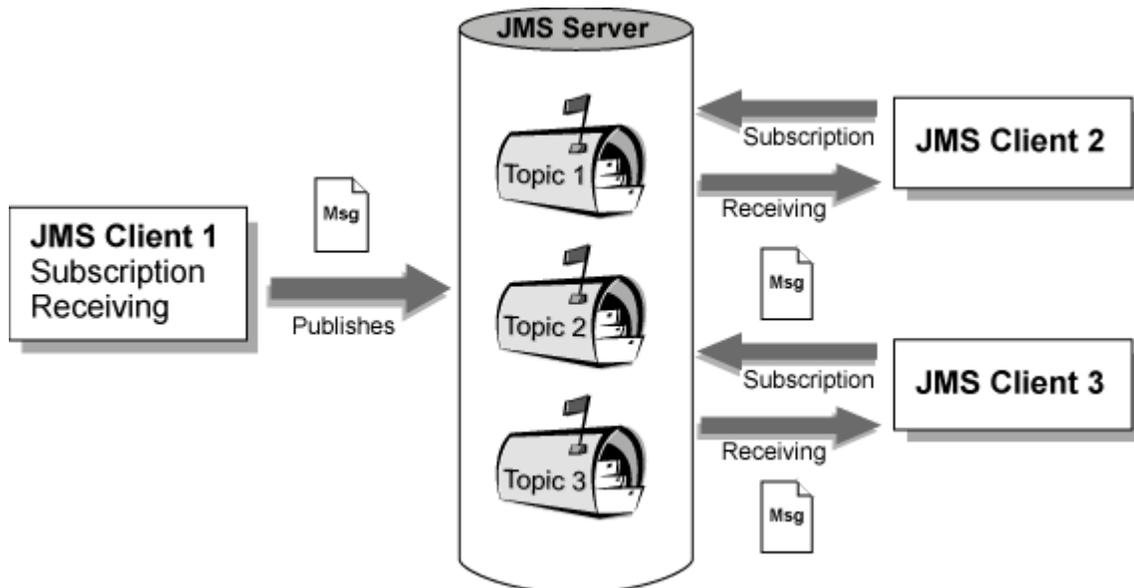
The picture above illustrates the high level architecture.

Topics are located at the JMS Provider or JMS Server. A JMS Provider can use one of the existing implementations of Sun's JMS API - for example, from the OpenJMS group [4].

A client uses a connection factory to create a proper connection to the JMS Server. There are two connection factories available with the JDK from Sun, but generally speaking a client program should start its work with a JNDI lookup for available connection factories.

We can read the JNDI context from the *jndi.properties* file (if exists).

Context ctx = new InitialContext( ); // reading jndi.properties file

The other alternative is to create the context on-the fly as in the example below.

Hashtable props = new Hashtable();
String modeType = RmiJndiInitialContextFactory.class.getName();
props.put(Context.PROVIDER_URL, "rmi://" + hostName + ":" +
  OPEN_JMS_PORT + "/" + JNDI_SERVER_NAME);
props.put(Context.INITIAL_CONTEXT_FACTORY, modeType);

Context ctx = new InitialContext(props);

The next step is to use JNDI lookup for the proper connection factory.

// in the case of using the Point-to-point domain with queues
QueueConnectionFactory queueConnectionFactory =
  (QueueConnectionFactory) ctx.lookup("QueueConnectionFactory");

// in the case of using the Publish/Subscribe domain with topics
TopicConnectionFactory topicConnectionFactory =
  (TopicConnectionFactory) ctx.lookup("TopicConnectionFactory");

The following code is used for the Publish/Subscribe mechanism.

// create a topic connection
TopicConnection topicConnection = topicConnectionFactory.createTopicConnection();
topicConnection.start();

// create a topic session (non-transactional) with auto-message-acknowledgement
TopicSession topicSession =
  topicConnection.createTopicSession(false, Session.AUTO_ACKNOWLEDGE);

// create a topic
Topic topic = topicSession.createTopic(topicName);

// make a client a publisher
TopicPublisher topicPublisher = topicSession.createPublisher(topic);

// We are ready now to publish a message
// The message can be any serializable Java object
// For example, a message can be a String or a Hashtable
ObjectMessage objMsg = topicSession.createObjectMessage(anyObject);

// finally publish the message to the topic
topicPublisher.publish(topic, objMsg);

// Here is the subscription part.

```
    // Note, that the same client can be a publisher and a subscriber
    TopicSubscriber topicSubscriber =
        topicSession.createSubscriber(topic, null, !ownReceiving);

    // set Message Listener in THIS class
    topicSubscriber.setMessageListener(this);
```

This class must be defined as MessageListener implementation
The MessageListener interface requires implementing the onMessage() method.

```
    /**
     * The onMessage() method does the message processing
     */
    public void onMessage(Message msg)  {
        // call a service to process the message
        // this example assumes that messages include "service" and "action" parts
        try {
            String serviceName = msg.getStringProperty("service");
            String action = msg.getStringProperty("action");
            ServiceConnector.performService(service, action, msg);
        } catch(Exception e) {
            // error handling procedure and msg
        }
    }
```

Please [follow this link](#) to see a great example of [using JMS Queue](#).
This example demonstrates the [Command Message](#), [Return Address](#), and [Correlation Identifier](#)
Design Patterns.

References:
1. "[Integration-Ready Architecture and Design](#)", Cambridge University Press, Jeff Zhuk, ISBN 0521525837,
2. [http://javaschool.com/about/publications.html](#), [http://www.amazon.com/exec/obidos/ASIN/0521525837](#)
3. [http://javaschool.com/school/public/web/web.apps.architectures.htm](#)
4. [http://javaschool.com/school/public/web/web.apps.internals.htm](#)
5. [http://javaschool.com/school/public/web/JZ.web.frameworks.htm](#)
6. [http://javaschool.com/school/public/JZ.Train.Reuse.htm](#)
7. [http://javaschool.com/school/public/JZ.Train.BEA.htm](#)
8. [http://javaschool.com/school/public/JZ.Hints.htm](#)
9. [http://javaschool.com/school/public/collaboration.htm](#)
10. [http://javaschool.com/school/public/DataService.html](#)
11. [http://www.enterpriseintegrationpatterns.com/RequestReplyJmsExample.html](#)