International Journal of Engineering Research and Development
e-ISSN: 2278-067X, **p**-ISSN : 2278-800X, www.ijerd.com
Volume 5, Issue 6 (December 2012), PP. 39-47

Using Factory Design Pattern for Database Connection and Daos (Data Access Objects) With Struts Framework

Mukesh D. Parsana[1], Jayesh N. Rathod[2], Jaladhi D. Joshi[3]
[1]M.E. (C.E., Pursuing), Atmiya Institute of Technology & Science (Gujarat Technological University), INDIA.
[2]HOD (Department of Computer Engineering), Atmiya Institute of Technology & Science, Gujarat, INDIA.
[3]M.S. (CE), Florida Atlantic University, Boca Raton, FL, USA.
[3] Research Assistant, Motorola Inc., Sunrise, FL, USA.

Abstract: - The Jakarta Struts is one of the most widely being used J2EE frameworks. It incorporates programming and design expertise from some important design patterns like MVC, Command, Front Controller, Adaptor and Template Method design patterns. The Struts framework provides utility classes to handle many of the most common tasks in Web application development i.e. classes for security, encryption, cryptography etc. This paper discusses how design and programming expertise of factory design pattern can be used with Struts to manage data objects and database connections efficiently. The Factory Pattern promotes loose coupling by eliminating the need to bind application-specific classes into the code.

Keywords: - MVC, Factory Method Pattern, Abstract Factory, Framework, Struts

I. INTRODUCTION TO ARCHITECTURE FRAMEWORKS

A framework is a set of prefabricated software building blocks that programmers can use, extend, or customize to suit their application. It is an object oriented reuse mechanism that allows the developer to decompose an application into a set of interacting objects. It describes the interfaces implemented by the framework components, the flow of control between these components, and the contracts between the components and the system. In this way the framework is a reusable design. The standard interfaces and interactions make it possible to mix and match existing components and create a wide variety of systems from a core set of components.

A framework can make it very easy to quickly build sophisticated web applications. Rather than simple Java Servlets that merely enable access to a database, a framework allows you to build entire systems, with secure, high-performance database access via an object-to -relational mapping (no need to embed SQL in your code), background job queuing and handling with dynamic invocation of custom server-side objects, and sophisticated event notifications and logging to facilitate management of your completed application. Developers can focus on creating the business logic and layout of the UI. All the tedious infrastructure work is already there to build upon, so that creating applications is easy and fast. New technologies can be easily incorporated into the framework's structure and work seamlessly with existing components and services. As Java technology evolves, any solution that incorporates the framework can evolve with it.

The result of using a framework for n-tier applications is that the quality of the application increases dramatically while decreasing total cost of ownership and speeding time-to-market. Frameworks capture the programming expertise and best practices necessary to solve a particular class of problems without to reinvent the wheel.

There are numerous types of frameworks - some of which work at presentation layer, some work at business layer while some work at all layers. Some of the most popular J2EE frameworks are: Tapestry (presentation framework), Struts (reference implementation of the MVC-II specification) Jetspeed and Liferay (portal frameworks integrated with Struts), Expresso (an architectural framework integrated with Struts), Spring MVC. [1]

II. STRUTS FRAMEWORK

The Jakarta Struts provides a unified framework for developing servlet and JSP based web applications that use the Model-View-Controller (MVC) design pattern. The Struts was designed with the intention of providing an open−source framework for creating Web applications that easily separate the presentation layer and allow it to be abstracted from the transaction/data layers. The Model represents the business or database code, the View represents the page design code, and the Controller represents the navigational code.

The Struts framework provides utility classes to handle many of the most common tasks in Web application development. The Struts framework also provides custom tag libraries for outputting bean properties, generating HTML forms, iterating over various types of data structures, and conditionally outputting HTML.

The Struts framework uses many design patterns. Design patterns capture the experience of expert software developers and present common recurring problems, their solutions, and the consequences of those solutions in methodical way. The main design pattern behind the struts framework is MVC which separates business logic from controller and view/display logic. Controller is based on command and Front Controller design pattern. Action classes use Adapter Design Pattern and process() method of the RequestProcessor uses the Template Method Design Pattern. Struts uses ValueObject pattern also as all the data are encapsulated in an object before passing to JSPs. [6][7][8]

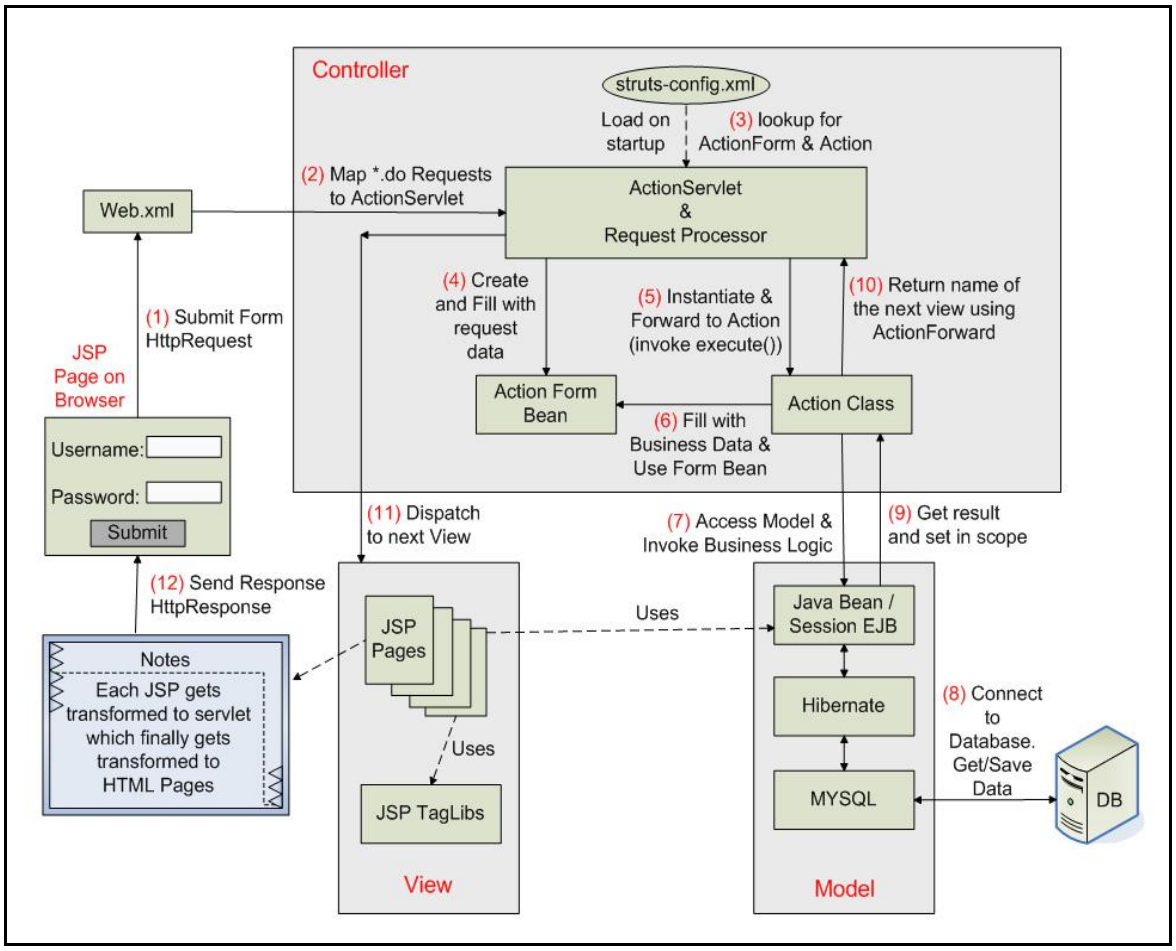

Fig. 1: Architecture of Struts based Web Application

Struts based web application works as per the steps given below.

(1) When user submits JSP form, form parameters will be sent to the web server along with the HttpRequest by post or get method.

(2) HttpRequest with *.do extension will be mapped to the appropriate Action Servlet / Request Processor according to configuration mentioned in web.xml file.

(3) Action Servlet/Request Processor lookups for appropriate Action and Action Form Bean classes from struts-config.xml file.

NOTE: Web server loads all configuration files like web.xml and struts-config.xml when it starts. Web.xml files contains configuration details to map *.do request to appropriate Action Servlet while the struts-config.xml configuration file is a link between the View and Model components in the Web Client. It plays an important role in building both Controller components and Application-specific configurations.[6]

(4) Action Servlet/Request Processor creates class for Form Bean and fills it with request data.

(5) Action Servlet/Request Processor instantiates Action class and calls its execute() method by passing ActionMapping, ActionForm, HttpServletRequest and HttpServletResponse as parameters.

(6) Action Servlet/Request Processor can use Form Bean class and fills it with business data.

(7) Action Servlet/Request Processor accesses Model component and invoke business logic. Model component can be Java Bean class or Enterprise Java Bean (EJB).
(8) EJB/Java Bean can communicate with database through Hibernate framework by using JDBC APIs.
(9) Result/Data retrieved from database will be set in the Request or Session scope variables.
(10) Action class returns name of the next view to Action Servlet/Request Processor according to the business logic.
(11) Action Servlet/Request Processor dispatches request to the next JSP.
(12) User can view next JSP page on browser window. JSP page will be transformed to servlet which in turn transformed to HTML page so browser can parse and display it to the user.

Table 1: Main Elements of struts-config.xml Configuration File [15]

Element Name	Description
<form-beans>	Contains form bean definitions. The Form beans create ActionForm instances at runtime. The details of each form bean are provided in the <form-bean> element.
<global-forwards>	Contains the global forward definitions. The forward name is the logical name used to map to a specific JSP. The <forward> element contains the logical name and the name of the corresponding resource which it maps to.
<action-mappings>	Contains the action definitions. Each action mapping is defined in an <action> element. The <forward> definition within <action>, maps the result of the action to the jsp page invoked.

III. FACTORY DESIGN PATTERN

One of the goals of object-oriented design is to delegate responsibility among different objects. This kind of partitioning is good since it encourages Encapsulation and Delegation. [3]

A class may need it's subclasses to specify the objects to be created or delegate responsibility to one of several helper subclasses so that knowledge can be localized to specific helper subclasses. Even Sometimes, an Application (or framework) at runtime, is not able to judge properly the class of an object that it must create. The Application (or framework) may know that it has to instantiate classes, but it may only know about abstract classes (or interfaces), which it cannot instantiate. Thus the Application class may only know when it has to instantiate a new Object of a class, not what kind of subclass to create.

Creational design pattern, more specifically the Factory Pattern tries to resolve out such issues. This pattern helps to model an interface for creating an object which at creation time can let its subclasses to decide which class to instantiate. We call this a Factory Pattern since it is responsible for "Manufacturing" an Object. It helps to instantiate the appropriate subclass by creating the right object from a group of related classes. The Factory Pattern promotes loose coupling by eliminating the need to bind application-specific classes into the code. It is an object oriented design pattern that returns an instance of one of the several possible classes based on the data passed to it. Generally all the classes that it returns have common methods and also have a common parent class. It should be noted that all the subclasses performs the different task and is optimized for different kind of data. [4][5]

A factory is not needed to make an object. A simple call to new will do it for you. However, the use of factories gives the programmer the opportunity to abstract the specific attributes of an object into specific subclasses which create them.

A. Factory Method Design Pattern

The factory method pattern defines an interface for creating an object, but let's subclasses decide which class to instantiate. Factory method lets a class defer instantiation to subclasses. As with every factory, the Factory Method Pattern gives a way to encapsulate the instantiations of concrete types. As shown in the figure below, the abstract creator gives an interface with a method for creating objects, also known as the "factory method". Any other methods implemented in the abstract creator are written to operate on products produced by the factory method. Only subclasses actually implemented the factory method and create products.

As mentioned above, the factory method lets subclasses decide which class to instantiate. Developers say "decides" not because the pattern allows subclasses themselves to decide at runtime, but because the creator class is written without knowledge of the actual products that will be created, which is decided purely by the choice of the subclass that is used.

The implementation is really simple as given below.

- The client needs a product, but instead of creating it directly using the new operator, it asks the factory object for a new product, providing the information about the type of object it needs.
- The factory instantiates a new concrete product and then returns to the client the newly created product (casted to abstract product class).
- The client uses the products as abstract products without being aware about their concrete implementation.

When you design an application just think if you really need it a factory to create objects. Maybe using it will bring unnecessary complexity in your application. Anyway if you have many object of the same base type and you manipulate them mostly as abstract objects, then you need a factory. [9]

[1] ***Applicability:*** Classes that are parallel in hierarchies usually require objects of one hierarchy for creating appropriate objects from another. Factory methods are mostly used in frameworks and toolkits where library codes required creating the objects of specific types that may be sub classed by applications with the help of framework. The Factory patterns can be used in following cases:

1) When a class does not know which class of objects it must create.
2) A class specifies its sub-classes to specify which objects to create.
3) In programming language (very raw form), you can use factory pattern where you have to create an object of any one of sub-classes depending on the data provided.

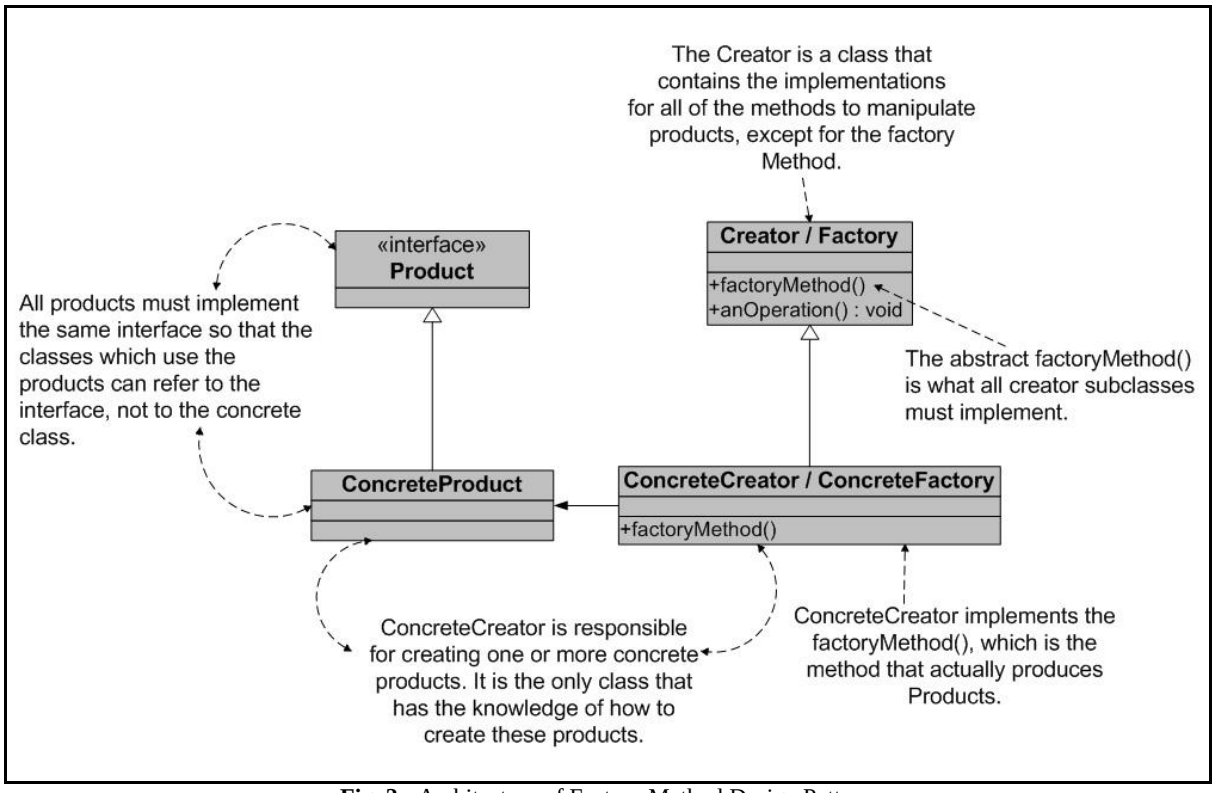

Fig. 2: Architecture of Factory Method Design Pattern

[2] ***Examples:*** Some examples in which factory method pattern is being used are sited below.
A graphical application works with shapes. In our implementation the drawing framework is the client and the shapes are the products. All the shapes are derived from an abstract shape class (or interface). The Shape class defines the draw and move operations which must be implemented by the concrete shapes. Let's assume a command is selected from the menu to create a new Circle. The framework receives the shape type as a string parameter; it asks the factory to create a new shape sending the parameter received from menu. The factory creates a new circle and returns it to the framework, casted to an abstract shape. Then the framework uses the object as casted to the abstract class without being aware of the concrete object type.The advantage is obvious: New shapes can be added without changing a single line of code in the framework (the client code that uses the shapes from the factory).

[3] ***Drawbacks and Benefits:*** Here are the benefits and drawbacks of factory method pattern:

- The main reason for which the factory pattern is used is that it introduces a separation between the application and a family of classes (it introduces weak coupling instead of tight coupling hiding concrete classes from the application). It provides a simple way of extending the family of products with minor changes in application code.
- It provides customization hooks. When the objects are created directly inside the class it's hard to replace them by objects which extend their functionality. If a factory is used instead to create a family of objects the customized objects can easily replace the original objects, configuring the factory to create them.
- The factory has to be used for a family of objects. If the classes doesn't extend common base class or interface they cannot be used in a factory design template.

B. Abstract Factory Pattern

Abstract factory pattern provides an interface for creating families of related or dependent objects without specifying their concrete classes. Modularization is a big issue in today's programming. Programmers all over the world are trying to avoid the idea of adding code to existing classes in order to make them support encapsulating more general information. Take the case of a information manager which manages phone number. Phone numbers have a particular rule on which they get generated depending on areas and countries. If at some point the application should be changed in order to support adding numbers form a new country, the code of the application would have to be changed and it would become more and more complicated.

In order to prevent it, the Abstract Factory design pattern is used. Using this pattern a framework is defined, which produces objects that follow a general pattern and at runtime this factory is paired with any concrete factory to produce objects that follow the pattern of a certain country. In other words, the Abstract Factory is a super-factory which creates other factories (Factory of factories).

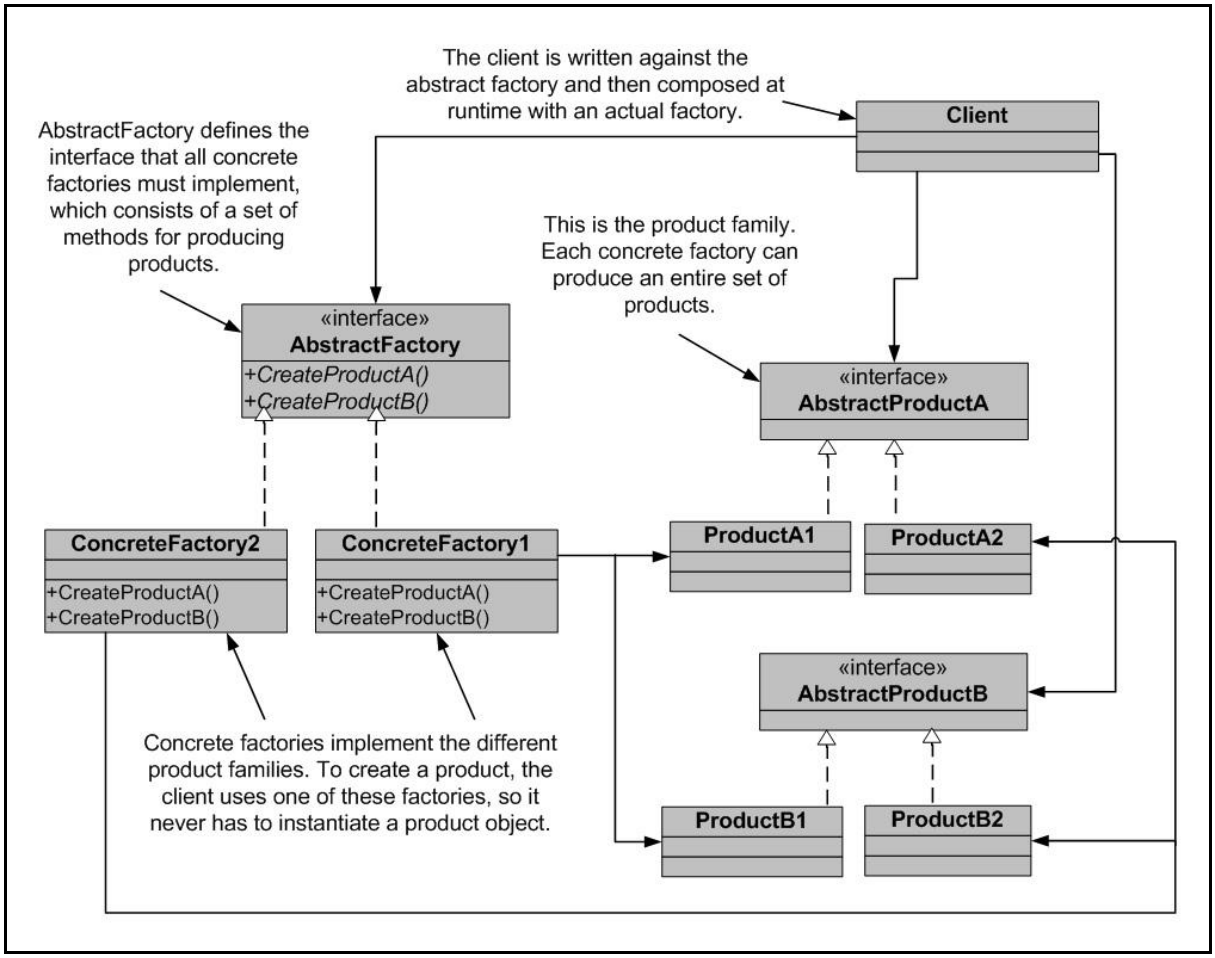

Fig. 3: Architecture of Abstract Factory Design Pattern

The AbstractFactory class is the one that determines the actual type of the concrete object and creates it, but it returns an abstract pointer to the concrete object just created. This determines the behaviour of the client

that asks the factory to create an object of a certain abstract type and to return the abstract pointer to it, keeping the client from knowing anything about the actual creation of the object. [10]

The fact that the factory returns an abstract pointer to the created object means that the client doesn't have knowledge of the object's type. This implies that there is no need for including any class declarations relating to the concrete type, the client dealing at all times with the abstract type. The objects of the concrete type, created by the factory, are accessed by the client only through the abstract interface.

The second implication of this way of creating objects is that when the adding new concrete types is needed, all we have to do is modify the client code and make it use a different factory, which is far easier than instantiating a new type, which requires changing the code wherever a new object is created.

AbstractFactory class declares only an interface for creating the products. The actual creation is the task of the ConcreteProduct classes, where a good approach is applying the Factory Method design pattern for each product of the family.

Extending factories can be done by using one Create method for all products and attaching information about the type of product needed. All factory patterns promote loose coupling by reducing the dependency of your application on concrete classes. Factories are powerful technique for coding to abstractions, not concrete classes. [2]

[1] ***Applicability:*** We should use the Abstract Factory design pattern when:

- The system needs to be independent from the way the products it works with are created.
- The system is or should be configured to work with multiple families of products.
- A family of products is designed to work only all together.
- The creation of a library of products is needed, for which only the interface is relevant, not the implementation.

[2] ***Examples:*** Some examples in which abstract factory pattern is being used are sited below.

Example-1: The purpose of the Abstract Factory is to provide an interface for creating families of related objects, without specifying concrete classes. This pattern is found in the sheet metal stamping equipment used in the manufacture of Japanese automobiles. The stamping equipment is an Abstract Factory which creates auto body parts. The same machinery is used to stamp right hand doors, left hand doors, right front fenders, left front fenders, hoods, etc. for different models of cars. Through the use of rollers to change the stamping dies, the concrete classes produced by the machinery can be changed within three minutes.

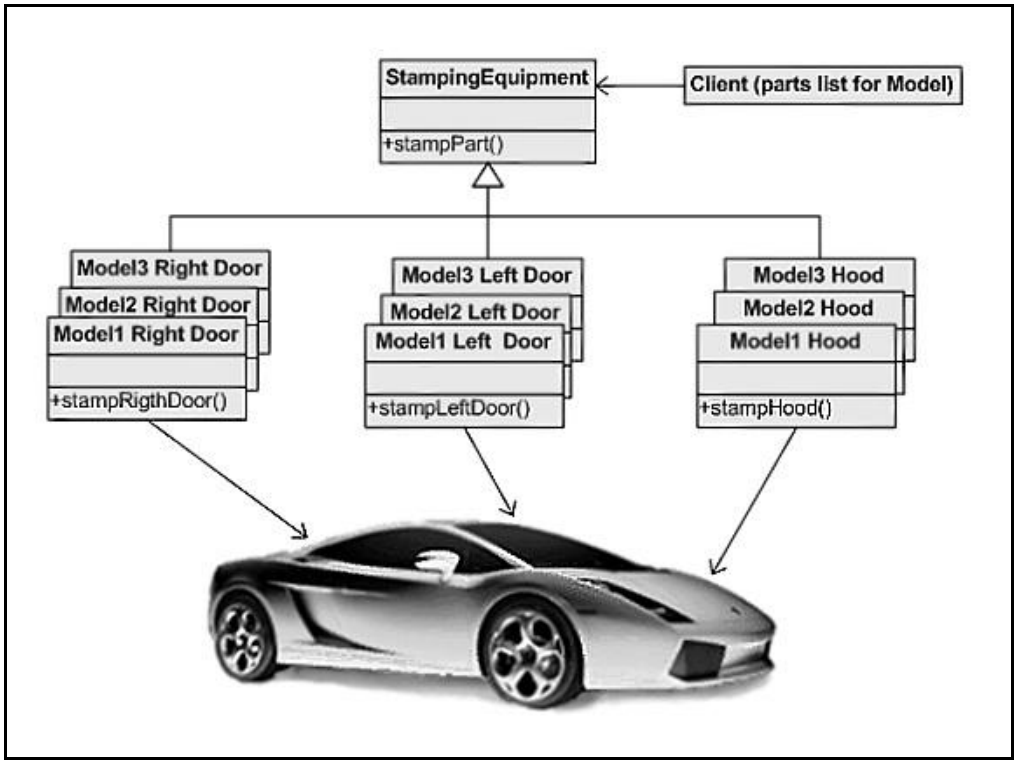

Fig. 4: Using Abstract Factory for Automobile

Example-2: Take the case of a information manager which manages phone number. Phone numbers have a particular rule on which they get generated depending on areas and countries. If at some point the application should be changed in order to support adding numbers form a new country, the code of the application would have to be changed and it would become more and more complicated. The AbstractFactory class will contain methods for creating a new entry in the information manager for a phone number and for an address, methods that produce the abstract products Address and PhoneNumber, which belong to AbstractProduct classes. The AbstractProduct classes will define methods that these products support: for the address get and set methods for the street, city, region and postal code members and for the phone number get and set methods for the number. The ConcreteFactory and ConcreteProduct classes will implement the interfaces defined above and will appear in our example in the form of the USAddressFactory class and the USAddress and USPhoneNumber classes. For each new country that needs to be added to the application, a new set of concrete-type classes will be added. This way we can have the EnglandAddressFactory and the EnglandAddress and EnglandPhoneNumber that are files for English address information.

C. Difference Between Factory Method Pattern and Abstract Factory Pattern

Abstract factory pattern provides an interface for creating families of related or dependent objects without specifying their concrete classes.

Table 2: Difference between Factory Method and Abstract Factory[4]

Factory Method Pattern	Abstract Factory Pattern
The factory method pattern defines an interface for creating an object, but let's subclasses decide which class to instantiate. Factory method lets a class defer instantiation to subclasses.	Abstract factory pattern provides an interface for creating families of related or dependent objects without specifying their concrete classes.
Factory method relies on inheritance: object creation is delegated to subclasses which implement the factory method to create objects.	Abstract factory relies on object composition: object creation is implemented in methods exposed in the factory interface.
The intent of factory method is to allow a class to defer instantiation to its subclasses.	The intent of abstract factory is to create families of related objects without having to depend on their concrete classes.

IV. INCORPORATING FACTORY DESIGN PATTERN FOR DATABASE CONNECTION INTO STRUTS FRAMEWORK

Factory pattern can be used within struts framework as shown in the below diagram to manage database connection objects.[11] Database connection management and exception handing is important concern for web application developer. Different modules and functionalities of web application use different DAO to connect with the database. Factories can be created so that appropriate DAO object can be created or retrieved as and when required by passing name of the appropriate DAO.

Sequence to work with the database connection and to retrieve the desirable result is given below.

(1) When user submits JSP form, form parameters will be sent to the web server along with the HttpRequest by post or get method.
(2) HttpRequest with *.do extension will be mapped to the appropriate Action Servlet / Request Processor.
(3) Action Servlet/Request Processor lookups for appropriate Action and Action Form Bean classes from struts-config.xml file.
(4) Action Servlet/Request Processor instantiates Action class and calls its execute() method by passing ActionMapping, ActionForm, HttpServletRequest and HttpServletResponse as parameters.
(5) Action Servlet/Request Processor can use Form Bean class and fills it with business data.
(6) Appropriate DAO object can be retrieved within execute() method using IUserDAO dao=(IUserDAO)DAOFactory.getDAO("UserDAOImpl");
(7) Required method can be called on this DAO object i.e. to create new user, to edit user edit userrole=dao.checkLogin(username, password);
(8) Inside this checkLogin method, connection object can be retrieved by calling methods of DBUtil class. i.e. C=DBUtil.getConnection();
(9) Appropriate queries can be fired by using queries defined within Queries class. i.e. PreparedStatement ps=c.prepareStatement(Queries.CHECK_LOGIN);

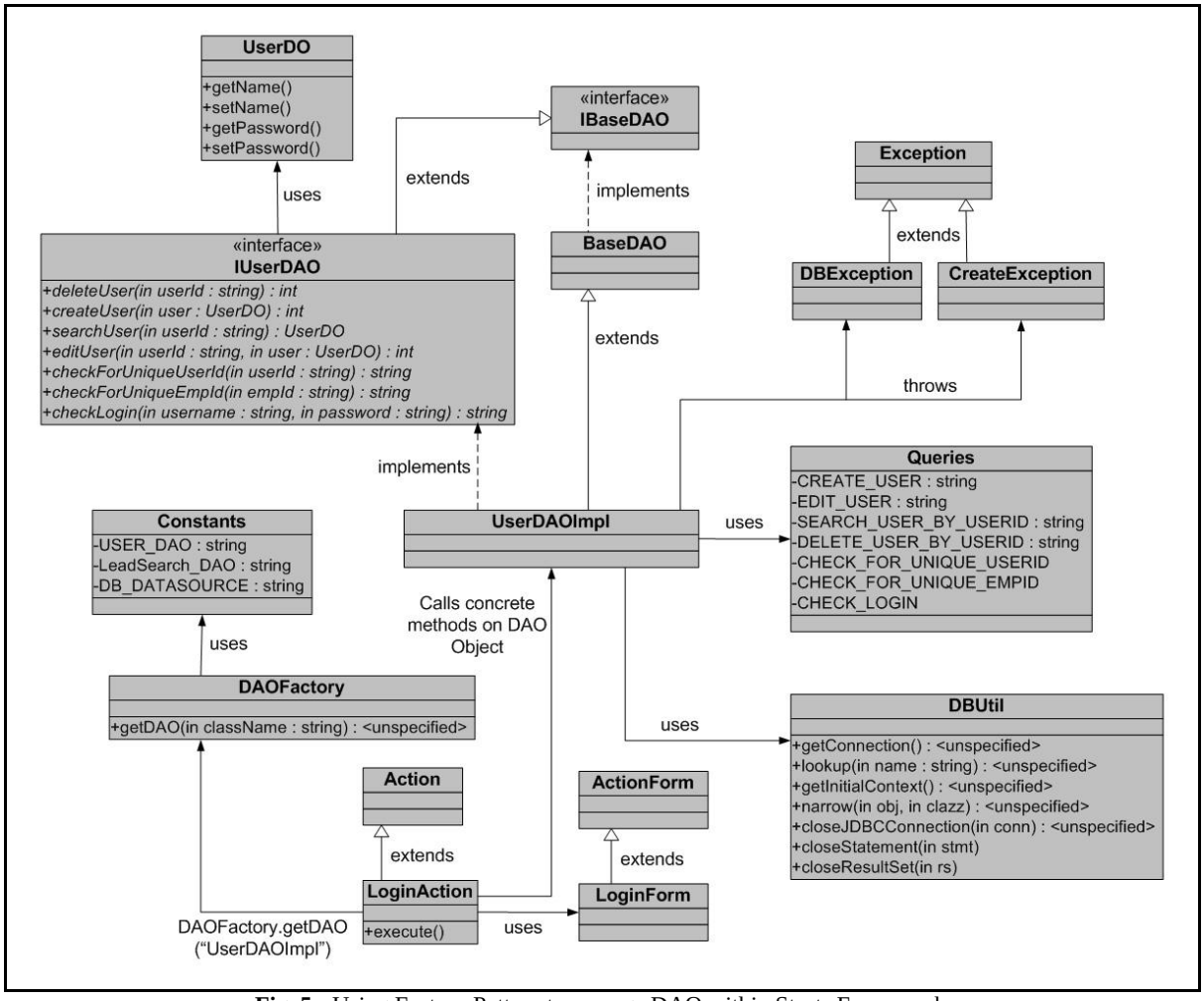

Fig. 5: Using Factory Pattern to manage DAO within Struts Framework

V. CONCLUSIONS

By using factory pattern for database connection, we can achieve weak coupling instead of tight coupling hiding concrete classes from the application. The main reason for which the factory pattern is used is that it introduces a separation between the application and a family of classes. It provides a simple way of extending the family of products with minor changes in application code. When the objects are created directly inside the class it's hard to replace them by objects which extend their functionality. If a factory is used instead to create a family of objects the customized objects can easily replace the original objects, configuring the factory to create them. As shown in the above figure, new DAO and new DAOImpl can be easily added to the existing architecture with minor changes in the application time. Using design pattern saves maintenance and development time and cost significantly.

ACKNOWLEDGMENT

The author would like to express his earnest gratitude to his guide Jayesh N. Rathod, Head of Computer Engineering department at Atmiya Institute of Technology & Science, Gujarat, INDIA for his constant guidance, encouragement and moral support which helped the author to accomplish this research work. The author would also like to extend his sincere gratitude to Mr. Jaladhi D. Joshi (M.S. (CE), Florida Atlantic University, Boca Raton, FL, USA) Who is working as Research Assistant at Motorola Inc., Sunrise, FL, USA for helping him in all possible ways.

REFERENCES

[1] MU Huaxin & JIANG Shuai, Design Patterns in Software Development, Beijing, IEEE 2011 (ISBN: 978-1-4244-9699-0)

[2] Fernando Barros, Increasing Software Quality through Design Reuse, 2010 Seventh International Conference on the Quality of Information and Communications Technology, 2010 (ISBN: 978-1-4244-8539-0)
[3] Chuanjun Li & Qing Wang & Wenwen Cai & Jun He, An Efficacious Software Design Method Based on Pattern and Its Application, National Key Technology R&D Program Project Grant, 2010 (ISBN: 978-1-4244-7324-3)
[4] Head First Design Patterns by Eric Freeman, Elisabeth Robson, Bert Bates, Kathy Sierra, O'Reilly Media inc.
[5] Head First Object-Oriented Analysis and Design by Brett D. McLaughlin, Gary Pollice, Dave West, O'Reilly Media inc.
[6] Professional Struts Applications: Building Web Sites with Struts, Object Relational Bridge, Lucene, and Velocity by John Carnell, Jeff Linwood, Wrox Publication
[7] Liu Chao, He Keqing, Liu Jie and Ying Shi, Some Domain Patterns in Web Application Framework, Proceedings of the 27th Annual International Computer Software and Applications Conference (COMPSAC'03), IEEE 2003 (ISBN: 0-7695-2020-0)
[8] Shu-qiang Huang & Huan-ming Zhang, Research on Improved MVC Design Pattern Based on Struts and XSL, Shanghai, IEEE, 2008 (ISBN: 978-1-4244-2727-4)
[9] Web Reference (http://en.wikipedia.org/wiki/Factory_method_pattern)
[10] Web Reference (http://en.wikipedia.org/wiki/Abstract_factory_pattern)
[11] Xia Sun & Chang Yao, Research and Implementation of Data Access Framework Base on Design Patterns, Hangzhou, IEEE 2011 (978-1-4577-1085-8)

ABOUT THE AUTHOR

Mr. Mukesh Parsana completed his graduation in Information Technology discipline from one of the most reputed engineering colleges in India, Nirma Institute of Technology Ahmedabad. Currently he is pusuing his master degree in Computer Engineering form Atmiya Institute of Technology and Science Gujarat. He has worked with India's top multinational software firm, Infosys Technologies, for four years as senior software developer. Sun Microsystems/Oracle awarded him with 3 international certificates (SCJP, SCWCD and SCBCD) for his expertise in different J2EE technologies. Due to his expertise on j2EE technologies, he has received offer of employment from multinational companies like Wipro Technologies, TCS, IBM and Persistent Technologies.

SCJP: Sun Certified Java Programmer for Java 2 Platform 1.4 (CX-310-035)
SCWCD: Sun Certified Web Component Developer for J2EE V1.4 (CX-310-081)
SCBCD: Sun Certified Business Component Developer for J2EE V1.3 (CX-310-090)