Chapter 3

Response Time and Throughput
CHAPTER 3

Chapter - 3. Response Time and Throughput

<table>
<thead>
<tr>
<th>S.No.</th>
<th>Name of the Sub-Title</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Identifying design patterns</td>
<td>44</td>
</tr>
<tr>
<td>3.2</td>
<td>The Base Level Architecture</td>
<td>45</td>
</tr>
<tr>
<td>3.3</td>
<td>The New XWADF Architectural Pattern</td>
<td>47</td>
</tr>
<tr>
<td>3.4</td>
<td>XWADF Refactoring Algorithm</td>
<td>54</td>
</tr>
<tr>
<td>3.5</td>
<td>Case Studies</td>
<td>58</td>
</tr>
<tr>
<td></td>
<td>3.5.1 Hospital Management System</td>
<td>58</td>
</tr>
<tr>
<td></td>
<td>3.5.2 Library Management System</td>
<td>59</td>
</tr>
<tr>
<td>3.6</td>
<td>Summary</td>
<td>61</td>
</tr>
</tbody>
</table>
CHAPTER 3

Response Time and Throughput

This chapter focuses on proposing and implementation of an architectural pattern with underlying design patterns that leverage the performance of web applications in terms of access time and throughput. The proposed architectural pattern is named extensible Web Application Development Framework (XWADF) which incorporates appropriate design patterns to improve response time and throughput of web applications.

The XWADF is flexible in nature and can be extended to add more design patterns in future. In fact it is improved further incrementally to promote scalability, availability and fault tolerance. These details can be found in subsequent chapters. In this chapter the architecture is implemented using two case study applications namely HMS (Hospital Management System) and LMS (Library Management System) that exhibits the subtle difference in performance between the web application that uses our architecture and the web application that does not use it.

The architectural pattern is evaluated using metric as explored in [126] in order to compute average number of responses rendered for given unit time [127]. Another performance attribute by name Response time that indicates how long user waits to get response to a
request. The Response Time measure is taken as explored in [128] where it is said to have two types of Response Time namely fetch Response Time and render time.

The time taken to load web page is known as fetch Response Time while the time required to receive elements referenced by the loaded web page is known as render time. After proposing XWADF it is understood that an algorithm is required so that it can help developers to refactor existing web applications to conform to the architectural patterns to get benefits of underlying design patterns.

Towards this end a road map algorithm named “Refactoring Algorithm” is proposed in this chapter. This algorithm is testing using the case study applications named HMS and LMS. The empirical results revealed that with the proposed architectural pattern the response time and throughput of web applications, either newly built or refactored, increased significantly.

3.1 Identifying Design Patterns

The different design patterns that are identified for new XWADF architectural pattern for improving response time and throughput. The design patterns are given below.

Value Object (or) DTO (Data Transfer Object) Design Pattern
DAO (Database Access Object) Design Pattern
SQL mapping Design Pattern
Caching Design Pattern
Connection Pooling Design Pattern
Delegate Design Pattern
Decorator Design Pattern
POJO (Plain Old Java Object) Design Pattern

All the above design patterns are used in the proposed XWADF architectural pattern for improving response time and throughput.

3.2 The Base Level Architecture

The proposed architectural pattern focuses on the appropriate usage of design patterns [132] in different layers of MVC (Model View Controller). We identified various services and design patterns that can improve the performance of web application in terms of Response Time and throughput. However, the architecture is extensible to consider other performance and quality attributes such as scalability, fault tolerance, availability and maintainability in future.

Before presenting our architectural pattern, we felt the description of basic MVC architecture is appropriate here. Figure 3.1 presents MVC pattern which have plethora of advantages including maintainability, availability, scalability and so on. As can be seen in Figure 3.1, every request goes to a controller which is always a Servlet that can handle request.

However, the servlet does contain business logic (BL) and presentation logic (PL).
The BL is moved to model while the presentation logic is moved to view components in order to make the architecture maintainable. Controller invokes BL methods on model. Model layer interacts with database and gives response back to controller. Then the controller invokes suitable view to render response. Due to the clear separation of layers, this architecture realizes many advantages such as maintainability, availability, reduces development time and cost.

However, the MVC gives freedom to use any design patterns in View, Model and Controller layers in order to improve the performance of web applications. This fact has motivated us to propose a new architectural pattern based on MVC. The proposed architectural pattern is elaborated in section 3.3. The notations used in the processing of the above diagram are as follows.

Controller → C
Model → M
The processing is as follows

1. C ← Rp + Dv  [Rp and Dv both processed in Controller C]
2. M ← Bl + Dm  [Bl and Dm processed in Model M]
3. V ← Rg     [Rg is in View V]

Rg is response generation that is sent to the user as a response.

3.3 The NEW XWADF Architectural Pattern

This section throws light into the proposed architectural pattern for web application development for improving response time and throughput. The pattern is based on MVC described in section 3.2. Our architectural pattern incorporates various design patterns into M, V, and C layers. We identified the need for simple solution for model layer which can reduce the communication cost and improve throughput and response time.

We also explored connection pooling configuration, caching, decorator design pattern [133], Data Transfer Object (DTO), Database Access Objects (DAO) and other patterns. Figure 3.2 shows our architectural pattern.
In all layers of XWADF design patterns are proposed to improve throughput and response time of web applications. These patterns have plethora of advantages. However, in this paper we focus on their ability to improve response time and throughput. In the Model layer design patterns are used for returning values, database interaction, SQL mapping, caching and connection pooling.

Response time or Access time [129] is a measure that tells how long user waits to get response to a query. That is \( R_t = \frac{W_t}{res} \) where \( W_t \) is waiting time, and \( res \) is a response.

Throughput is the amount of work accomplished in a certain amount of time period. That is \( T_p = \frac{W}{T} \) where \( W \) is the amount of work and \( T \) is a Time Period.
When response time decreases the throughput increases and when response time increases, the throughput decreases.

**Value Object or DTO (Data Transfer Object) Design Pattern**

This pattern is best used when the underlying data storage is a relational database. The properties of this object correspond to columns in relational table. Thus it is suitable to hold a return value from database. When multiple records are to be returned multiple instances of this object can be added to a collection and returned to the controller. SQL Mapper program can map results of queries to Value Object or collection automatically. This will help in avoiding extra boiler plate coding in the application development [134].

**DAO (Database Access Object) Design Pattern**

This design pattern separates about low level data access operations from its high level business services. Thus it can be reused in applications as it is separated from other layers. It is dedicated for database interaction only. It is made up of three participants namely DAO interface, DAO implementation and VO or DTO or POJO [135].

**POJO (Plain Old Java Object) Design Pattern**

The POJO is meant for either data transfer from other layers to DAO or to return values back to higher layers [136].

**SQL Mapping Design Pattern**

We introduce a special design pattern that maps results of SELECT queries to required data objects like POJOs, collections of various kinds or a simple primitive type. A design pattern that ensures
this kind of mapping can help developers to reduce development effort and time by avoiding reinventing the wheel every time. By reducing lot of boilerplate code in web applications, this design pattern can improve the speed of web applications [137].

**Caching Design Pattern**

Cache is a portion of local memory which holds data objects which are frequently accessed from database. Web application performance and scalability can be improved through caching. Caching API helps increase performance in orders of magnitude in terms of response time and throughput. Caching reduces round trips to database by reusing the data present in local memory. Figure 3.3 shows communication diagram to illustrate the caching mechanism [138].

![Figure 3.3: Communication diagram illustrating caching](image-url)
Application hits database only when the object required is not in the cache memory. Since round trip to database is time consuming and costly, this solution can improve the performance of web applications. The performance of cache is measured using hit/miss ratio which is computed as number of cache hits divided by number of cache misses. A high hit/miss ratio reflects high performance of cache.

**Connection Pooling Design Pattern**

Establishing connection to database is costly and time consuming as it involves in a series of steps including protocol handshaking. Closing and opening database connections are not desirable. Moreover database vendors provide less number of connections per schema by default. These connections are exhausted as web applications are used by number of users concurrently.

To overcome this problem connection pooling is required. Connection Pooling is a phenomenon that maintains a set of pre-established connections to a database in a pool. The connections are never closed. They are available round the clock. When application needs a connection, the connection pool manager gives a connection.

Once the request processing is completed, the connection goes back to the pool and ready to serve the next request. This ways the performance of web applications in terms of response time and throughput is increased in large numbers of magnitude. Connection pooling usage can be compared with normal database connectivity as illustrated in figure 3.4 [139].
In case of direct connectivity user wait time is increased, connections scarcity arises and overall application performance goes down. With connection pooling (figure 3.5) these drawbacks are overcome as it makes the application more responsive and scale well for concurrent request processing. Connection pooling is recommended for high performance of enterprise web applications [140].

**Delegate Design Pattern**

Delegate design pattern helps an object to delegates its tasks to other objects instead of doing itself. This will help in achieving inversion of responsibility or to pass the buck for making the design
efficient. It reduces the complexity of the application as it follows the division of labor approach to divide the work into various helper classes [141].

**Decorator Design Pattern**

This design pattern [142] helps in improving the abilities of runtime object without making any changes in source code or modifying other instances. Decorators are very useful to add new responsibilities and features to runtime objects. When only some of the objects need new features this design pattern is preferred. Thus it can avoid having large number of sub classes in order to add new features to the object. Decorator provides lot of flexibility in adding features to existing objects [143].

The different notations in the Figure 3.5 is given by

Model \(\rightarrow\) M
Controller \(\rightarrow\) C
View \(\rightarrow\) V
Cp \(\rightarrow\) Connection Pool
P \(\rightarrow\) POJO
Da \(\rightarrow\) DAO Database Access Object
S \(\rightarrow\) SQL Mapper
Ca \(\rightarrow\) Cache
Dp \(\rightarrow\) Delegation Pattern
Dt \(\rightarrow\) DTO Data Transfer Object (or Value Object)
De \(\rightarrow\) Decorator
Req —> Request

Res —> Response

The processing of the above XWADF architectural pattern approach is as follows.

1. $M \leftarrow Cp + P + Ca$  [First Cp, P, Ca carried in Model M]
2. $M \leftarrow S + Da + Dt$  [Next S, Da, Dt carried in Model M]
3. $C \leftarrow Dp + Dt$  [Dp and Dt carried in Controller C]
4. $V \leftarrow De$  [Finally Response is generated in View V]

In the above process first Model uses the connection pooling next uses the POJO and so on finally View uses the Decorator Design Pattern.

### 3.4 XWADF Refactoring Algorithm

Existing web applications that do not perform well can be refactored to be XWADF compliant in order to leverage their performance. We designed a refactoring algorithm that can help convert existing web applications into the XWADF compliant applications. The algorithm guides developers to refactor successfully. This algorithm can be a basis for developing an automatic conversion tool in future.

//Algorithm: XWADF Refactoring Algorithm

**Inputs:** Existing Web Application

**Outputs:** XWADF Web Application

**Assumptions:** Java Web Application with Servlets and JSPs as Web Resources
Process:

1. START

2. If Web Application is in MVC Pattern Then
   a. Configure connection pool in web server/application server
   b. Implement a design pattern that gets connection from pool
   c. Define POJO for every relational table
   d. Move the Database interaction logic to DAO’s
   e. Implement SQL Mapper design pattern and use it in DAOs
   f. Implement Design Pattern for Caching
   g. Apply caching in Model layer
   h. Use Delegation Pattern and DTO in controller
   i. Identify responses to decorator in the view
   j. Use decorator pattern to decorate responses

3. If Web Application is not in MVC Pattern Then
   a. Move Presentation logic to view layer
   b. Move business logic layer to model layer
   c. Go back to step 1 to refactor as specified.

4. STOP

Figure 3.6: XWADF Refactoring algorithm
The refactoring algorithm guides developers to refactor their existing web applications to be compliant with our architectural framework XWADF. We have taken two case studies to demonstrate the effectiveness of the proposed architectural framework with suggested design patterns. The above algorithm is presented in the form of flow chart for easy understanding in figure 3.7
Figure 3.7: Flow chart for XWADF Refactoring algorithm
3.5 Case Studies

To evaluate the performance of web applications in terms of response time and throughput two case studies are used.

3.5.1 Hospital Management System

The first case study considered is Hospital Management System. This Hospital Management System allows users to download the prescriptions. Let us consider different set of prescriptions in the database. There may be many users who download the prescriptions. When there is only 1 user then the time taken to download that prescription is less. When the number of users increases then the time taken to download the prescriptions per user will be more.

Here the performance is evaluated in terms of response time and throughput. This downloading time depends on number of entities (records) in the database. When number of entities increases the downloading time also increases.

![Diagram of Users downloading data entities from hospital database](image)

Figure 3.8 : Users downloading data entities from hospital database
From the figure 3.8 the entities taken in the hospital database are 50000, 75000, and 100000. By using web container (MVC) & by using XWADF the response time and throughput values are determined by different users for different entities and related graphs are drawn. This response time and throughput values and resultant graphs are shown in chapter 6.

All resultant graphs are compared. The number of different users considered here are 1, 5, 10, 15, 20, 25, 30, 35, 40, 45, and 50. The software tool used for this purpose is LoadUIWeb 2. Here the throughput is determined by the amount of work accomplished in 1 sec time period per user.

### 3.5.2 Library Management System

The second case study considered is Library Management System. This Library Management System allows users to download the different kinds of books. Let us consider different set of books in the database. There may be many users who download the different kind of books. When there is only 1 user, then the time taken to download those books is less.

When number of users increases, then the time taken to download the books will be more. The performance is evaluated in terms of response time and throughput. This downloading time depends on number of entities (records) in the database. When number of entities increases the downloading time also increases
From the figure 3.9 the entities taken in the library database are 10000, 25000, and 50000. By using web container (MVC) & by using XWADF the response time and throughput values are determined by different users for different entities and related to that graphs are drawn. This response time and throughput values and resultant graphs are shown in chapter 6.

All resultant graphs are compared. The number of different users considered are 1, 5, 10, 15, 20, 25, 30, 35, 40, 45, and 50. The software tool used for this purpose is LoadUIWeb 2. Here the throughput is determined by when the amount of work accomplished in 1 sec time period per user.
3.6 Summary

Users always prefer the system with good performance because they need to download the items quickly. To get these items fast we need to get a good performance of web applications design. So in this chapter we proposed an architectural pattern XWADF comprising of design patterns, that will help to download the items with less response time comparing without using the proposed architectural pattern (i.e. MVC Architectural Pattern).

Similarly the throughput using XWADF is more when compared without using XWADF. One refactoring algorithm is proposed. The refactoring algorithm guides developers to refactor their existing web applications to be compliant with our architectural web application development framework XWADF.

The proposed architectural pattern XWADF is applied successfully for two case studies Hospital Management System (HMS) and Library Management System (LMS) and found that the proposed architecture is more useful in downloading the items fast comparing with without using XWADF architectural pattern that is using MVC.