Mphasis
Unleash the Next

FlexiUtil

Do it yourself Model Driven Development

Ananda Joshi
Architect Advanced,
Enterprise Web Technologies, Mphasis

Table of Contents

Prologue

The methodology of Model Driven Development (MDD) is not new. The advantages of using MDD are innumerous. However, there are not many development projects that use the tools and the methodology. Some of the reasons could be:

- The licensing cost and hardware requirements for the tools are a very high initial investment
- There will be recurring cost, even after development which increases operational cost
- The tools require special training and there are not many trained resources available in the market
- While MDD helps in having fewer resources to deliver large volume of output, the experience and the skill level of these resources are expected to be very high.
- Customization of tool (if at all needed) is complex and requires long turnaround time

The other factors that may have to be considered:

- Difficult to start or exit the methodology in the middle of the project
- Difficult to migrate from one tool to another

In general, the objective of MDD tools is to separate the functional aspects from the technical implementation. They takeout most of the coding effort; thereby improving turnaround time and maintainability.

The focus of this paper is not to build an MDD tool that would be comparable to a commercial one, it is to identify the problems faced by the project and how best we can address them by changing our design approach.

Most common factors affecting project cost

In absence of MDD, the typical problems faced as part of project execution are:

- Development of discrete technological components
- More code usually results in less maintainability
- Skilled resources of all technological components are needed during most part of the development and testing.

This is further explained in the sections that follow, with a simple and widely used framework as an example: Struts-based web application

There are plenty of open source frameworks, 3rd party products that address many architectural and design patterns and have been widely accepted by the development community. While they make development easier, the project team still has to develop the technological components as needed by the frameworks to address the business functionality.

The proposed approach does not mandate or prescribe any additional tools or frameworks, but sticks to the finalized framework. Whatever is decided for the project is more than sufficient.

While the sample application uses struts-based architecture, the concept can be applied to applications using Dot Net based technologies too.

Struts-based Web Application

The Struts, as you may already know, is a framework that implements Model-View-Controller architectural pattern and helps web applications to standardize the development. While it provides several utilities and plug-ins that can be used directly, the project team will have to develop / hand-code the key technological components such as Action, ActionForm (in Struts 2, this may be combined with Action) and JSP for each set of business functionality. As part of multi-layered architecture, they also need to develop the corresponding services and DAO.

Let us assume that a web application requires Customer Search, Installer Search, and Device Search functionality. The search functionality should provide a web form that accepts a set of criteria. The criteria may vary for each functionality. They need to execute a query on corresponding table(s) and display a set of matching records as table.

Technical Components/Artifacts	Function	Device Search	Installer Search	Customer Search
	View	Device Search JSP	Installer Search JSP	Customer Search JSP
	Model	Device Search Action From	Installer Search Action Form	Customer Search Action Form
	Controller	Device Search Action	Installer Search Action	Customer Search Action
	Service	Device Search Service	Installer Search Service	Customer Search Service
	DAO	Device Search DAO	Installer Search DAO	Customer Search DAO

Figure 1: Sample Web Application Using Struts

The diagram, as shown in Figure 1: Sample Web Application Using Struts, shows the artifacts involved in implementing search functionality of different entities. For a Struts-based architecture, in the traditional approach, each functionality is implemented with its own View, Model and Controller. In addition, it may have used separate set of services and DAO.

If you are involved in estimating and planning activities of the project, you may immediately recognize that you have to allocate time and resource(s), for each of these artifacts for development and testing. In most cases, the resources who work on the frontend development, and that of service and DAO layers would be different. You may end up having skilled resources blocked for each unit task, which is directly proportional to the number of artifacts being developed.

Development of discrete technological components

For every functional sub-module, you may end up developing five artifacts. In the above example, you may see 15 artifacts to be developed for the three functionalities. The project effort is directly proportional to the number of artifacts to be developed.

Increased coding results in lower maintainability

As the number of hand-coded components increase, the application maintainability reduces. For example, if you want to introduce pagination of results or to change the number of rows to be displayed per page, you will have to make changes to every JSP and DAO (at the least).

Over-allocation of skilled resources

The developers of UI components and server-side components may take up one task or the other sequentially. But they need to be allocated to the project until the development and testing are complete, at the minimum.

The cost of project due to above problems can be reduced by large extent, if not eliminated, if there is a way to keep functional requirements and technological needs separate. The technological sub-components can be developed once, and reused across functional sub-components.

FlexiUtils - The Concept

FlexiUtils is the name given to the framework that Mphasis developed implementing the concept. However, the focus of this whitepaper is the concept, not the framework.

With reference to the sample web application, the functional entities – the device, the installer, and the customer – are unique. The characteristics and the content of these entities are different. The details that goes in to each of functional sub-component in the above example may be different. For example, number of elements of html form, the model (Java object) used for the functional entity, and the validations to be performed may all be different.

However, if you look at the abstract level, they all appear to be similar in nature:

1. Display a web form to accept & validate input; look-and-feel are standardized
2. Transform the http input data to java objects; conversion of http parameter to Java attributes are standardized
3. Call the service, which in turn uses a DAO to execute required query; execution of query and retrieving results are standardized
4. Transform the fetched details back to output format; converting Java objects into http parameters are standardized
5. Display the result as a table in html; construction of the result table is standardized

Most part of the technological implementation is standardized and most part of the functional implementation is specific to each functional unit. The concept tries to take this standardization as an advantage and suggests that the technical implementations can be generic and operated with a functional model.

The concept involves following steps:

- Create a meta-model of the functionality
- Create technical components following all project standards, but keep them independent of functionality
- Combine them together to implement the required functionality, by injecting the reference of meta-model in to the technical component.

The Meta-Model

The meta-model, in general, will contain “what” part of the functional requirement. Since the architecture is multi-layered, the meta-model should support “what” is required in each of those layers. For example, following could be the simplest distribution of functional needs across the architectural layers:

Presentation layer:

- What is the page title?
- What are the form fields?
- What are the types of each form field?
- What are the columns to be displayed as part of search results?

Service layer:

- What user roles are allowed to search?
- Which Java object has to be used for the functionality?
- What validations are to be performed?
- What processing is required on the result, before presenting?

Data access layer:

- What query to be used for searching?
- What filter criteria needs to be applied?
- What columns are to be retrieved?

Technological Components

The technological components are the ones that implement the functionality; they cover "how" part of functional requirements. There may be one or more pieces that address each architectural layer and they may not have direct reference to a functional entity. Instead, they would use the instance of meta-model to execute / deliver the functionality.

For example, following could be a simple implementation in a JSP, assuming the right meta-model is passed on to it, based on the functionality being requested:

- Get the page title and display with right CSS and at appropriate position of the page
- Get the form, loop through the form elements. For each element, display it as text-box, drop-down, label, etc., as per the configuration.
- Get the list of commands and display them as buttons.

The JSP just serves its purpose, which is to render the html form as per the definition of meta-model. It need not have any reference to Device Search, Customer Search, or Installer Search functional entities directly. However, when the generated html is delivered to the browser, it just behaves as the intended function - Device Search, Customer Search, or Installer Search.

Similarly, the Action Class may implement the intended functionality by using the meta-model:

- Populate the Java object configured for the functionality
- Perform initial validations
- Invoke the service intended for the search functionality and get the search results
- Prepare the search results so that they are ready for display
- Forward to the JSP intended to display the search results

Implementation: The reference injection

This step is to inject the reference of the functional meta-model into the technical component. In case of frameworks like Spring, this may happen through the framework's capability. In Struts, the technical component may have to obtain the reference to the model explicitly.

If the above approach is followed, the effort is only required to develop the technical components as generic components once. The functional architecture for the example being discussed in this white paper may be revised as below:

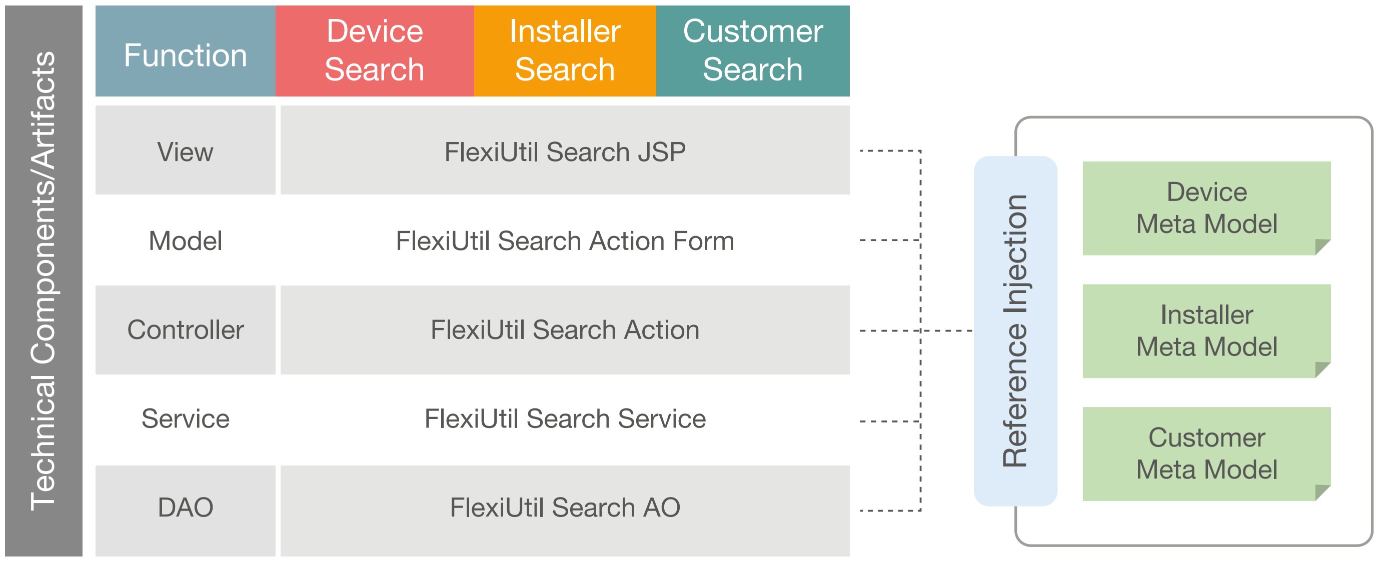

Figure 2: Revised Web Application - FlexiUtil concept

As shown, there is no deviation from the intended architecture or any compliance to the standards. The functionality, as needed, is still delivered without any compromise, but with fewer components.

In addition, the same set of components can be used for any new search related functional requirements. It does not require any changes to the technological components; only the corresponding meta-model needs to be created.

Alternate Implementation: Code generation

Once we have the functional specifications documented as meta-model, we may use the code generators as an alternate approach, which creates the JSP, Action, Service, DAO, etc., for each functional unit. They can be done either as a manual step as part of build process, or automated (using ant, for example). In this approach, the effort needs to be considered for the development of Code Generators.

The number of artifacts finally used in deploying the application may be same as that of traditional approach (refer Figure 1: Sample Web Application Using Struts). However, the development effort is limited to that of code generators.

Benefits

With better design and focused implementation, the approach helps in achieving following benefits.

Reduced Coding Effort

Whether you use Runtime Reference Injection approach or Code Generation approach, the effort is not proportional to the number of functional units.

Higher the number of functional units, more are the savings in effort. In most cases, the break-even of efforts can be achieved for reusing these components for 3 functional units. Anything beyond this would be direct savings of effort.

Improved Maintainability

Most functional changes need updates to the meta-models. For any global change that requires code modifications, only the generic components are to be modified. For example, if a new filter criteria is to be added to the Device Search functionality, it needs to be added to the meta-model. If a change that requires position of form buttons to be aligned to the right in all forms, then only one JSP needs to be changed, which in turn is applied to all functional units.

For any new enhancements, it requires only one such implementation, which can be then reused across all functional units. For example, if the search results are allowed to be exported as PDF and to be made available to all search functionality, the PDF rendering component is to be developed once. Then the feature is to be enabled through the meta-models.

Improved Resource Utilization

The effort from the actual developers of front-end or back-end components are limited to development of generic components. If planned appropriately, the number of resources required can be reduced quickly and need not be kept until end of the project.

The recurring effort is limited to creation of functional meta-models for each functional unit. Anyone with good functional understanding and with a brief training, can create the meta-models.

Other Benefits

- There is no risk of binding yourself to a vendor or tool. The approach does not require any tools, other than those required for the project.
- No deviations from the architecture are required. It is more of change in the design approach.
- There is no additional learning required.
- No investment on licensing cost, additional hardware

Mphasis Case Study

Mphasis has developed this concept as a framework called FlexiUtils. It also has Java-based implementation that supports Struts + Spring + Hibernate combination. This section describes case study of a project that uses FlexiUtils framework. The focus of the case study mentioned below is towards the challenges and achievements, in relation to the subject of the whitepaper.

The Client

The client is the world leader in home automation devices.

The Project

To enable Internet-based access for their home automation products using smart devices. There were two major sub-projects for the solution - the web application and messaging server. The web application includes a self-service portal and an admin portal. The messaging server is the one that allows communication between the smart devices and the automation devices.

The Technology

The web application uses Struts 2.x, Spring 3.x, Hibernate 3.x based architecture and uses RESTful web service for asynchronous messaging.

The messaging server uses Netty 3.x-based NIO processing for high scalability and concurrency management. Hibernate is used for data access.

Rest of the case study is with reference to the development of web application using FlexiUtil framework.

The challenges

- Availability of the skilled resources
- Duration of the project
- Too many internal and external dependencies
- Volatile requirements

How FlexiUtil Framework helped:

1. The development of JSP files was limited to one login page and one application page. They were ensured to comply with the standards of the project. All screens were rendered using the meta-model. This includes form rendering, data rendering, report rendering and anything related to the user interface. This drastically reduced the UI effort.
2. Only one action was used to render the page. Asynchronous functionality provided through Spring Controllers.
3. At the beginning of development phase, the UI design was not available. The team started the development with dummy screen layout and their own CSS. When the screens, style, branding was finalized, all we did was to transform the single JSP to the new style. In about 8 hours of time, all the developed artifacts worked absolutely fine with the new screens.
4. When requirements are added to convert the search results into PDF and spreadsheet, these capabilities were developed with FlexiUtil approach. The features were developed once and they were available to all search screens and reports. Only the corresponding buttons were added to the meta-model. No other change was required to the code.
5. The project functionality went through two major revisions in the first year after completion of the development. Each time the team turned it around in less than six weeks.

Conclusion

Bringing in Model-Driven Development approach does not always require additional tools. The core objective of separating functional model and technical implementations can be achieved with change in design approach.

The functional requirements can be built as meta-models, which include all the information needed for every architectural layer.

The technical implementations are focused towards development of generic components, instead of developing one component per functionality. They refer the meta-model for the functional behavior.

The approach helps in reducing the number of hand-coded components, thereby reducing the overall project effort. It also improves the maintainability as applying a global change would be limited to generic components and adding a new generic component can be used across the functional units. The approach also helps in better utilization of resources.

About the Author

Ananda Joshi is Architect Advanced as part of Enterprise Web Technologies practice, with over 26 years of experience; 13 of which are with Mphasis. He has extensive experience in architecting, design, and implementing the web applications of various complexity.

Developing reusable components is always his obsession. He has deep experience in providing solution architecture and other pre-sales support activities for Application Development and Maintenance projects. He also has experience in presales activities of large scale application transition and long term maintenance of COTS products and Custom Applications. His experience is around the business domains of Manufacturing, Supply Chain and Logistics. Ananda can be reached at Ananda.Joshi@mphasis.com.