5-2013

Efficient Architectures for Retrieving Mixed Data with Rest Architecture Style and HTML5 Support

Koushik Maddipudi
Western Kentucky University, koushik.maddipudi564@topper.wku.edu

EFFICIENT ARCHITECTURES FOR RETRIEVING MIXED DATA WITH REST ARCHITECTURE STYLE AND HTML5 SUPPORT

A Thesis
Presented to
The Faculty of the Department of Computer Science
Western Kentucky University
Bowling Green, Kentucky

In Partial Fulfillment
Of the Requirements for the Degree
Master of Science

By
Koushik Maddipudi

May 2013

EFFICIENT ARCHITECTURES FOR RETRIEVING MIXED DATA WITH REST ARCHITECTURE STYLE AND HTML5 SUPPORT

Date Recommended 28 March 2013

Dr. David Erbach, Director of Thesis

Dr. James Gary

Dr. Qi Li

4-30-13

Dean, Graduate Studies and Research Date

ACKNOWLEDGEMENTS

I personally thank Dr. David Erbach for his continuous support in completion of this thesis work. He helped me in designing the models and his thoughtful advice helped me to think in a right way to develop the thesis work.

I would like to thank Dr. James Gary for acknowledging and supporting my appeal. It would not have been possible to defend my thesis without his approval.

I would like to thank Dr. Qi Li for reading my thesis and suggesting corrections.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

EFFICIENT ARCHITECTURES FOR RETRIEVING MIXED DATA WITH REST ARCHITECTURE STYLE AND HTML5 SUPPORT

Koushik Maddipudi May 2013 101 Pages

Directed by: Dr. David Erbach, Dr. James Gary and Dr. Qi Li

Department of Computer Science Western Kentucky University

Software as a service is an emerging but important aspect of the web. Web Services play a vital role in providing it. Web Services are commonly provided in one of two architectural styles: a "**RE**presentational **S**tate **T**ransfer" (REST), or using the "**S**imple **O**bject **A**ccess **P**rotocol" (**SOAP**.)

Originally most web content was text and small images. But more recent services involve complex data structures including text, images, audio, and video. The task of optimizing data to provide delivery of these structures is a complex one, involving both theoretical and practical aspects.

In this thesis work, I have considered two architectures developed in the REST architectural style and tested them on mixes of data types (plain text, image, audio) being retrieved from a file system or database.

The payload which carries the actual content of a data transmission process can either be in Extensible Markup Language (XML) or JavaScript Object Notation (JSON). Both of these language notations are widely used.

The two architectures used in this thesis work are titled as Scenario 1 and Scenario 2.

Scenario 1 proposes two different cases for storing, retrieving and presenting the data via a REST web service. We investigate the question of what is the best way to

provide different data types (image, audio) via REST Web Service. Payload size for JSON and XML are compared.

Scenario 2 proposes an enhanced and optimized architecture which is derived from the pros of the first two cases in Scenario 1. The proposed architecture is best suited for retrieving and serving non-homogeneous data as a service in a homogenous environment.

This thesis is composed of theoretical and practical parts. The theory part contains the design and principles of REST architecture. The practical part has a Web Service provider and consumer model developed in Java. The practical part is developed using the Spring MVC framework and Apache CXF, which provides an implementation using JAX-RS, the Java API for RESTful services.

(A glossary of acronyms used in this thesis appears in the appendix on page 101.

CHAPTER 1

Introduction

1.1 Motivation and background

Web Service is a method for communication between two machines over web. The essence of web services emerges from Software as a service. Why would someone want to develop software from scratch unless the need is proprietary and does not match the offerings of a vendor?

Two important factors which are considered as a base for web services are reusability and interoperability. Two architectural styles are preferred in developing web services. **RE**presentational **S**tate **T**ransfer (**REST**) is a simple and widely used architectural style in providing software as a service. **S**imple **O**bject **A**ccess **P**rotocol (**SOAP**) is another architectural style or protocol in developing web services.

The original web services provided text and small images to consumers. As time passed, the services offered became more complex. Modern services as represented by REST and SOAP started providing text, images, audio and videos. The task of providing consumers with these complex data types is difficult for several reasons. One concern in almost all such services is the integration of heterogeneous data as a response to a client request. With the emergence of modern web there is a need to utilize existing services rather than building them from scratch. In parallel, new technologies like AJAX and JQUERY (a library for JavaScript) have evolved in presenting and utilizing serialized data which comes in as payload via XML or JSON.

1.2 Problem Statement and Objective

As described above, wrapping up different data types in a response payload is a typical task. I would like to propose a situation, expressing it in terms of a problem statement, and then disclose the objective of this thesis. Let us say there is a big company with many divisions. These divisions import data from many resources and services. The imported data can be in XML or JSON format.

The needed response to a client can be a mixture of text, image, audio and video files. Examining different ways to manage heterogeneous data and analyzing the core engines behind the architecture are the main objectives of this thesis. Based on performance, different architectures are examined and the best one is determined.

Developing application as a service is part science, but is also an art. The core engines and the architecture behind the web application are vital in delivering a response based on performance. As explained in the problem statement, if there is an extensive use of services by the divisions, an effective way of communication should be chosen without degrading the performance. We investigate the question of what is the best way to deal with different data types in proposed architectures.

1.3 Outline

This thesis is structured in the following manner.

Chapter 1 provided an overview of this thesis.

Chapter 2 gives a brief overview about the notions of web services. Different architectural styles are used in developing web services, with the focus on REST architectural style and message exchange formats used for communication between two or more services or systems. The main idea of REST architecture is explained in detail.

Chapter 3 talks about open source tools used in building the practical part of this thesis.

Chapter 4 concentrates on the practical part of thesis. Test cases of different scenarios and performance of three different architectures are discussed in detail. The best architecture is evaluated taking every possible detail into consideration.

Chapter 5 provides the conclusions and the possible future work.

CHAPTER 2

Overview of current web service standards

This chapter focuses on the web service standards and the notion behind the emergence of REST services. Section 2.1 formulates the basic communication channel between two systems, via HTTP (Hyper Text Transfer Protocol) and in a more secure way HTTPS (Secure). Section 2.2 explains how web services are standardized in new technologies and how services can be integrated with third party tools via Service Oriented Architecture (SOA). Section 2.3 explains the notion behind web services and explains the different architectural styles associated with the web services, mainly REST and SOAP. Differences between these two architectural styles are explained in detail and in Section 2.4; message formats used for communication are compared and explained in detail.

2.1 Hypertext Transfer Protocol (HTTP) and Secure (S)

HTTP is the basis of internet machine-to-machine communication. The functionality of HTTP can be explained as merely a request – response communication. This mode of communication is applicable for the client server architecture or computing model. In the web platform, HTTP is the standard communication protocol for client-server or request-response architecture.

In real world scenario, a client (browser / an application) sends a request to the server where the web site is hosted. The server receives a message in a certain format which carries the client's IP address. In response, the server replies to the client if the

requested page is found. There are many intermediaries which fall between the client and server communication. Services are built using the fundamental HTTP request interface.

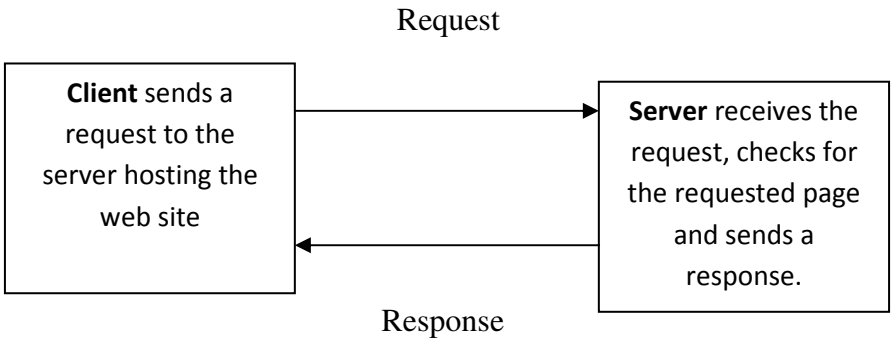

Figure 1: Simple Client-Server communication

A request message has following lines:

1. The requested content, **[METHOD] [LOCATION] HTTP/1.1**. The location refers to the location of the requested content in the server. We have several methods, PUT, DELETE, GET, POST etc. which can be used based on the type of request.
2. A list of headers such as Accept-Language: en.
3. An empty line. The empty line must consist of only <CR><LF> and no other whitespaces.
4. An optional message **[1]**.

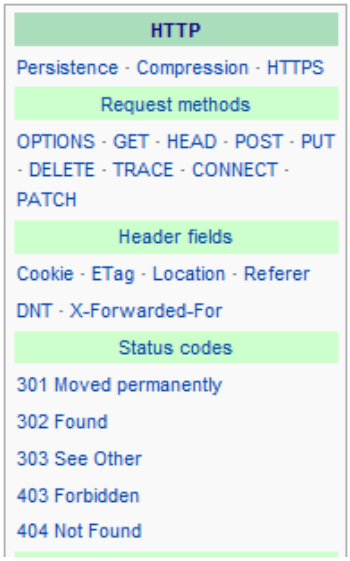

Figure 2: HTTP Template [2]

Request Methods (Verbs):

HTTP provides us with nine request methods. These methods are based on the actions or the requests which the client attempts. Most of the time we will be using four basic methods in the header file which are GET, POST, PUT, and DELETE. The remaining five are not extensively used. They are: HEAD, TRACE, OPTIONS, CONNECT, PATCH.

REST follows HTTP architecture. So, GET, PUT, DELETE and POST can be used by the REST client (service consumer) and performs these operations as part of a web service. GET is meant to be a safe operation. The four methods are used on resources. A Uniform Resource Identifier (URI) is used to identify a resource which may support all the four methods if the service provider has no objection.

For example, a film can be created, retrieved, updated or deleted. If the service provider restricts the service consumer from deleting a film, the service provider will not expose the DELETE method to the service consumer.

HTTP supports CRUD (Create, Read, Update and Delete) operations. HTML4 lacks the support for PUT and DELETE request through various elements. POST is overloaded with PUT and DELETE methods. HTML5 is trying to fix this issue.

An operation which produces the same result after calling it many times is said to be idempotent. POST is not idempotent. The remaining three methods are considered to be idempotent. Retrieving (GET) details about a particular film will have no change in the state of the database. The final value remains the same after execution of the PUT method. Deleting (DELETE) a film from the database will have no effects. The record gets deleted.

Hypertext Transfer Protocol Secure (HTTPS)

HTTPS is a combination of HTTP and SSL/TLS protocol. With HTTPS, encrypted and secure communication is possible. The communication with HTTPS creates a secure channel over an insecure network. Communication via HTTPS provides protection from man-in-the-middle attacks.

Differences between HTTP and HTTPS:

1. HTTPS URL's begin with https:// and HTTP URL's begin with http://.
2. With HTTPS data will be encrypted and not open to all users, whereas HTTP is open to man-in-the middle attack or eavesdropping.

HTTPS encrypts all the data sent to the server. It verifies the certificates of the server before allowing the communication channel to establish connection between the client and server. HTTPS does not cache any data. So the scope of eavesdropping is reduced.

METHOD	PURPOSE	SAFE	IDEMPOTENT
GET	Requests a specific representation of a resource identified by URI which is unique for every resource.	YES	YES
PUT	Create a resource or update a specific resource.	NO	YES
DELETE	Deletes the specified resource from the server.	NO	YES
POST	Submits data to be processed by the identified resource. Acts like an insert command in the database operations.	NO	NO

Table 1: HTTP Methods

HTTP status codes:

The status codes of HTTP have a range. The status code in the 200 range means "successful" and the status code in the 400 range means that the client has issued a bad request.

STATUS	DESCRIPTION	EXAMPLE
100	Informational	100 Continue
200	Successful	200 OK
201	Created	
202	Accepted	
300	Redirection	301 Moved Permanently
304	Not Modified	
400	Client error	401 Unauthorized
402	Payment Required	
404	Not Found	
405	Method Not Allowed	
500	Server error	500 Internal Server Error
501	Not Implemented	

Table 2: Status table [3]

There are headers which provide important communication concepts like redirection, content negotiation, security (authentication and authorization), caching, and compression. Using these headers, a better communication can be established between the client and the server.

EXAMPLE:

Sample conversation between HTTP client and server is explained below.

Client Request:

GET /index.html HTTP/1.1

Host: www.marketplace.com

The requested page is index.html and the HTTP version is 1.1.

Server Response:

HTTP/1.1 200 OK

Date: Mon, 27 Feb 2012 22:30:34 GMT

Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)

Last-Modified: Mon, 27 Feb 2012 20:30:34 GMT

Etag: "3j90f-1b6-3e1cb89b"

Accept-Ranges: bytes

Content-Length: 338

Connection: close

Content-Type: text/html; charset=UTF-8

Explanation on Server Response message:

From the response message, the content type is text/html and the content length is 338 bytes. “Etag” is related to caching concept. Server information is also provided in the response. The status of the request is 200, which means the client made a valid request. Browser renders the response and displays the message to the client.

2.2 Service Oriented Architecture (SOA)

In this section, I will be defining what SOA is and how web services are utilized with architecture standards.

SOA is an architectural style which enhances a business platform by integrating several services that are developed by others. Most major vendors like Oracle and IBM have developed SOA and published their view and architecture guidelines for a business organization to implement it.

The goal of Service Oriented Architecture is to bridge the gap between information technology and business organizations. SOA is basically an architecture which can bring savings to IT and business organizations in terms of cost. According to IBM, **"**SOA is considered as an enhancing factor in achieving better alignment between business and IT, creating more flexible and responsive IT infrastructure and simplifying integration implementation **[4]."**

"The policies, practices, frameworks that enable application functionality to be provided and consumed as sets of services published at a granularity relevant to the service consumer. Services can be invoked, published and discovered, and are abstracted away from the implementation using a single, standards-based form of interface. (CBDI) **[5]."**

Based on the modern standards, Business Integration has evolved by making heterogeneous computers, platforms, and applications work together, which wouldn't have been possible without web services and SOA. Previously, interoperability was a concern for applications to communicate. Standard protocols like SOAP made it possible

by easing the communication between applications or services with the standardized language XML.

“IBM sees Service-Oriented Architecture (SOA) as key to interoperability and flexibility requirements for its vision of on demand business **[6]**.”

Web Service approach:

Web services can implement Service Oriented Architectures. Functional building blocks are made accessible via HTTP / protocols with web services. These services can be on legacy applications or wrapped up on new applications.

The mandatory building blocks of SOA are:

Service provider: A service provider creates a web service and publishes it. It’s up to the provider about what functionalities of the services are to be exposed. A provider is also responsible for building a way to expose the functionalities securely and moreover availability of the functionality should be made easy. The provider registers the service and the customer can look up a service registry for the services.

Service broker: Service broker is meant to be a registry of services. Basically it stores information about the services and who are eligible to use them. It is more appropriate to call it a web service registry. Other parties discover the services and try to bond the connection. Sometimes, it serves as a documentation of services.

Service consumer: Service consumer or client approaches the service broker to test and match the requirements of the client. The consumer binds with the provider via a broker. When a service consumer finds an appropriate service, they have to take it to the broker,

bind it with the appropriate provider, and then use it. This can also be possible with multiple services at a time.

2.3 Web Services

This section gives a detailed explanation about the notion of a web service and different established protocols used in the building of web services with REST or SOAP. Section 2.3.2 Resource Oriented Architecture, explains what a resource is, in terms of a service and how different applications can access the available resources. In Section 2.3.3, Representational State Transfer, gives a detailed explanation on REST architectural style; when and why should REST be preferred over SOAP. In the later part of this section REST and SOAP are compared.

2.3.1 Notion

Web Services emerged from the notion of Software as a Service. A company can have any number of divisions and each division exchanges data with other divisions through a medium. Different scenarios can be considered in this approach. The exchanged data can be an image, audio, video, plain text, information or any kind of electronic data.

Instead of developing a new application for every division to get access to each other's data, developing or providing an application as a Web Service is an intelligent approach and saves a lot of work and time. W3C defines Web Service as **"**A software system designed to support interoperable machine-to-machine interaction over a network**"**.

There are two established architectural styles in developing web services. They are:

- Representational State Transfer (**REST**)
- Simple Object Access Protocol (**SOAP**)

There are three main roles in web service architecture. They are Service Consumer, Service Provider and Service Broker.

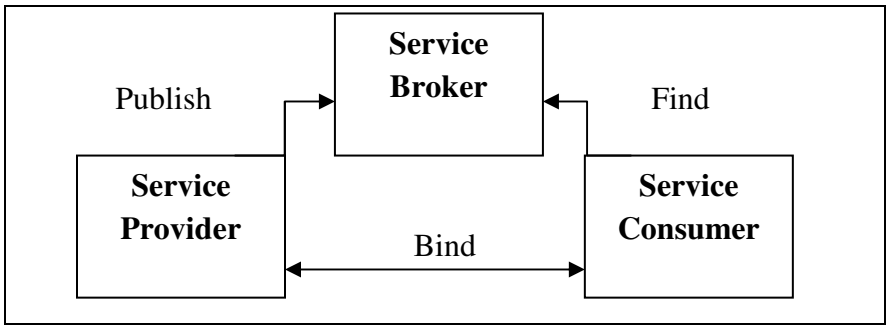

Figure 3: Basic Control flow of Web Services.

REST uses the basic HTTP functionality in accessing and providing the services. SOAP has some overheads compared to REST. **"**Amazon has both SOAP and REST interfaces to their web services, and 85% of their usage is of the REST interface **[7]."** Important differences between REST and SOAP are pointed in Section 2.3.4.

2.3.2 Resource Oriented Architecture

"If we lack the ability to identify, we lack the ability to signify".

From the book "RESTful Web Services", Leonard Richardson and Sam Ruby described Resource Oriented Architecture as a set of guide lines based on REST architecture style implementation. "We came up with a new term because REST isn't architecture: it's an architectural style, a way of judging architectures." From the book

"Evaluating Software Architectures: Methods and Case Studies" an architectural style describes a class of architectures or significant architecture pieces.

Universal Resource Identifier (URI) a type of Universal Resource Locator is a unique id given to identify / represent a resource. When a browser or an application makes a request to a service via URI, the corresponding service gets invoked and the client receives the response as HTML, XML or JSON. REST relies on verbs, namely GET, PUT, POST and DELETE. A client can GET information from the resources through the URL. Insertion or updating information is possible via PUT or POST methods, removing information from the resource by DELETE method. This covers majority of the REST definition but still REST has much more to say about it. REST is not about Create, Retrieve, Update and Delete (CRUD) operations. With above definitions, URL's are being named after verbs which eliminates using nouns as names of the resources. With Resource Oriented Architectures, a set of design principals should be followed in order to name the resource (nouns), identify the resource, and determine what actions (verbs) to be performed on the resources.

Following are the benchmarks of ROA:

Resource: Resources which are considered to be important should be identified.

Resource name: Identified resources should be given a name, obviously a noun.

Resource representation: Current state of the resource should be represented.

Resource link: Link to another or same resource representation should be provided.

Resource interface: A well-defined uniform interface should be presented in order to communicate with the resource.

Below table provides you with a list of HTTP operations that can be used on resource methods.

Resource method	Description	HTTP Operation
createResource	Create a resource with URI.	PUT
getResourceRepresentation	Get representation of the resource.	GET
deleteResource	Delete the resource.	DELETE
modifyResource	Modify the resource.	POST
getMetaInformation	Get Meta information about the resource.	HEAD

Table 3: Resource Methods and HTTP Operations [8]

2.3.3 Representational State Transfer (REST)

REST stands for Representational State Transfer. It is neither a technology nor a standard. It's an architectural style with a set of guidelines which are used in exposing the resources. A resource here can be an application which accepts or provides certain operations or action types like GET, PUT, POST and DELETE on or by itself. Each action on the resource can be identified as a resource with a unique id URI

The basic communication explained in Section 2.2 HTTP and HTTPS, which follows client-server architecture, is a perfect way of defining a RESTful web service. The URI's identify the resources hosted on different servers and HTTP is the protocol through which the communication is taking place. A client, like a web browser / a web application can access the resource with GET, PUT, POST, DELETE and other methods.

The REST architectural style is based on the following principles **[9] [10].**

1. **Client-Server:** In order to implement REST based architecture, the client and server are completely independent of each other. They are separated by a Uniform Interface. Clients are not concerned with data storage, Servers are not concerned with user interface, so portability of clients can be improved and scalability of the server can be enhanced.
2. **Stateless:** The communication between the client and the server is considered to be stateless. Each request, from any client, contains all the information needed to process the request. HTTP request should happen in complete isolation. The server should never rely on information from previous requests. In fact the server cannot maintain any details about the client.
3. **Cacheable:** A client can cache responses. Therefore, responses should implicitly or explicitly define themselves as cacheable. This improves the scalability and performance of the application.
4. **Layered System:** A client should not be able to tell who the end server is. There can be intermediary servers which improve load balancing techniques providing shared caches.
5. **Uniform Interface:** A UI between a client and a server simplifies and decouples the architecture.

   Architecture is said to be REST styled if all the above constraints are met.

Fielding describes REST's effect on scalability as

"REST's client–server separation of concerns simplifies component implementation, reduces the complexity of connector semantics, improves the

effectiveness of performance tuning, and increases the scalability of pure server components. Layered system constraints allow intermediaries—proxies, gateways, and firewalls—to be introduced at various points in the communication without changing the interfaces between components, thus allowing them to assist in communication translation or improve performance via large-scale, shared caching. REST enables intermediate processing by constraining messages to be self-descriptive: interaction is stateless between requests, standard methods and media types are used to indicate semantics and exchange information, and responses explicitly indicate cacheability **[9]**."

RESTful Web Services:

A RESTful web service is implemented using HTTP and the principals of REST. It is a collection of resources with four defined aspects:

1. The **base URI** for the web service

   Example:
   http://localhost:8090/Serviceone/app/filmservices/film/1

2. Internet Media Type supported by the web service. This is XML, JSON etc.
3. The set of operations supported by the resources via web services using HTTP (GET, PUT, POST, DELETE, etc.)
4. The API must be hypertext drive.

2.3.4 REST vs. SOAP

SOAP based web services:

In SOAP based web services, the service provider publishes the contract, Web Service Description Language (WSDL file) of the service where a customer can access it directly or by looking up the WSDL file in the registry. The WSDL file helps the client in developing the client code for calling a service. Many open source tools which are introduced and discussed in Chapter 3 offer web service framework which helps in designing WSDL document to interact with a web service.

The choice of opting for SOAP or REST depends on the requirements of building an application. If your requirements consist of transmitting simple XML message data formats it is preferable to choose REST because it has fewer overheads compared to SOAP. The overheads associated with SOAP are explained briefly later in this section. However, if the requirements needed to be negotiated between the provider and consumer a WSDL file and WS specifications should be adopted by an enterprise and SOAP communication is more likely to be a right option.

SOAP is not a replacement for REST. Both have their own benefits compared to each other. REST is easy to approach and is considered an architectural design. SOAP is an industry standard approach and is well defined protocol with well-established rules to communicate **[9]**.

Areas where REST approach can be used extensively:

1. **Limitation on bandwidth and identification of resources:** Response can be in any format and any browser can support the basic GET, PUT, POST and DELETE operations. The response can be parsed easily and applications using AJAX and JavaScript can find it more useful. This is one kind of architecture we have chosen to test in our thesis work.
2. **Stateless nature:** Best found in stateless operations, if any operations need to be continued SOAP is a good option, as it supports state full operations and transactions. If the requirement does not focus beyond CREATE, RETRIEVE, UPDATE and DELETE operations, REST is the best choice.
3. If the information / data are intended to be cached, stateless nature of REST is perfect.
4. REST uses a global address space, so each resource can be easily identified and used. The response will be served to the client who requested the resource. SOAP message are always addressed to the dispatcher and the dispatcher routes the data to different objects. REST is lightweight (not a lot of XML).

Considering SOAP over REST approach:

1. **Web Service- Reliable Messaging:** Reliability and security can be achieved by using SOAP specification in WS- * stack.
2. **Contracts:** If both, provider and consumer need a formal contract in exchanging data; Web Service Description Language (WSDL) provides a rigid way to establish the communication.

2.4 Message Exchange Formats

This section gives a brief introduction about the commonly used message exchange formats and their structures. In order for the communication to take place between a service provider and the service consumer, there should be a commonly understood message format. Communication need not be between a client and service; it could be between two services. The communication link between web service and the client can be via XML, JSON, plain text, HTML; image which is best depends on the type of request. XML and JSON are widely used data interchange formats for communication.

2.4.1 Extensible Markup Language (XML)

XML is commonly used markup language for exchange of messages between two services. It is widely used for encapsulating information in order to pass it between different computing systems. XML supports a wide range of data types. Image, video, audio files should be encoded in base64. XML provides a common envelope for inter process communication (messaging). Weather Services, e-commerce sites, RESTful services and data exchange services use XML for data management and transmission **[11]**. The internet media type for XML is **application/xml**.

It is not a good idea to send an encoded image in XML as the size of the encoded image is increased by 1.3 times the original size which is shown in the thesis work. The best way to deal with binary data in XML is to avoid it entirely. Instead it is advisable to send a URL of the content location which hosts the images. There can be a dedicated file server which hosts the images.

“In the case of HTML, the browser just makes another HTTP request to get the data included through elements like <img>. By not including the binary data directly in the XML, you avoid potentially wasteful text encodings and make it possible to implement other enhancements, such as the image caching most people love in their Web browsers **[12].”**

The following listing shows a sample XML document with a Base64 encoded image.

```
<?xml version="1.0" encoding="UTF-8"?>
<sample>
   <description>
   An embedded image file.
   </description>
   <image name="fun.jpg" encoding="base64"
      source="filehosting"
      href="http://localhost:8090/images/Service/”>
iVBORw0KGgoAAAANSUhEUgAAABAAAAAQ
CAYAAAAf8/9hAAAABGdBTUEAAK/INwWK
6QAAABl0RVh0U29mdHdhcmUAQWRvYmUg
SW1hZ2VSZWFkeXHJZTwAAAJOSURBVDjL
pZI9T1RBFIaf3buAoBgJ8rl6QVBJVNDC
ShMLOhBj6T+wNUaDjY0WmpBIgYpAjL/A
ShJ+gVYYYRPIony5IETkQxZ2770zc2fG
YpflQy2MJzk5J5M5z/vO5ESstfxPxA4e
rkJggg==
   </image>
</sample>
```

Listing: XML document with Base64 encoded image.

2.4.2 JavaScript Object Notation (JSON)

JSON is a lightweight message format used in data exchange format. It is derived from JavaScript scripting language which supports simple data structures. JSON can be used as a replacement for XML. JSON is extensively used in applications developed with AJAX. The internet media type is application/JSON. Yahoo! began offering some of its

web services in JSON. Google started offering JSON feeds for its GData web protocol in December 2006.

Basically, JSON message is a collection of name, value pairs. JSON also supports an ordered list of values. For AJAX applications, JSON is faster and easier than XML. In case of XML AJAX fetches the XML document, uses the XML DOM to look through the document and then extract the values and store them. It's a long process compared to JSON format. AJAX fetches a JSON string and then uses eval () the JSON string.

Syntax:

Data is in name/value pairs. They are separated by comma. Curly bracket holds name/value pairs and square brackets are used for storing array values. The following listing shows the format of the JSON payload.

```
{"Film":{"film_id":1,"title":"ACADEMY DINOSAUR","description":"A Epic Drama of a Feminist And a Mad Scientist who must Battle a Teacher in The Canadian Rockies","length":86,"image_id":1}}
```

Listing: JSON Format

There are many java libraries which provide support for JSON. We have used Google-gson for serializing the content with java objects.

CHAPTER 3

Open Source Tools and Frameworks

This chapter provides you with a detailed insight of all the open source frameworks, tools, plug-ins and standards used in the implementation part of thesis. Primarily, we have developed an application to test the core idea behind this thesis. Section 3.1 gives detailed information about specifications we have used, what framework we have chosen and why. This section also mentions the support provided by the framework to REST architectural style standards. Section 3.2 explains the MVC (Model, View and Control) architecture pattern, marshaling, un marshaling concepts and API's (Application Programming Interfaces) used for parsing XML and JSON languages that come as payloads from services to the client application. Section 3.3 references on data type support for MySQL database and Java programming language and the schema used in the development work.

A brief idea on the terminology, open source tools which are explained in the later part in different sections of this chapter:

Web Application: Used SPRING MVC for the development of application.

Web Service Server side implementation: APACHE CXF is chosen for implementing the JSX-RS semantics for the web application developed in Spring MVC.

Container support: Tomcat 7.0 is used as a container where application is deployed.

Data Binding: The output from the web service is XML or JSON payload. If XML is the payload generated, JAXB is used as the data binding tool and for JSON as payload, GSON, an open source API provided by Google Inc. is used.

Client side implementation (Web Service): JAX-RS does not provide support for client side development. We have chosen Jersey – client 1.12 for testing the service developed on server side by Apache CXF.

MySQL Workbench: Workbench IDE is used for creating the tables in the schema. It's as open source tool from MySQL.

IDE (Integrated Development Environment): Eclipse Indigo IDE is used for integrating all the API's and developing this application.

3.1 Web Services Framework

In this section we are going to see **JAX-RS** Java API for RESTful Web Services that provides support for creating a web service in RESTful architecture. We will also see how REST service is supported by JAX-RS and which open source frameworks support REST architecture according to the standards.

Java Community Process **JCP** is a mechanism for developing standard technical specifications for Java technology. JAX-RS is a specification defined from **JSR 311** (Java Specification Requests). JSR is an expert group which provides standard protocols for a specification. JAX-RS standardizes the way a RESTful service can be developed in JAVA.

Choosing a framework that provides better flexibility, good support in binding, marshalling and support for different message formats is a typical task. There are many frameworks which provide support for building, deploying and publishing a service. A wide range of frameworks which support JAX-RS specification are AXIS -1 and 2.x, Glassfish Metro, JBOSSWS, APACHE CXF etc.

Among these, we have chosen APACHE CXF because it provides support for building SOAP or REST oriented services. Also, provides better support for API's like JAXB (for marshalling XML payload) etc. There are three different ways of building a RESTful service using Apache CXF.

1. **JAX-RS:** A standard way of building a RESTful application using JSX-RS specification.

2. **JAX-WS provider and dispatch:** JAX-WS is SOAP oriented architectural style which helps in building a RESTful style service with the JAX-WS provider and dispatcher API's.
3. **HTTP Binding:** A flexible way of creating resources and mapping them to the services. This can be done by using annotations.

This thesis uses a simple way to build a RESTful application. CXF provides both server side development and client side development support.

Support for RESTful Services by Apache CXF:

JAX-RS specify the semantics to create web services according to the REST style. It does not provide any support for RESTful client. CXF goes a step further and provides various options to create clients that can interact with the JAX-RS specification. CXF also supports different data formats like JOSN and XML which are used in web 2.0 applications.

The key concept for developing a REST service is data binding. Data binding means, mapping java objects with message format. The payload can be in XML or JSON. Both the formats are supported. The message format of the payload can have different data types embedded into it. CXF provides JAXB, an API for XML binding. The payload for SOAP is XML and it does not have a choice whereas REST can choose XML or JOSN. A web service can send XML or JSON based on the request. Apache CXF supports both JSON and XML message formats.

SPRING framework:

It is an open source application framework for Java platform. It provides many extensions for building web applications on top of Java Enterprise Edition platform. It does not impose a programming model but it is a replacement to Enterprise Java Beans (EJB). EJB is a JEE specification and a server side model that encapsulates the business logic of the application. The SPRING framework features its own MVC design pattern. It provides more separation between the presentation layer and the business logic.

CXF provides first class integration with SPRING framework where a Plain Old Java Object (**POJO**) can expose a web service and leverages the services provided by Spring Framework. CXF also provides spring configuration support which eases configuration for deployment. CXF also provides support for Maven and Ant users. This thesis did not use Maven or Ant for deployment; we have chosen traditional way of developing and deploying the application. The Apache CXF provides a robust framework that makes web service development easy, simplified in a standardized way.

Deployment Descriptor:

In Java EE a deployment descriptor is a configuration file for an application or an artifact which is deployed to a container. It directs the deployment tool to deploy an application specifically according to the configuration requirements. XML is the language used in building the deployment descriptor. For a web application the deployment descriptor should be called ‘web.xml’ and placed under WEB-INF directory structure. Configuring an application is important because the application has to be deployed and executed according to the requirements.

We are using CXFServlet to process all the requests, which we are assuming to be the requests for web services. The below listing is the Deployment Descriptor web.xml file configured for the web application.

```
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">

        <display-name>Serviceone</display-name>
        <context-param>
                <param-name>contextConfigLocation</param-name>
                <param-value>/WEB-INF/*-context.xml</param-value>
        </context-param>
        <context-param>
                <param-name>log4jConfigLocation</param-name>
                <param-value>/WEB-INF/classes/log4j.xml</param-value>
        </context-param>
        <listener>
                <listenerclass>
           org.springframework.web.context.ContextLoaderListener
         </listener-class>
        </listener>
        <listener>
                <listenerclass>
           org.springframework.web.util.Log4jConfigListener
         </listener-class>
        </listener>
        <servlet>
                <servlet-name>cxf-rest</servlet-name>
                <servlet-class>
           Org.apache.cxf.transport.servlet.CXFServlet
         </servlet-class>
                <load-on-startup>1</load-on-startup>
        </servlet>
        <servlet-mapping>
                <servlet-name>cxf-rest</servlet-name>
                <url-pattern>/app/*</url-pattern>
        </servlet-mapping>
</web-app>
```

Listing: Deployment Descriptor (web.xml)

The service bean definition in the below listing tells us that the jax-rs service bean is defined in spring. The jaxrs:server part specifies about the 'film service' class. In the later part of the listing the bean is identified and mapped to the corresponding class.

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:jaxrs="http://cxf.apache.org/jaxrs"
        xsi:schemaLocation="http://www.springframework.org/schema/beans

        http://www.springframework.org/schema/beans/spring-beans.xsd
                        http://cxf.apache.org/jaxrs
                        http://cxf.apache.org/schemas/jaxrs.xsd">

        <import resource="classpath:META-INF/cxf/cxf.xml" />

        <!-- Specifies FilmService class as a RESTful resource -->

        <jaxrs:server id="filmRESTService" address="/">


    <!--Specifies the actual RESTful implementation classes -->

                <jaxrs:serviceBeans>
                        <ref bean="filmService" />
                </jaxrs:serviceBeans>

        </jaxrs:server>

        <bean id="filmService"
         class="scenarioone.usecaseone.services.FilmService">
        </bean>

</beans>
```

Listing: CXF-Servlet Configuration file

3.2 Other API's, tools and plug-in

This section explains Model View Controller (MVC) architecture in detail. Different API's used in the development of server side and client side application are explained. The later part gives you a detail explanation on the different tools chosen for marshalling, un-marshalling and parsing of XML, JSON message formats.

3.2.1 Frameworks used in server side application development

This section gives you a brief explanation on Model View Controller architecture pattern.

MVC is a software architecture that separates the presentation of information from the user's interaction with it. The model consists of application data and the controller mediates the request from the client to the appropriate model or view. The MVC divides the application into three different components. The controller formulates the interaction between model and view. The controller sends commands to the model to change the view associated with it, basically it's about the values which are to be displayed by the view screen.

Key benefits of MVC are:

1. **View-Model decoupling**

   Attach multiple views to a single model to provide different presentations.

2. **View-Controller decoupling.**

   Change the way the view responds to user input.

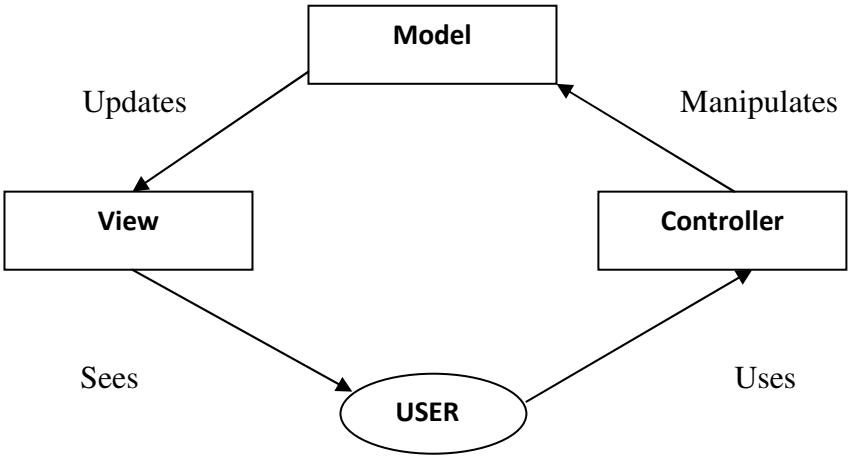

Figure 4: MVC architecture

3.2.2 API's used in client side application development

JERSEY REST CLIENT

JERSEY is an open source JAX-RS specification. JERSEY supports for REST Server and REST Client. The server side application is implemented by Apache CXF and the client part which is basically used for testing the service is developed by Jersey client. In order to implement this process we need to import or utilize **jersey-client** and **jersey-core** jar files. It is a light weight REST client.

The resource URL:

String **resource_url** ="http://localhost:8090/Serviceone/app/filmservices/film/" +id;

REST CLIENT plug-in

REST Client is an open source Java application provided as a plug-in by Mozilla Firefox for calling a REST application. It can be used to test a variety of services. JSON and XML message payloads are displayed as output. The time taken to call the web service and display the payload to the client window is displayed.

Following listing provides a methodology for invoking the service via a client:

```
try {

Client client = Client.create();
WebResource webResource = client.resource(resource_url);

 // for JSON
ClientResponse response = webResource.accept("application/JSON")
                                    .get(ClientResponse.class);

 // for XML
 ClientResponse response = webResource.accept("application/XML")
                                    .get(ClientResponse.class);

if (response.getStatus() != 200) {
throw new RuntimeException("Failed : HTTP error code : "
                                    + response.getStatus());
                            }

          String output = response.getEntity(String.class);
```

Listing: Calling a Web Service

So, now that the connection is established and the service is called, the response should be presented via JSP or HTML5. Now, based on the request, the response can be in XML or JSON. Both the formats should be parsed and presented by the client (web application). The client web application parses the message and displays it to the end user.

3.2.3 Tools used for marshalling, parsing message exchange formats

Marshalling and Un-Marshalling:

In computer science, marshalling is the process of converting or transforming the memory representation of object to data format suitable for storing. In the context of Web Services, marshalling moves the data between different applications. Un-marshalling maps the message format to a POJO classes. Later these POJO classes will be used in serving the view part.

PARSING:

Parsing or syntactic analyzing is a process of analyzing a text to determine its grammatical structure. The message should be well structured in order to be parsed. The combination of Java and XML helps developers in exchanging messages between applications. That's the reason behind the emergence of XML. Java technology provides a platform for building portable applications. XML and Java technologies are ideal building blocks of web services.

Java Architecture for XML Binding (**JAXB**) a framework which relies on annotations in the source code in order to map an XML file to a java object. Apache CXF by default supports JAXB. The following table gives a description on annotations that can be used on Java classes for structuring/marshalling of message.

Annotation	Description
XmlRootElement	Maps a class or an enum type to an XML element.
XmlElement	Maps a JavaBean property to a XML element derived from property name.
XmlAttribute	Maps a JavaBean property to a XML attribute.

Table 4: JAXB annotations

Parsing XML:

The XML message is appended to the "output" String. The JAXBContext class provides the client's entry point to the JAXB API. It provides an abstraction for managing the XML/Java binding information necessary to implement the JAXB binding framework operations: un-marshal, marshal and validate **[14]**.

The Unmarshaller class governs the process of deserializing XML data into newly created Java content trees, optionally validating the XML data as it is unmarshalled. It provides an overloading of un-marshal methods for many different input kinds. All the POJO classes will be equipped with the setter and getter methods. Therefore unmarshalling can be completed by following these steps.

```
public Film unmarshaling(String output)
 throws JAXBException, IOException {
InputStream is = new ByteArrayInputStream(output.getBytes("UTF-8"));
JAXBContext jc = JAXBContext.newInstance(Film.class);
Unmarshaller u = jc.createUnmarshaller();
Film film = (Film) u.unmarshal(is);
// Setter methods
return film;
```

Listing: Unmarshalling the content

A standard XML message submitted by a web service based on a request.

XML Message format:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Film>
<film_id>1</film_id>
<title>ACADEMY DINOSAUR</title>
<description>A Epic Drama of a Feminist
And a Mad Scientist who must Battle a Teacher
 in The Canadian Rockies</description>
<length>86</length>
<image_id>1</image_id>
  <image>
    <image_id>1</image_id>
    <name>1.jpg</name>
    <size>408307</size>
    <type>.jpg</type>
    <image_location>
      http%3A%2F%2Flocalhost%3A8090%2
FServiceone%2Fimageloc%2F1.jpg
    </image_location>
  </image>
<audio_id>1</audio_id>
  <audio>
    <audio_id>1</audio_id>
    <name>14935__incarnadine__echo001.mp3</name>
    <size>17668</size>
    <type>.mp3</type>
    <audio_location>
http%3A%2F%2Flocalhost%3A8090%2
FServiceone%2Faudioloc%2F14935__incarnadine__echo001.mp3
    </audio_location>
  </audio>
</Film>
```

Listing: XML Message format

The following listing shows the structure of Film Java class which maps the XML elements to the objects and these variables can be used to display the content. The annotations used in the listing provide guidance to the structuring of the message.

```
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlElement;
@XmlRootElement(name = "film")
public class Film {
        private int film_id;
        private String title;
        private String description;
        private int length;
        private int image_id;
        private Image image;
        private int audio_id;
        private Audio audio;
        @XMLElement(name = "image")
        public Image getImage() {
                return image;
        }
        @XMLElement(name = " audio ")
        public Image getAudio() {
                return audio;
        }
```

Listing: POJO for Film class

If the message format is JSON, following library is used to convert the message to Java objects.

Gson: [15]

Gson is a Java library that can be used to convert Java Objects into their JSON representation. It can be used to convert a JSON string to an equivalent Java object.

Goals of Gson

- Provide simple toJson() and fromJson() methods to convert Java objects to JSON and vice-versa
- Allow pre-existing non-modifiable objects to be converted to and from JSON
- Extensive support of Java Generics
- Allow custom representations for objects

- Support arbitrarily complex objects (with deep inheritance hierarchies and extensive use of generic types)

The message format should be processed and displayed to the end user (browser). The **FilmClass.java**, an object reference to **Film** is set as a variable. In 'Film.java' all the variables are declared with getter and setter methods. As you can see in the JSON message format, the image and audio location URL's are encoded.

Apache CXF produces application/JSON MIME type and the response to the web application is shown in the listing below.

JSON message format:


```
{"Film":{"film_id":1,"title":"ACADEMY DINOSAUR","description":"A
Epic Drama of a Feminist And a Mad Scientist who must Battle a Teacher in
The Canadian
Rockies","length":86,"image_id":1,"image":{"image_id":1,"name":"1.jpg","
size":408307,"type":".jpg","image_location":"http%3A%2F%2Flocalhost%3
A8090%2FServiceone%2Fimageloc%2F1.jpg"},"audio_id":1,"audio":{"audi
o_id":1,"name":"14935__incarnadine__echo001.mp3","size":17668,"type":".
mp3","audio_location":"http%3A%2F%2Flocalhost%3A8090%2Fserviceone
%2Faudioloc%2F14935__incarnadine__echo001.mp3"}}}
```


Listing: JSON message format

Binding and Processing:

Gson provides two important functionalities toJson ( ), fromJson ( ). fromJson ("JSON formatted message", POJO class). This method desterilizes the specified JSON into an object of the specified class.

```
Gson gson = new Gson();
FilmClass filmc = gson.fromJson(output, FilmClass.class);
```

Listing: Parsing using Gson.

3.3 Database Schema defined

In this section I would like to present the schema and tables used in the practical part of the thesis. The backend part is supported by MySQL which is an open source Relational Database Management System provided by Oracle. Also, this section provides information on different data types and support for data types in MySQL and Java programming language.

This thesis requires retrieval of different data types which can hold contents like image, audio and some metadata particular to the record stored in the tables of SAKILA schema. Spring provides JDBCTemplate class, which pulls the required information, from the backend schema 'SAKILA'.

A table in the next page gives detailed information on different data types supported by both MySQL and Java programming language. Ideally, it acts like a lookup table for a programmer, which is important for implicit or explicit conversions of data types in code development process.

MySQL type name	Return value of GetColumnClassname	Returned as Java Class
SMALLINT	SMALLINT	Java.lang.Integer
INT, INTEGER	INTEGER	Java.lang.Integer
DATETIME	DATETIME	Java.sql.Timestamp
VARCHAR	VARCHAR	Java.lang.String
TEXT	VARCHAR	Java.lang.String
MEDIUMBLOB	MEDIUMBLOB	Byte[ ]
LONGBLOB	LONGBLOB	Byte[ ]

TABLE 5: Data type support in MySQL and Java

SAKILA is an open source schema with real time data provided by MySQL. 'SAKILA' schema stores information on movies. There are a total of 17 tables provided by this schema. This thesis work requires 2 additional tables "IMAGES" and "AUDIO". These tables are populated from Wiki Commons, an open source website which handles a variety of information which is free to access, download and usage. Images and audio files when stored in database tables have data types as MEDUIMBLOB and LARGEBLOB. Out of the 19 tables we have chosen 3 tables for the development and testing of this thesis. Next page provides you with a snapshot of the 3 tables and the interdependency (often called as primary key and foreign key relationships) of the schema.

FILM

film_id	title	description	length	**image_id**	**audio_id**
Smallint	varchar	text	smallint	Int	Smallint

IMAGES

Id	name	size	type	Content	Location
Int	varchar	int	varchar	Longblob	Varchar

AUDIO

Idaudio	a_name	a_size	a_type	a_content	audio_location
smallint	varchar	Int	varchar	mediumblob	Varchar

Figure 5: Schema for SAKILA

In this chapter we looked at all the open source frameworks, tools, API's and plug-ins used in the development process. A detailed view about the backend schema and a snapshot of the three tables are provided.

CHAPTER 4

Analysis and Evaluation of Architectures

This chapter provides you with a detailed explanation on different test cases, especially on encoding and compression. Both the techniques are explained in detail. As explained in the abstract, this thesis is built on two scenarios. Section 4.1 explains Scenario 1 which has two distinct cases to be tested. With XML and JSON as message exchange formats, the size of the payload is compared after applying all the test cases. Conclusions on determining what message formats are to be used are discussed in detail. In Section 4.2, Scenario 2 architecture is built by considering all the test cases and pros of Scenario 1.

Image and audio files are encoded before transmitting the payload to the destination. Base 64 encoding is a technique which takes binary data and converts it into text which makes the transmission process easy via HTML forms or emails. Basically, it is a way of encoding arbitrary binary data to ASCII text. Compression is another technique which is applied on the content (payload) before the transmission of the message or the payload. These two techniques are widely used in transmitting messages from one system to another. These two techniques are applied on the payload. The sequences in which they are applied made a huge difference, which lead us to test two different test cases. They are:

1. Compression after encoding
2. Encoding after compression

Both the test cases are explained in detail, but first, let us take a look at the basis of these techniques. These two techniques are applied on both the message formats, namely, XML and JSON. Scenario 2 chose and followed the best sequence.

Compression

It is not necessary to compress the payload when transmitting the message to the destination. For better performance (i.e. to fasten up the process of transmitting the payload) we are compressing the message exchange formats, namely, XML and JSON. It is the developer's choice whether to compress the complete payload or certain elements of the payload like image and audio. Quality is a concern when compressing the image and audio files. For compressing image and audio files we have used GZIP, a widely used compression technique provided by GZIPOutputStream **[16]** class from java.util.zip **[17]** package.

The direct subclass for OutputStream is ByteArrayOutputStream which uses write (byte b [ ]) **[18]** method to write data into byte array. The buffer capacity is initially 32 bytes, though its size increases if necessary. The data can be retrieved using toByteArray( ) or toString( ). Below is a code snippet which explains step by step procedure to compress the payload.

```
// This method takes byte [ ] as a parameter and returns a compressed byte
[ ].
public byte[ ] compression(byte [ ] content_to_be_compressed  ) throws
IOException{
// Creates a new byte array output stream.
ByteArrayOutputStream baos = new ByteArrayOutputStream();

// Creates a new output stream with a default buffer size.

GZIPOutputStream gos = new GZIPOutputStream(baos);

// Writes content_to_be_compressed.length bytes from the specified byte
array to this output //stream.
gos.write(content_to_be_compressed );

// Flushes this output stream and forces any buffered output bytes to be
written out.
gos.flush();

// Closes this output stream and releases any system resources associated
with this stream.
gos.close();

// Creates a newly allocated byte array.
byte [ ] compressed_content= baos.toByteArray();
return compressed_content;
}
```

Snippet: Applying compression technique in Java using GZIP

Encoding

Encoding is the process of putting a sequence of characters into a specialized format that makes transmission of the message easy. Wikipedia says “Base64 is a group of similar encoding schemes that represent binary data in an ASCII string format by translating it into radix-64 representation.” **[19]**. Base64 encoding schemes are used to encode the binary data that needs to be transferred. When sending complex data in XML

or JSON payload, Base64 encoding schemes are used to encode the content and transmit the data which remain intact and cannot be changed.

Base64 is a class provided by Apache Commons which defines the functionality of converting byte [ ] data into Base64 encoded data. The service consumer builds his web application with the return data types provided by the resource. The resource returns image and audio as encoded and compressed byte [ ]. The consumer reads it as byte [] and decodes it. **org.apache.commons.codec.binary.Base64** provides functionality for converting an encoded Base64 byte [ ] into a String or byte array Input Stream. The snippet below explains how to encode the byte array by using Base64 in Java.

```
// This method takes byte [ ] as a parameter and returns an encoded byte [ ].
public byte[ ] encoding( byte[ ] content_to_be_encoded ){
// encodeBase64 (byte [ ]) takes byte [ ] as a parameter and returns an encoded byte [ ].
byte [ ] encoded_content=Base64.encodeBase64( content_to_be_encoded) ;
return encoded_content;
}
```

Snippet: Base64 encoding in Java

NOTE: Throughout the document, **Size** is represented in **bytes** and **Time** is represented in **ms** (milliseconds). Time in **ms** is calculated by taking the average of first four requests – response communication. The image is basically a JPG file and the audio is a ".wmv" formatted file.

Payload carries metadata about film, image related to film, an audio file related to the film and metadata related to the image and audio file.

Compression after encoding

Image, audio are stored as MEDIUMBLOB in MySQL and retrieved as bytes [] (byte array) in action class. The retrieved content is encoded with Base64 and then compressed using GZIP. The content is then mapped to the XML element or JSON {name, value} pair. Payload is then sent to the requested destination. The web application expands the content and then decodes it in order to display the image and the audio file in HTML5. The length (bytes) of the payload in JSON and XML are noted and the time (ms) taken to send the response packet to the client (web application) is recorded and compared.

Content / Operation	Original size (bytes)	After encoding (bytes)	After compression (bytes)
Image 1	408,307	544,412	399,939
Audio 1	17,668	23,560	13,370
Image 2	616,937	822,584	514,464
Audio 2	17,669	23,560	14,792
Image 3	298,951	398,604	297,140
Audio 3	17,668	23,560	14,529

Table 6: Results after encoding and compression of image, audio files

```
getFilm() method called from Service class!
IMAGE
Length of Image file from Database: 408307
Length of Image file after encoding with base64: 544412
Length of encoded image after compression GZIP: 399939
********************
checking on compression!
********************
Compressed successfully
double check! 399939
AUDIO
Length of Audio file from Database: 17668
Length of Audio file after encoding with base64: 23560
Length of encoded audio after compression GZIP: 13370
********************
checking on compression!
********************
Compressed successfully
double check! 13370
```

Figure 6: Output after encoding and compression

Encoding after compression

In this test, image and audio are first compressed and then encoded with Base64. The content is then mapped to the XML elements or JSON name, value pairs. Expanding of the content is followed by decoding of the content. The process is explained in detail in the CODE APPENDIX. Following table shows the recorded test results.

Content / Operation	Original size (bytes)	After compression (bytes)	After encoding (bytes)
Image 1	408,307	387,481	516,644
Audio 1	17,668	13,255	17,676
Image 2	616,937	522,686	696,916
Audio 2	17,669	14,641	19,524
Image 3	298,951	294,945	393,260
Audio 3	17,668	14,408	19,212

Table 7: Results after compression and encoding of image, audio files

```
getFilm() method called from Service class!
IMAGE
Size of image from database is: 408307
After applying compression size of image is 387481
********************
checking on compression
********************
Compressed successfully
encoding!
Length of Image file after comprressing and encoding with base64: 516644
AUDIO
Size of audio from database is: 17668
After applying compression size of audio is 13255
********************
checking on compression
********************
Compressed successfully
encoding!
Length of Audio file after comprressing and encoding with base64: 17676
double check! 17676
```

Figure 7: Output after compression and encoding

Observations on "Compression after encoding" test case

1. After Base64 encoding, size of the content increased by 33% i.e., it is expanding 3 bytes into 4 characters which results in an increase in the size of the content.
2. The difference between the original size and (Base64 + compressed) content is 100000 bytes on an average for images.

Observations on "Encoding after compression" test case

1. It depends on what kind of data the payload is carrying. Encoding and compressing works great with textual / binary data types. Applying compression to image and audio files did not really do much.
2. Encoding of content after compression is a bad idea. As you can see from the above result table, size of the content to be transferred drastically increased.

3. Previous test case i.e., compression after encoding has better results compared to this case.

So, in further scenarios of this thesis work “Compression after encoding” sequence is preferred in developing applications or services.

4.1 Scenario 1

Scenario 1 is about developing a REST service for a web application developed in Spring MVC. The REST service is developed using Apache CXF. In Scenario 1 we have two independent cases which are later used in building Scenario 2 of this thesis. The web application is a simple service which provides information about films. The information includes all the metadata about the movie, images in the movie and a sample audio file related to the movie. The metadata includes the film director’s name and the cast of the movie. This service will not necessarily be used by an end client, but can be by an application which serves the end client. The payload consists of a mixture of data which is transmitted to the destination.

In Scenario 1, we have all the metadata stored in the database which is considered to be a case explained in Section 4.1.1, and in Section 4.1.2, all the metadata about the film is stored in the database and the associated image and audio files are stored in the file system. The architecture associated with the file system is explained in Section 4.1.2 in detail. Both Case 1 and Case 2 are REST Service providers developed using Apache CXF. CXF has in built annotations which expose the functionality of the application via a service.

This section also tests message exchange formats like XML and JSON. These message exchange formats are compared by the size of the content being transmitted by them. As proven by the initial test cases that compression after encoding technique should be applied for transmission of messages, now, this technique is applied specific to the message exchange formats. This technique is best suited when transferring image, audio and video files.

4.1.1 Case (I): Database as backend

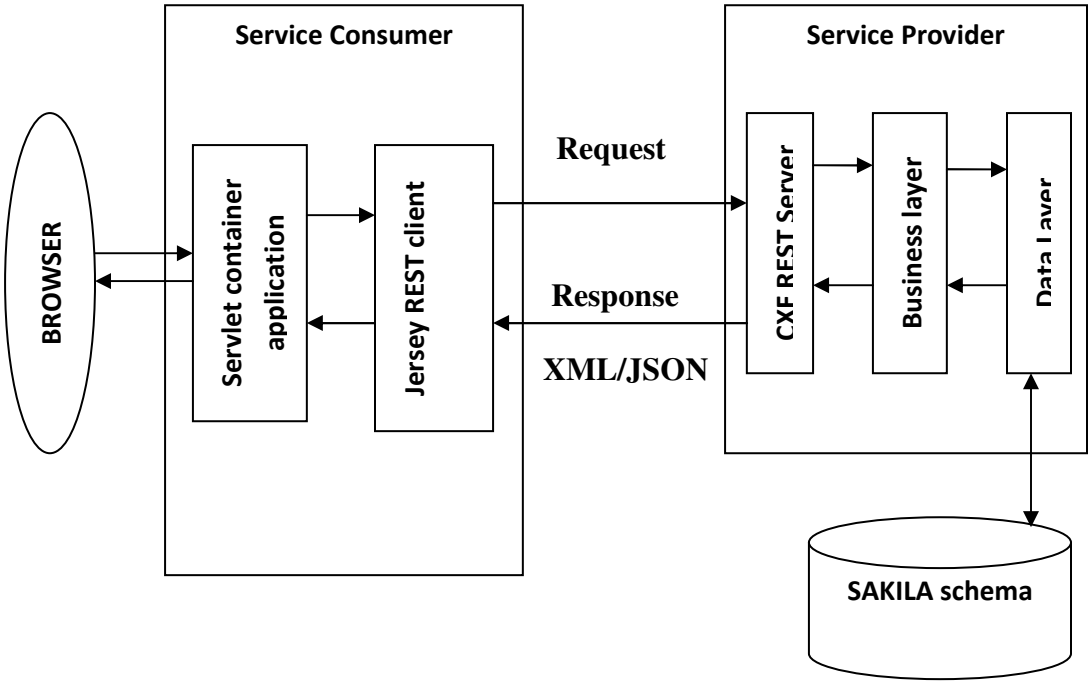

Figure 8: Case 1, Architecture with database as backend

We have developed a Spring MVC application with a database as backend. The spring MVC application is exposed as a service by Apache CXF. Image and audio are stored in the database as MEDIUMBLOB, a data type for storing byte [ ] content in MySQL. The database is an open source schema provided by MySQL. When this service

is requested, CXF handles the request to the appropriate spring action class and the database is queried. The result is mapped to POJO class after Base64 encoding and compressing the image and audio file.

The response message includes Meta content about film, an image file and Meta content about image, an audio file and Meta content about the audio. The table below shows the payload (encoded and compressed) size and response time of the content retrieved from database.

Content / Performance	XML payload	Response time (ms)	JSON payload	Response time (ms)
Response 1	551,601	339	565,189	368
Response 2	706,197	424	717,889	430
Response 3	416,098	238	428,142	257

Table 8: Payload size and response time for content retrieved from database

Observations specific to XML and JSON message formats:

1. The size difference of the XML payload and JSON payload is at least 10000 bytes. XML wins. Perhaps, it is not a good idea to send Base64 encoded content in JSON.
2. JSON format can be processed easily by JavaScript and Ajax, so JSON can be chosen for further development process. Data type is a major concern.
3. Compression works fine on textual data, but in case of image and audio it did not turn out well.

When a web service is invoked by a RESTClient, the response is generated via XML or JSON based on the content/type negotiation.

Payload is represented in bytes.

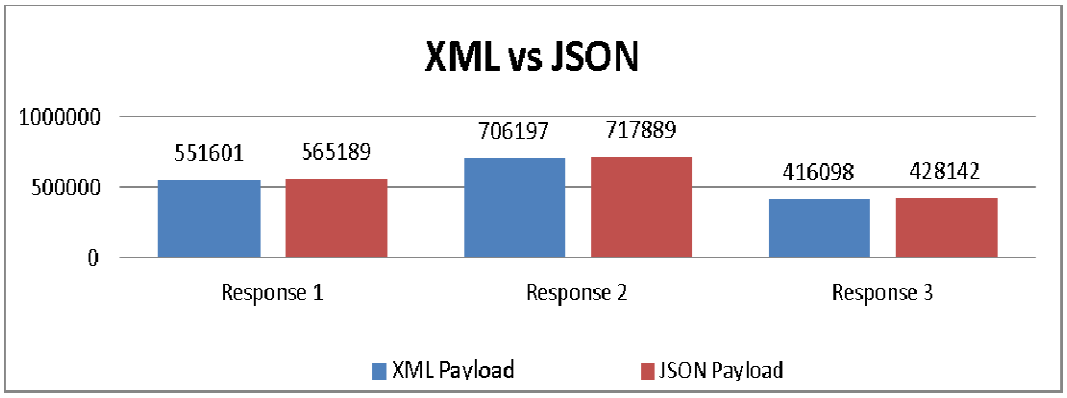

Figure 9: Comparison of XML and JSON after encoding and compression

Response time in Milli Seconds (ms)

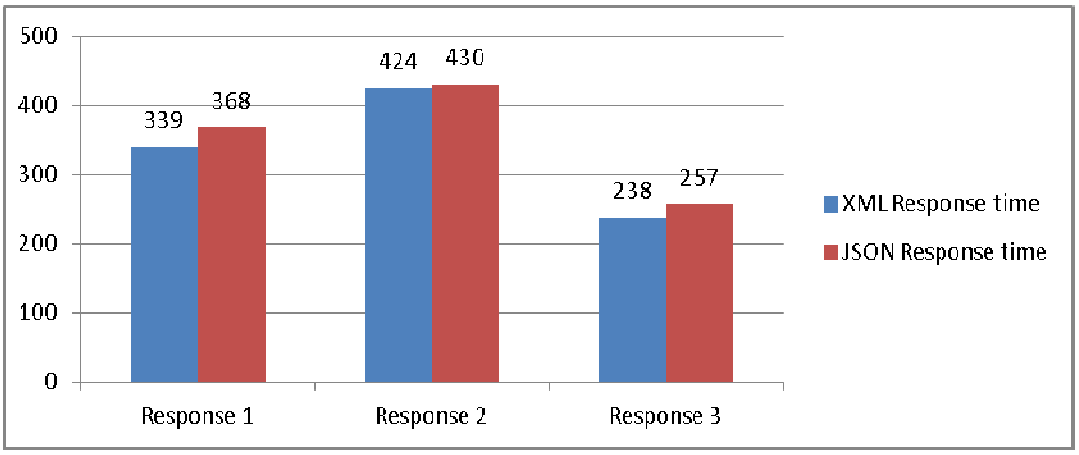

Figure 10: Response time for XML and JSON after encoding and compression

Conclusion on Case (I): Database as backend

1. Image and audio files should be encoded in order to send the packet to the destination.
2. If it's a small application, it's not a bad idea to store avatar images in database.
3. As the payload in Scenario 1 is carrying audio and image (non-avatar) files, compressing after encoding works fine.
4. JSON can be easily interpreted with the combination of JavaScript and Ajax by browsers, so for the next scenario JSON is the message format used for transferring data.
5. Sending the images with CSS makes much more sense because CSS can be cached by the browser.

4.1.2 Case (II): File system as backend

In Case 2, image and audio files are stored in the file system. The location of the image and audio is in the Sakila database. In fact, there can be a dedicated server which stores image and audio files. So now the payload carries only the location of the image and audio files. The size of response content is reduced a lot.

Experimental results show that storing image and audio files in the file system has a better performance when compared to a database. The advantages of storing the images and audio files in the file system rather than in a database are discussed later in this section.

Architecture for Case2:

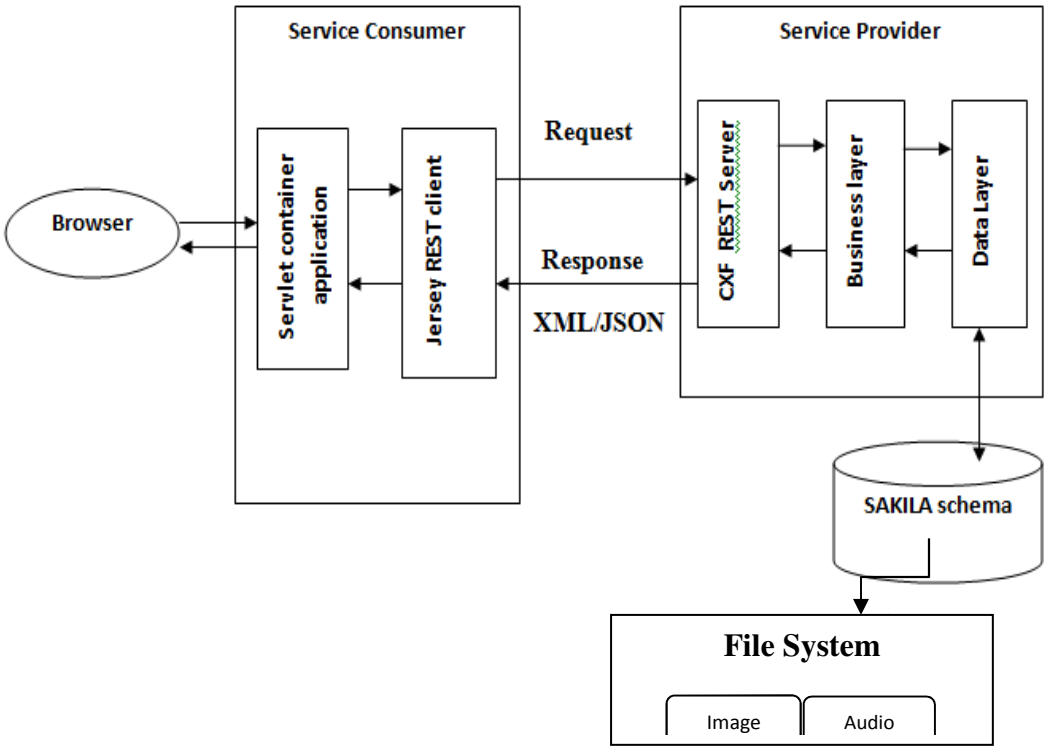

Figure 11: Storing image and audio files in file system

In this case, Base64 encoding technique is not used as there is no byte [ ] data being transferred. Instead we are transferring a URL, the location of the image and audio file. The URL needs to be encoded to avoid spaces that are transferred via payload. The application which receives the payload decodes the URL using URLDecoder class provided by Java. A snippet below shows how to encode and decode URL.

Encoding of URL

```
try {
        String URL = "http://localhost:8090/Serviceone/imageloc/";
// URLEncoder is a class using encode () which takes a String and the type of
//encoding technique it is using.
        String encodedurl = URLEncoder.encode(URL, "UTF-8");
//the name of the image is located in the database.
        String encodedurl_dbloc = encodedurl +
rs.getString("image_location");
        images.setImage_location(encodedurl_dbloc);
        } catch (UnsupportedEncodingException ue) {
                ue.printStackTrace();
        }
```

Snippet: URLEncoder functionality in Java

Decoding of URL

This s performed on the client side.

```
String decoded_url_image = URLDecoder.decode(filmc.getFilm()
                                        .getImages().getImage_location(),

"UTF-8");
```

Snippet: URLDecoder functionality in Java

XML is definitely more verbose than JSON. The following practical results show how XML and JSON differ in size and response time. The response time is tested on RESTClient, a plug in provided by the Firefox browser. The following snippets show the XML payload and JSON payload that are transferred to the web application.

XML Payload

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Film>
<film_id>4</film_id>
<title>AFFAIR PREJUDICE</title>
<description>A Fanciful Documentary of a Frisbee And a Lumberjack who
must Chase a Monkey in A Shark Tank</description>
<length>117</length>
<image_id>4</image_id>
<image>
   <image_id>4</image_id>
   <name>4.jpg</name>
   <size>563688</size>
   <type>.jpg</type>
<image_location>http%3A%2F%2Flocalhost%3A8090%2FServiceone%2Fim
ageloc%2F4.jpg</image_location>
</image>
<audio_id>4</audio_id>
<audio>
   <audio_id>4</audio_id>
   <name>14938__incarnadine__echo004.mp3</name>
   <size>17668</size>
   <type>.mp3</type>
<audio_location>http%3A%2F%2Flocalhost%3A8090%2FServiceone%2Faud
ioloc%2F14938__incarnadine__echo004.mp3</audio_location>
</audio>
</Film>
```

JSON Payload

```
{"Film":{"film_id":4,"title":"AFFAIR PREJUDICE","description":"A Fanciful
Documentary of a Frisbee And a Lumberjack who must Chase a Monkey in A
Shark
Tank","length":117,"image_id":4,"image":{"image_id":4,"name":"4.jpg","size"
:563688,"type":".jpg","image_location":"http%3A%2F%2Flocalhost%3A8090
%2FServiceone%2Fimageloc%2F4.jpg"},"audio_id":4,"audio":{"audio_id":4,"
name":"14938__incarnadine__echo004.mp3","size":17668,"type":".mp3","audi
o_location":"http%3A%2F%2Flocalhost%3A8090%2FServiceone%2Faudioloc
%2F14938__incarnadine__echo004.mp3"}}}
```

Snippet: XML and JSON payload carrying image and audio files URL

The following tables show the performance of JSON over XML.

	XML Payload	Response time for XML	JSON payload	Response time for JSON
Payload 1	726	318	545	170
Payload 2	728	145	547	151
Payload 3	726	234	545	157
Payload 4	723	156	542	147

Table 9: Record of XML and JSON payload size and time in (ms)

In the previous case, when transferring image and audio files in bytes via XML and JSON payloads, we have clearly proved that XML has a better performance than JSON in transferring Base64 encoded data. In Case 2, we have practically proved that JSON has a better performance compared to XML. So, data type does matter in transferring content.

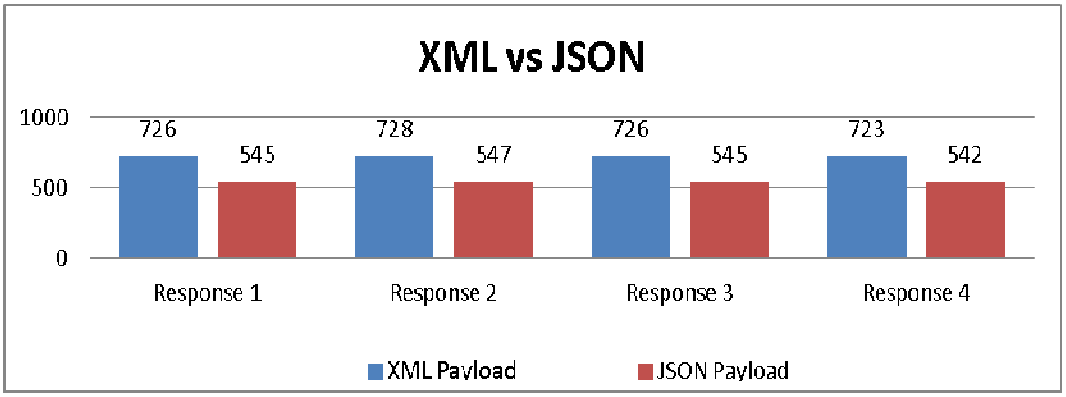

Figure 12: Comparison of XML and JSON with respect to response content size

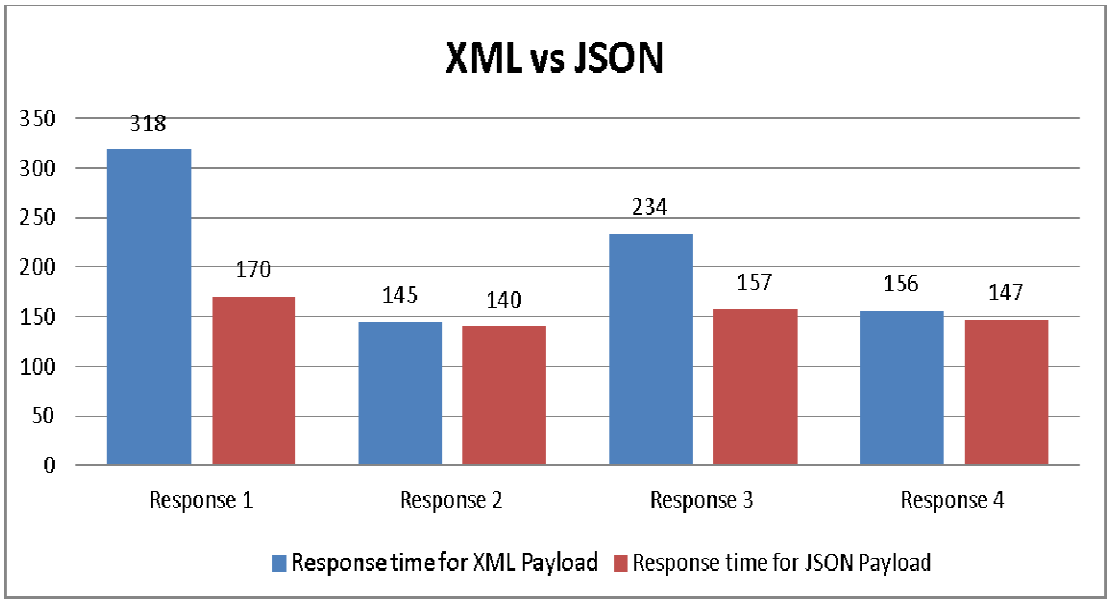

Figure 13: Comparison of XML and JSON in Response time when image and audio are stored in file system.

Conclusion on Case (II): File system as backend

Though there are many advantages of using a database for storing images and audio files, it has considerable disadvantages too. For a small application, storing images and audio files inside a database is definitely suggestible. For a large scale application where millions of images are retrieved, it's a bad idea to store images or audio type content in the database. Conclusions which can be derived by looking at the experimental values in Case 2 are

1. JSON message format serves better when performance and speed are considered to be factors.
2. Placing image and audio files in the file system is a good option to be considered when dealing with large image or hyper media content.
3. Indexing of data wins when compared to database and file system.

4.2 Scenario 2: Just-in-time Architecture

The data and multimedia content are served as services to the clients. From the previous architectures, we have shown that storing multimedia content in a file system is a good idea. Storage mediums like file systems and databases should be chosen based on the size of the application.

As we are talking about real time applications and big services, the storage medium should be a file system. We have also proven that message exchange format JSON has its advantages over XML. Based on the pros of Scenario 1, Scenario 2 uses file system as storage medium and JSON as message exchange format.

Just-in-time Architecture:

The implementation of every service is independent. Sakila database stores the information of audio service. So, from the first response payload, the appropriate link of the audio file is provided to the AJAX call. The AJAX mechanism then makes a call to the service provider 2. Service provider 2 can be utilized by other clients if needed. That is the base idea behind just-in-time architecture.

The central idea of AJAX is to communicate with the server asynchronously without a page refresh. An XMLHTTPRequest object makes a request to the Service provider allowing GET method to the server. This does not show any status message to the user.

In this architecture, multimedia content and text data are provided as just-in-time services. Initially all the text data related to the request is sent as a response. The response also includes URI's of different services which host audio content. So, when the end user

(browser) clicks on the audio link, the service will be invoked via an AJAX call and JSON payload is parsed using JQERY.

We have noticed that the performance drastically increased because the response for the first call to REST service carries a payload of text and an encoded URL of the image resource and a service URI for audio resource. One big challenge we faced was integrating the right data to the call. It would be of no use if the right data is not presented to the end client.

In suggestions for future work, this factor should be considered.

Architecture for Scenario 2

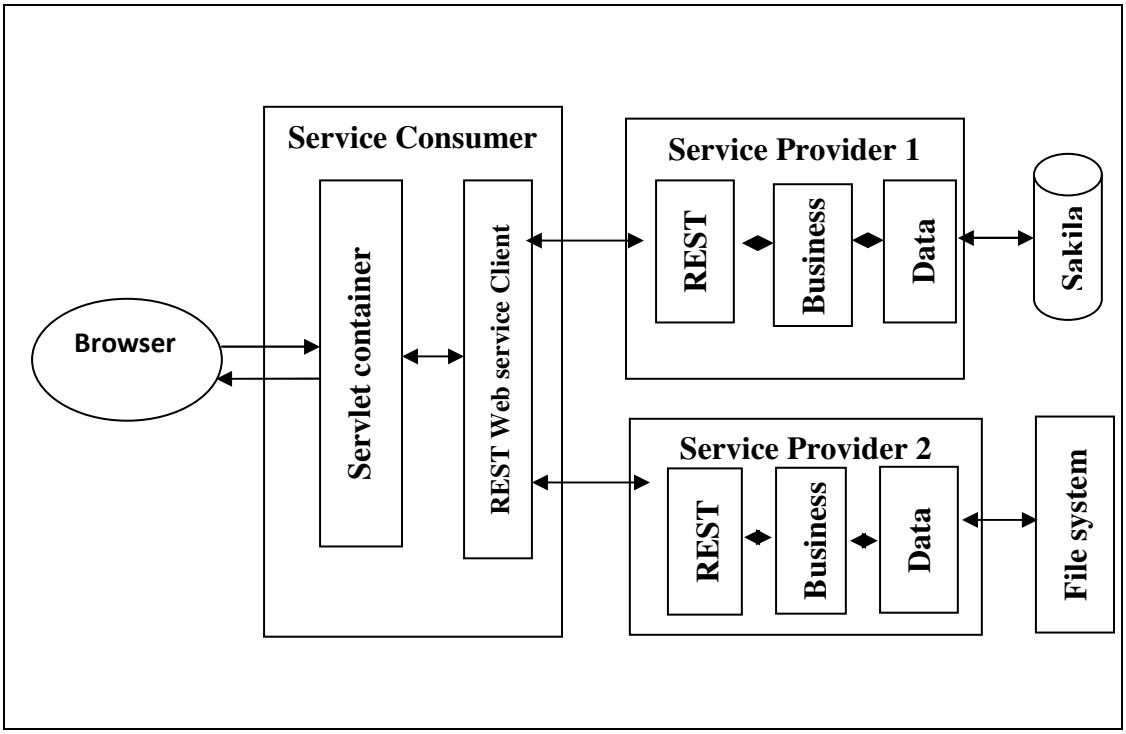

Figure 14: Just in time architecture

Same Origin Policy

"In computing, the same origin policy is an important security concept for a number of browser-side programming languages, such as JavaScript. The policy permits scripts running on pages originating from the same site to access each other's methods and properties with no specific restrictions, but prevents access to most methods and properties across pages on different sites **[20]**."

JSONP

JSON with padding is a complement to JSON. It provides functionality for requesting data from a service hosted a different domain. This is prohibited by many browsers because of the concept of "same origin policy".

AJAX call to Web Service

When a call is made to Service provider 2, usually with JSON the date returned is {audio_location: http://localhost:8080/audio_files/name_of_audio.mp3}.

But with JSONP callback

({audio_location: http://localhost:8080/audio_files/name_of_audio.mp3}).

Below table provides with size of the payload and the response time

	JSON payload	**Response time for JSON**
Response 1	540	160
Response 2	542	141
Response 3	540	147
Response 4	537	137

Table 10: Message payload and response time

The below listing shows how an AJAX call was made to service provider 2.

```
<html>
<head>
   <script type="text/javascript" src="jquery-1.6.2.min.js"></script>
</head>

<body>

<button id="ajax_call">audio</button>
<button id="json_msg">json_message</button>

<script type="text/javascript">
   $('#json_msg').click(function(){
      alert('json');
       $.getJSON("http://localhost:8080/filmservices/film/get?callback=?",
       function(data) {
          alert(data);
        });
   });

   $('#ajax_call').click(function(){
      alert('ajax');
       $.ajax({
          type: "GET",
          dataType: "json",
          url: "http://localhost:8080/ filmservices/film/get",
          success: function(data){
             alert(data);
          }
       });
   });

</script>
</body>
</html>
```

Listing: AJAX call made with Jquery to display the result to the client

Our concentration throughout this thesis is how different architectures will react and support different data type behavior. As no service gives direct access to the end user (browser), a client here is not an end user; it's a web application (REST client). Whether it can be numerous clients or a single client, the response message from the service depends on how an individual client parses the response (XML or JSON) and displays it to the end user (a browser).

CHAPTER 5

Conclusions

This chapter concludes the best design principles and the standard architecture suitable for large scale business environments. Sections 5.1 and 5.2 explain the objectives and performance related results of the tested sequences. A recommended architecture is explained in detail, and why we have chosen it to be the recommended architecture is explained. Section 5.3 proposes some future options for development or extension of this thesis work.

5.1 Conclusion on Scenario 1:

Storing multimedia in database:

When storing multimedia in databases, caching can be applied on the client side. This improves the performance. Usually storing multimedia content in a database is not suggested for large scale applications. When sending the encoded content via XML or JSON, XML had better performance over JSON.

Lesson learned: The data type is vital and should be considered in transferring payload.

Storing multimedia in file system:

Storing multimedia in a file system has its own advantages. This type of storage is preferred when developing large scale applications or services. To the matter of fact, a dedicated central file system is suggested to host the multimedia content. Indexing and linking the multimedia content to the request is a challenging task.

Future work: Different mechanisms could be developed for hiding the resource URL of the multimedia content to the service

5.2 Conclusion on Scenario 2:

As our concentration on large scale applications and services, storing the hypermedia in the file system is suggested in this model. According to the investigations of this thesis, this architecture is best proposed and is recommended by us. The request for an audio or video file should go by an AJAX call with HTML5 support. A dynamic call can be made to the service with a request. The requested resource stores hypermedia in the file system, or it could be a dedicated server which stores multimedia content. The tough part we encountered was the indexing of data in the file system and linking it with the right content in the database. As all the text data is stored in the database. The hypermedia content which comes as a response via a service should be related to the text data. Linking location paths of files and service URLs with the right content of text data was a tedious task.

Maintainability need not be an issue because we are not providing DELETE, PUT and POST operations via a service.

5.3 Future Work

In the field of computer science there is no component which has a perfect solution.

As stated, the REST architecture is stateless, and caching techniques can be applied on the client side for better performance [21]. Cashing techniques could be applied to Scenario 1, which may improve the performance and response time.

ODATA:

"The **Open Data Protocol (OData)** is a Web protocol for querying and updating data that provides a way to unlock your data and free it from silos that exist in applications today. OData is being used to expose and access information from a variety of sources including, but not limited to, relational databases, file systems, content management systems and traditional Web sites.

OData is consistent with the way the Web works - it makes a deep commitment to URIs for resource identification and commits to an HTTP-based, uniform interface for interacting with those resources (just like the Web). This commitment to core Web principles allows OData to enable a new level of data integration and interoperability across a broad range of clients, servers, services, and tools **[22]**."

With ODATA, a new architecture can be suggested, which would act as a replacement to REST.

CODE APPENDIX

Since we have two applications in this thesis work, service consumer application and service provider application, we had to build two applications to test the scenarios defined in Chapter 4.

Service provider application

FilmService.java

```
package scenarioone.usecaseone.services;
import java.util.List;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.WebApplicationException;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.ResponseBuilder;
import javax.ws.rs.core.Response.Status;
import scenarioone.usecaseone.daoimpl.FilmDaoImpl;
import scenarioone.usecaseone.entities.Film;

@Path("/filmservices")
@Produces({ "application/JSON" })
public class FilmService {

	FilmDaoImpl filmdaoimpl;

	public FilmDaoImpl getFilmDaoImpl() {
		return filmdaoimpl;
	}

	public void setFilmDaoImpl(FilmDaoImpl filmdaoimpl) {
		this.filmdaoimpl = filmdaoimpl;

	}

	@GET
	@Path("/film/{film_id}")
	public Film getFilmbyID(@PathParam("film_id") int film_id) {
		System.out.println("getFilm() method called from Service
class!");
```

```
                Film film = (Film) getFilmDaoImpl().getFilm(film_id);
                if (film == null) {
                        ResponseBuilder builder =
Response.status(Status.BAD_REQUEST);
                        builder.type("application/xml");
                        builder.entity("<error>Category Not Found</error>");
                        throw new WebApplicationException(builder.build());
                } else {
                        return film;
                }
        }

        @SuppressWarnings("unused")
        @GET
        @Path("/films")
        public List<Film> getFilms() {
                System.out.println("getFilms() is called");
                List<Film> film = (List<Film>)
getFilmDaoImpl().getAllFilms();
                System.out.println("There are a total of " + film.size()
                                + " of movies in the database");
                if (film == null) {
                        ResponseBuilder builder =
Response.status(Status.BAD_REQUEST);
                        builder.type("application/JSON");
                        builder.entity("<error>Category Not Found</error>");
                        throw new WebApplicationException(builder.build());
                } else {
                        return film;
                }
        }

}
```

FilmDao.java

An interface for handling web service request.

```
package scenarioone.usecaseone.dao;
import java.util.List;
import scenarioone.usecaseone.entities.Film;

public interface FilmDao {
	// returns film by id and all the particulars about the film!
	public Film getFilm(int id);
	// returns all films in the database
	public List<Film> getAllFilms();

}
```

FilmDaoImpl.java

The implementation class of the data access object

```
package scenarioone.usecaseone.daoimpl;
import java.util.ArrayList;
import java.util.List;
import org.springframework.jdbc.core.JdbcTemplate;
import scenarioone.usecaseone.dao.FilmDao;
import scenarioone.usecaseone.entities.Film;
import scenarioone.usecaseone.mapper.AllFilmMapper;
import scenarioone.usecaseone.mapper.FilmMapper;

public class FilmDaoImpl implements FilmDao {

	private JdbcTemplate jdbctemplate;

	public void setJdbcTemplate(JdbcTemplate jdbctemplate) {
		this.jdbctemplate = jdbctemplate;
	}

	// called when querying about individual films.
	@SuppressWarnings("unchecked")
	@Override
	public Film getFilm(int film_id) {

		String sql = "SELECT f.film_id
,f.title,f.description,f.length,f.image_id,f.audio_id,i.id,i.name,i.size,i.type,i.i
mage_location,a.idaudio,a.a_name,a.a_size,a.a_type,a.audio_location from
film as f, images as i ,audio as a where f.image_id=i.id and
f.audio_id=a.idaudio  and f.film_id=?;";
		final Object[] params = new Object[] { film_id };
		List<Film> film = jdbctemplate.query(sql, params, new
FilmMapper());
		return film.get(0);

	}

	// called for retrieving all the films

	@Override
	public List<Film> getAllFilms() {
		String sql = "SELECT film_id,
title,description,length,image_id,audio_id FROM sakila.film;";
		List<Film> films = new ArrayList<Film>();
```

```
        films = jdbctemplate.query(sql, new AllFilmMapper());
        return films;  }}
```

FilmMapper.java

Connects with the database and retrieves the values and links with the setter methods of POJO classes.

```
package scenarioone.usecaseone.mapper;

import scenarioone.usecaseone.GZIP.base64.AudioCompressionEncode;
import scenarioone.usecaseone.GZIP.base64.ImageCompressionEncode;
import scenarioone.usecaseone.base64.GZIP.AudioEncodeConversion;
import scenarioone.usecaseone.base64.GZIP.ImageEncodeConversion;
import scenarioone.usecaseone.base64.encoding.JustEncoding;
import scenarioone.usecaseone.entities.Audio;
import scenarioone.usecaseone.entities.Film;
import scenarioone.usecaseone.entities.Images;
import java.io.UnsupportedEncodingException;
import java.net.URLEncoder;

//import scenarioone.usecaseone.entities.Video;

import java.io.IOException;
import java.net.URLEncoder;
import java.sql.ResultSet;
import java.sql.SQLException;
import org.springframework.jdbc.core.RowMapper;

@SuppressWarnings("rawtypes")
public class FilmMapper implements RowMapper {

        @Override
        public Film mapRow(ResultSet rs, int row_no) throws
SQLException {

                Film film = new Film();
                Images images = new Images();
                Audio audio = new Audio();
                // Video video=new Video();
                JustEncoding je_base64 = new JustEncoding();

                // retrieving film details from Film table
                film.setFilm_id(rs.getInt("film_id"));
                film.setTitle(rs.getString("title"));
```

```
                    film.setDescription(rs.getString("description"));
                    film.setLength(rs.getInt("length"));
                    film.setImage_id(rs.getInt("image_id"));
                    film.setAudio_id(rs.getInt("audio_id"));

                    if (rs.getInt("image_id") == rs.getInt("id")) {
                              // retrieving Image details from Images table
                              System.out.println("IMAGE");
                              images.setImage_id(rs.getInt("id"));
                              images.setName(rs.getString("name"));
                              images.setSize(rs.getInt("size"));
                              images.setType(rs.getString("type"));
//                            if (rs.getBytes("content").length == 0)
//                                      System.out
//                                                                .println("there is nothing in th
content ! check your IMAGE TABLE in sakila schema");

                              // plain bytes

                              // else {
                              // images.setContent(rs.getBytes("content"));

                              // first compressing and then base64

                              // ImageCompressionEncode ice = new
ImageCompressionEncode();
                              //
images.setContent(ice.compression(rs.getBytes("content")));

                              // first base64 and then compressing

                              // ImageEncodeConversion iec = new
ImageEncodeConversion();
                              //
images.setContent(iec.encoding(rs.getBytes("content")));

                              // just base64 encoding

                              //
images.setContent(je_base64.encoding(rs.getBytes("content")));

                              // }// content else close
                              try {
                                        String URL =
"http://localhost:8090/Serviceone/imageloc/";
                                        String encodedurl =
```

```
URLEncoder.encode(URL, "UTF-8");
                                String encodedurl_dbloc = encodedurl
                                                                +
rs.getString("image_location");
                                images.setImage_location(encodedurl_dbloc);
                        } catch (UnsupportedEncodingException ue) {
                                ue.printStackTrace();
                        }

                }// if end image

                else
                        System.out.println("Check SQL! image id are not
equal!");

                // retrieving Audio details from Audio table
                if (rs.getInt("audio_id") == rs.getInt("idaudio")) {

                        System.out.println("AUDIO");
                        audio.setAudio_id(rs.getInt("idaudio"));
                        if (rs.getString("a_name").isEmpty())
                                System.out.println("Field empty in
database");
                        else
                                audio.setName(rs.getString("a_name"));
                        audio.setSize(rs.getInt("a_size"));
                        audio.setType(rs.getString("a_type"));
                        try {
                                String URL =
"http://localhost:8090/Serviceone/audioloc/";
                                String audioencodedurl =
URLEncoder.encode(URL, "UTF-8");
                                String audioencodedurl_dbloc =
audioencodedurl
                                                                +
rs.getString("audio_location");

        audio.setAudio_location(audioencodedurl_dbloc);
                        } catch (UnsupportedEncodingException ue) {
                                ue.printStackTrace();
                        }
//                       if (rs.getBytes("a_content").length == 0)
//                       System.out
//                       .println("there is nothing in th content ! check your
AUDIO TABLE in sakila schema");
                        // else {
```

```
                        // audio.setContent(rs.getBytes("a_content"));

                        // first compressing and then base64

                        // AudioCompressionEncode ace = new
AudioCompressionEncode();
                        //
audio.setContent(ace.compression(rs.getBytes("a_content")));

                        // first base64 and then compressing

                        // AudioEncodeConversion aec = new
AudioEncodeConversion();
                        //
audio.setContent(aec.encoding(rs.getBytes("a_content")));

                        // just encoding of audio

                        //
audio.setContent(je_base64.encoding(rs.getBytes("a_content")));
                        // }// end of content else
                }// audio if end
                else
                        System.out.println("Check SQL! audio id are not
equal!");

                // initializes all the properties in Images class
                film.setImages(images);

                // initializes all the properties in Audio class
                film.setAudio(audio);

                return film;
        }
}
```

Encoding and Compression:

Program for base 64 encoding and GZIP compression.

AudioEncodedConversion.java

```
package scenarioone.usecaseone.base64.GZIP;

/*Author Koushik Maddipudi
 */
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.util.zip.GZIPInputStream;
import java.util.zip.GZIPOutputStream;
import org.apache.commons.codec.binary.Base64;

public class AudioEncodeConversion {

	private static byte[] compressed_audio = null;
	private static byte[] base64_and_compressed_audio = null;

	// method for encoding the content, takes a byte retrieved from the
database
	// and returns a compressed byte [ ]

	public byte[] encoding(byte[] audio_from_db) {
		System.out.println("Length of Audio file from Database: "
				+ audio_from_db.length);

		// returns an encoded byte [ ] and stores it in the variable
declared

		byte[] base64_encoded_audio =
Base64.encodeBase64(audio_from_db);
		System.out.println("Length of Audio file after encoding
with base64: "
				+ base64_encoded_audio.length);

		// checks if the content is encoded, it yes then compresses
it.

		if (Base64.isBase64(base64_encoded_audio)) {

			// method for compressing byte[ ]
```

```
                        compressionOfAudio(base64_encoded_audio);
                        base64_and_compressed_audio =
compressed_audio;
                } else
                        System.out.println("U have an error in encoding
image");
                System.out.println("double check! "
                                + base64_and_compressed_audio.length);
                return base64_and_compressed_audio;
        }

        // method for compression, takes an encoded byte[] and returns a
compressed
        // byte[]

        private byte[] compressionOfAudio(byte[]
base64_encoded_audio) {

                // Creates a new byte array output stream.
                ByteArrayOutputStream baos = new
ByteArrayOutputStream();
                try {
                        // Creates a new output stream with a default buffer
size.
                        GZIPOutputStream gos = new
GZIPOutputStream(baos);

                        // Writes content_to_be_compressed.length bytes
from the specified
                        // byte array to this output //stream.
                        gos.write(base64_encoded_audio);

                        // Flushes this output stream and forces any
buffered output bytes
                        // to be written out.
                        gos.flush();

                        // Closes this output stream and releases any system
resources
                        // associated with this stream.
                        gos.close();

                        // Creates a newly allocated byte array.
                        compressed_audio = baos.toByteArray();
                        System.out
                                        .println("Length of encoded audio
```

```
after compression GZIP: "
                                                        +
compressed_audio.length);
                        System.out.println("********************");
                        System.out.println("checking on compression!");

                        // checking for compression.
                        isCompressed(compressed_audio);
                } catch (IOException e) {
                        System.out.println("Error in compression");
                }

                return compressed_audio;
        }

        public boolean isCompressed(byte[] compressed_data) throws
IOException {
                if ((compressed_data == null) || (compressed_data.length <
2)) {
                        return false;
                } else {
                        System.out.println("********************");
                        System.out.println("Compressed successfully");
                        return ((compressed_data[0] == (byte)
(GZIPInputStream.GZIP_MAGIC)) &&
                                        (compressed_data[1] == (byte)
(GZIPInputStream.GZIP_MAGIC >> 8)));
                }
        }
}
```

ImageEncodedConversion.java

Methodology for Encoding the image files.

```
package scenarioone.usecaseone.base64.GZIP;

import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.util.zip.GZIPInputStream;
import java.util.zip.GZIPOutputStream;

import org.apache.commons.codec.binary.Base64;

public class ImageEncodeConversion {
	private static byte[] compressed_image = null;
	private static byte[] base64_and_compressed_image = null;

	public byte[] encoding(byte[] image_from_db) {
		System.out.println("Length of Image file from Database: "
				+ image_from_db.length);
		byte[] base64_encoded_image =
Base64.encodeBase64(image_from_db);
		System.out.println("Length of Image file after encoding with
base64: "
				+ base64_encoded_image.length);
		if (Base64.isBase64(base64_encoded_image)) {
			compressionOfImage(base64_encoded_image);
			base64_and_compressed_image =
compressed_image;
		}

		else
			System.out.println("U have an error in encoding
image");
		System.out.println("double check! "
				+ base64_and_compressed_image.length);
		return base64_and_compressed_image;
	}

	private byte[] compressionOfImage(byte[] base64_encoded_image)
{
		ByteArrayOutputStream baos = new
ByteArrayOutputStream();
		try {
			GZIPOutputStream gos = new
```

```
GZIPOutputStream(baos);
                        // byte[] converting_string_to_bytes =
                        // base64_encoded_image.getBytes();
                        gos.write(base64_encoded_image);
                        gos.flush();
                        gos.close();
                        compressed_image = baos.toByteArray();

                        System.out
                                        .println("Length of encoded image
after compression GZIP: "
                                                        +
compressed_image.length);
                        System.out.println("********************");
                        System.out.println("checking on compression!");
                        isCompressed(compressed_image);
                } catch (IOException e) {
                        e.printStackTrace();
                }

                return compressed_image;
        }

        public boolean isCompressed(byte[] bytes) throws IOException {
                if ((bytes == null) || (bytes.length < 2)) {
                        return false;
                } else {
                        System.out.println("********************");
                        System.out.println("Compressed successfully");
                        return ((bytes[0] == (byte)
(GZIPInputStream.GZIP_MAGIC)) && (bytes[1] == (byte)
(GZIPInputStream.GZIP_MAGIC >> 8)));
                }
        }
}
```

Compression and Encoding:

AudioCompressionEncoding.java

Methodology for compressing and encoding audio files.

```
package scenarioone.usecaseone.GZIP.base64;

/* Author : Koushik Maddipudi
 *
 */
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.util.zip.GZIPInputStream;
import java.util.zip.GZIPOutputStream;

import org.apache.commons.codec.binary.Base64;

public class AudioCompressionEncode {
	private static byte[] compressed_audio;
	private static byte[] base64_encoded_audio;
	private static byte[] compressed_encoded_audio;

	// Compression method which takes byte[] which is retrieved from
database
	// and returns an encoded byte[]
	public byte[] compression(byte[] audio_from_db) {
		System.out.println("Size of audio from database is: "
				+ audio_from_db.length);
		// Creates a new byte array output stream.
		ByteArrayOutputStream baos = new
ByteArrayOutputStream();
		try {

			// Creates a new output stream with a default buffer
size.
			GZIPOutputStream gos = new
GZIPOutputStream(baos);

			// Writes content_to_be_compressed.length bytes
from the specified
			// byte array to this output //stream.
			gos.write(audio_from_db);

			// Flushes this output stream and forces any buffered
```

```
output bytes
                        // to be written out.
                        gos.flush();

                        // Closes this output stream and releases any system
resources
                        // associated with this stream.
                        gos.close();

                        // Creates a newly allocated byte array.
                        compressed_audio = baos.toByteArray();
                        System.out.println("After applying compression size
of audio is "
                                        + compressed_audio.length);
                        System.out.println("**********************");
                        System.out.println("checking on compression");

                        // checking for compression
                        isCompressed(compressed_audio);
                        System.out.println("encoding!");

                        // calls encoding functionality
                        encoding(compressed_audio);

                        compressed_encoded_audio =
base64_encoded_audio;
                } catch (IOException e) {
                        e.printStackTrace();
                }
                System.out.println("double check! " +
compressed_encoded_audio.length);
                return compressed_encoded_audio;

        }

        // method for encoding, takes compressed byte[] and returns an
encoded
        // byte[]
        private byte[] encoding(byte[] audio_after_compression) {
                // conversion of base64
                base64_encoded_audio =
Base64.encodeBase64(audio_after_compression);

                if (Base64.isBase64(base64_encoded_audio))
                        System.out
                                        .println("Length of Audio file after
```

```
comprressing and encoding with base64: "
                                                        +
base64_encoded_audio.length);
                return base64_encoded_audio;
        }

        //method for checking compressed data.
        public boolean isCompressed(byte[] compressed_data) throws
IOException {
                if ((compressed_data == null) || (compressed_data.length <
2)) {
                        return false;
                } else {
                        System.out.println("********************");
                        System.out.println("Compressed successfully");
                        return ((compressed_data[0] == (byte)
(GZIPInputStream.GZIP_MAGIC)) &&
                                        (compressed_data[1] == (byte)
(GZIPInputStream.GZIP_MAGIC >> 8)));
                }
        }

}
```

ImageCompressionEncoding.java

Methodology for compressing images

```
package scenarioone.usecaseone.GZIP.base64;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.util.zip.*;

import org.apache.commons.codec.binary.Base64;
import org.apache.commons.io.IOUtils;

public class ImageCompressionEncode {
	private static byte[] compressed_image;
	private static byte[] base64_encoded_image;
	private static byte[] compressed_encoded_image;

	public byte[] compression(byte[] image_from_db) {
		System.out.println("Size of image from database is: "
				+ image_from_db.length);
		ByteArrayOutputStream baos = new
ByteArrayOutputStream();
		try {
			GZIPOutputStream gos = new
GZIPOutputStream(baos);
			gos.write(image_from_db);
			gos.flush();
			gos.close();
			compressed_image = baos.toByteArray();

			System.out.println("After applying compression
size of image is "
					+ compressed_image.length);
			System.out.println("********************");
			System.out.println("checking on compression");
			isCompressed(compressed_image);
			System.out.println("encoding!");
			encoding(compressed_image);
			compressed_encoded_image =
compressed_image;

		} catch (IOException e) {
			e.printStackTrace();
```

```
                }

                return compressed_encoded_image;

        }

        private byte[] encoding(byte[] image_after_compression) {

                base64_encoded_image =
Base64.encodeBase64(image_after_compression);

                if (Base64.isBase64(base64_encoded_image))
                        System.out
                                        .println("Length of Image file after
comprressing and encoding with base64: "
                                                        +
base64_encoded_image.length);
                return base64_encoded_image;
        }

        public boolean isCompressed(byte[] bytes) throws IOException
{
                if ((bytes == null) || (bytes.length < 2)) {
                        return false;
                } else {
                        System.out.println("********************");
                        System.out.println("Compressed successfully");
                        return ((bytes[0] == (byte)
(GZIPInputStream.GZIP_MAGIC)) && (bytes[1] == (byte)
(GZIPInputStream.GZIP_MAGIC >> 8)));
                }
        }
}
```

POJO classes for binding data from object to XML message

Film.java

```
package scenarioone.usecaseone.entities;

import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

@XmlRootElement(name = "Film")
@XmlType(name = "Film", propOrder = { "film_id", "title",
"description",
                "length", "image_id", "images", "audio_id", "audio" })

public class Film {

        private int film_id;
        private String title;
        private String description;
        private int length;
        private int image_id;
        private int audio_id;
        private Images images;
        private Audio audio;

        // private Video video;

        public int getImage_id() {
                return image_id;
        }

        public void setImage_id(int Image_id) {
                this.image_id = Image_id;
        }

        public int getAudio_id() {
                return audio_id;
        }

        public void setAudio_id(int audio_id) {
                this.audio_id = audio_id;
        }
```

```
public int getFilm_id() {
        return film_id;
}

public void setFilm_id(int film_id) {
        this.film_id = film_id;
}

public String getTitle() {
        return title;
}

public void setTitle(String title) {
        this.title = title;
}

public String getDescription() {
        return description;
}

public void setDescription(String description) {
        this.description = description;
}

public int getLength() {
        return length;
}

public void setLength(int length) {
        this.length = length;
}

@XmlElement(name = "image")
public Images getImages() {
        return images;
}

public void setImages(Images images) {
        this.images = images;
}

@XmlElement(name = "audio")
public Audio getAudio() {
        return audio;
}
```

```
	public void setAudio(Audio audio) {
		this.audio = audio;
	}

	// @XmlElement(name = "Video")
	// public Video getVideo() {
	// return video;
	// }
	//
	// public void setVideo(Video video) {
	// this.video = video;
	// }
}
```

Audio.java

```
package scenarioone.usecaseone.entities;

import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

@XmlRootElement(name = "audio")
@XmlType(name = "audio", propOrder = { "audio_id", "name", "size",
"type",
		"audio_location" })
public class Audio {

	private int audio_id;
	private String name;
	private int size;
	private String type;
   private byte[] content;
	private String audio_location;

//	 public byte[] getContent() {
//	 return content;
//	 }
//
//	 public void setContent(byte[] content) {
//	 this.content = content;
//	 }

	public int getAudio_id() {
		return audio_id;
```

```
	}

	public void setAudio_id(int audio_id) {
		this.audio_id = audio_id;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public int getSize() {
		return size;
	}

	public void setSize(int size) {
		this.size = size;
	}

	public String getType() {
		return type;
	}

	public void setType(String type) {
		this.type = type;
	}

	public String getAudio_location() {
		return audio_location;
	}

	public void setAudio_location(String audio_location) {
		this.audio_location = audio_location;
	}

}
```

Image.java

```
package scenarioone.usecaseone.entities;

import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

@XmlRootElement(name = "image")
@XmlType(name = "image", propOrder = { "image_id", "name", "size",
"type",
                "image_location" })
public class Images {

        private int image_id;
        private String name;
        private int size;
        private String type;
//      private byte[] content;

        // this is for scenario 2
        private String image_location;

//       public byte[] getContent() {
//       return content;
//       }
//
//       public void setContent(byte[] content) {
//       this.content = content;
//       }

        public int getImage_id() {
                return image_id;
        }

        public void setImage_id(int image_id) {
                this.image_id = image_id;
        }

        public String getName() {
                return name;
        }

        public void setName(String name) {
                this.name = name;
        }
```

```
	public int getSize() {
		return size;
	}

	public void setSize(int size) {
		this.size = size;
	}

	public String getType() {
		return type;
	}

	public void setType(String type) {
		this.type = type;
	}

	public String getImage_location() {
		return image_location;
	}

	public void setImage_location(String image_location) {
		this.image_location = image_location;
	}
}
```

Service consumer application

WSClientCall.java

This class is build for making a call to the service by stating its resource URL.

```
package serviceone.client.entity;
import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.ClientResponse;
import com.sun.jersey.api.client.WebResource;

public class WSClientCall {

	public WSClientCall(){

	}

	public Film getFilm(int id) {

		Film film = null;

		System.out.println("Id of film chosen by client: " + id);
		String resource_url =
"http://localhost:8090/Serviceone/app/filmservices/film/"
				+ id;
		try {

			Client client = Client.create();
			WebResource webResource =
client.resource(resource_url);
			ClientResponse response =
webResource.accept("application/xml")
					.get(ClientResponse.class);
			if (response.getStatus() != 200) {
				throw new RuntimeException("Failed :
HTTP error code : "
						+ response.getStatus());
			}

			String output = response.getEntity(String.class);

			System.out.println("Output from Server .... \n");
			System.out.println(output);
			XMLParsing xmlparsing = new XMLParsing();
			film = xmlparsing.unmarshaling(output);
```

```
                } catch (Exception e) {
                        e.printStackTrace();
                }

                return film;
        }

}
```

XMLParsing.java

Applying parsing techniques on the client side.

```
package serviceone.client.entity;

import java.awt.image.BufferedImage;
import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.UnsupportedEncodingException;

import org.apache.commons.codec.binary.Base64;

import client.servlet.ClientCall;

import conversion.image.audio.files.ImageConversion;

import javax.imageio.ImageIO;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Unmarshaller;

public class XMLParsing {

	public XMLParsing() {

	}

	// try JAXB
	public Film unmarshaling(String output) throws JAXBException,
IOException {
		InputStream is = new
ByteArrayInputStream(output.getBytes("UTF-8"));
		JAXBContext jc = JAXBContext.newInstance(Film.class);
		Unmarshaller u = jc.createUnmarshaller();
		Film film = (Film) u.unmarshal(is);

		film.setFilm_id(film.getFilm_id());
		film.setTitle(film.getTitle());
		film.setDescription(film.getDescription());
		film.setLength(film.getLength());
		film.setImage_id(film.getImage_id());
```

```
                Image image = new Image();

                image.setImage_id(film.getImage().getImage_id());
                image.setName(film.getImage().getName());
                image.setSize(film.getImage().getSize());
                image.setType(film.getImage().getType());
                byte[] decoded_image = Base64

        .decodeBase64(film.getImage().getContent());
                image.setContent(decoded_image);

                film.setImage(image);

                Audio audio = new Audio();

                audio.setAudio_id(film.getAudio().getAudio_id());
                audio.setName(film.getAudio().getName());
                audio.setSize(film.getAudio().getSize());
                audio.setType(film.getAudio().getType());
                byte[] decoded_audio = Base64

        .decodeBase64(film.getAudio().getContent());
                audio.setContent(decoded_audio);

                film.setAudio(audio);

                // printing stuff
                System.out.println("OUTPUT");
                System.out.println();
                System.out.println(film.getFilm_id());
                System.out.println(film.getTitle());
                System.out.println(film.getDescription());
                System.out.println(film.getLength());
                System.out.println(film.getImage_id());
                System.out.println("****************IMAGE
CLASS***************");
                System.out.println(film.getImage().getImage_id());
                System.out.println(image.getName());
                System.out.println(image.getSize());
                System.out.println(image.getType());

        System.out.println(Base64.isBase64(film.getImage().getContent()));
                System.out.println("****************AUDIO
CLASS***************");
                System.out.println(audio.getAudio_id());
```

```
                System.out.println(audio.getName());
                System.out.println(audio.getSize());
                System.out.println(audio.getType());

        System.out.println(Base64.isBase64(film.getAudio().getContent()));
                System.out.println(decoded_audio);

                return film;}}
```

Servlet class taking a request from the browser and processing it.

ClientCall.java

```
package client.servlet;
import java.io.IOException;
import java.io.OutputStream;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.ServletOutputStream;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import conversion.image.audio.files.ImageConversion;
import serviceone.client.entity.Film;
import serviceone.client.entity.WSClientCall;

/**
 * Servlet implementation class ClientCall
 */
public class ClientCall extends HttpServlet {

        private static final long serialVersionUID = 1L;

        private static String image_call_path = "ImageCall";
        private static String audio_call_path = "AudioCall";

        public ClientCall() {
                super();
                // TODO Auto-generated constructor stub
        }
        protected void doGet(HttpServletRequest request,
                        HttpServletResponse response) throws
ServletException, IOException {
                System.out.println("Film id entered by user: "
                                + request.getParameter("film_id"));
```

```
                int film_id_givenbyuser = Integer.parseInt(request
                                .getParameter("film_id"));
                long start_time = System.currentTimeMillis();
                WSClientCall wscc = new WSClientCall();

                Film film = wscc.getFilm(film_id_givenbyuser);

                response.setContentType("text/html");
                PrintWriter out = response.getWriter();

                out.println("<!DOCTYPE HTML>");
                out.println("<html><body>");
                out.println("<table width=\"753\" border=\"0\">");
                out.println("<tr><th width=175 scope=\"row\">Film ID
choosen by USER</th>");
                out.println("<td width=440>" + film.getFilm_id() +
"</td></tr>");
                out.println("<tr><th scope=\"row\">Film Title </th>");
                out.println(" <td>" + film.getTitle() + "</td></tr>");
                out.println("<tr><th scope=\"row\">Description about
Film</th>");
                out.println("<td>" + film.getDescription() + "</td></tr>");
                out.println("<tr><th scope=\"row\">Length of Film</th>");
                out.println("<td>" + film.getLength() + "</td></tr>");
                out.println("<tr><th scope=\"row\">Wallpaper </th>");

                ImageCall ic = new
ImageCall(film.getImage().getContent());

                out.println("<td>" + "<img src=\"" + image_call_path);

                out.println("\"width=250 height=100/></td></tr>");

                out.println("<tr><th scope=\"row\">Audio Song</th>");

                AudioCall ac = new
AudioCall(film.getAudio().getContent());
                out.println("<td>"
                                + " <audio autoplay=\"autoplay\"
controls=\"controls\" src=\""
                                + audio_call_path + " \">" +
"</audio></td>");
                out.println("</tr></table>");
                out.println("</body></html>");
                long end_time = System.currentTimeMillis() - start_time;
                System.out.println("Time taken for the request " +
```

```
end_time);
        }
}
```

AudioCall.java

Ajax call made to the service by client.

```
package client.servlet;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.ServletOutputStream;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import serviceone.client.entity.Audio;
/**
 * Servlet implementation class AudioCall
 */
public class AudioCall extends HttpServlet {
        private static final long serialVersionUID = 1L;

        private static byte[] audio_content;

        public AudioCall() {
                super();
                // TODO Auto-generated constructor stub
        }

        public AudioCall(byte[] audio_content) {
                this.audio_content = audio_content;
                System.out.println("Hello I am in audio servlet");
        }

        protected void doGet(HttpServletRequest request,
                        HttpServletResponse response) throws
ServletException, IOException {

                response.setContentType("audio/mp3");
                System.out.println(audio_content.length);
```

```
                ServletOutputStream ostr = response.getOutputStream();
                ostr.write(audio_content);
                ostr.flush();
                ostr.close();
        }}
```

ImageCall.java

Browser making request to Image via a servlet.

```
package client.servlet;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.ServletOutputStream;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import serviceone.client.entity.Image;
public class ImageCall extends HttpServlet {
        private static final long serialVersionUID = 1L;
        private static byte[] image_content;
        public ImageCall() {
                super();
                }
        public ImageCall(byte[] image_content) {
                ImageCall.image_content = image_content;
        }
        protected void doGet(HttpServletRequest request,
                        HttpServletResponse response) throws
ServletException, IOException {
                System.out.println("hello man I am in image servlet class");
                response.setContentType("image/jpg");
                System.out.println(image_content.length);
                ServletOutputStream ostr = response.getOutputStream();
                ostr.write(image_content);
                ostr.flush();
                ostr.close();
        }

}
```

BIBLIOGRAPHY

[1] List of HTTP header files.

http://en.wikipedia.org/wiki/List_of_HTTP_headers (Last visited on June 3rd, 2012)

[2] HTTP Template, source wiki

http://en.wikipedia.org/wiki/Template:HTTP (Last visited on June 3rd, 2012)

[3] Aaron Skonnard, Pluralsight **"**Some common HTTP status codes, MSDN" Microsoft publications

[4] Boris Lublinsky, "Defining SOA as an architectural style**",** IBM Research paper

[5] David Sprott and Lawrence Wilkes**,** "Understanding Service-Oriented Architecture**",** Microsoft publication

[6] Mark Colan**, "**Service-Oriented Architecture expands the vision of web services, Part 1**",** IBM Research paper

[7] R. Fielding. Architectural Styles and The Design of Network-based Software Architectures. PhD thesis, University of California, Irvine, 2000.

[8] REST, Constraints, http://en.wikipedia.org/wiki/Representational_state_transfer (Last visited on June 3rd, 2012)

[9] C. Pautasso, O. Zimmermann and F. Leymann, "RESTful Web Services vs. "Big" Web Services: Making the Right Architectural Decision." IW3C2, April 2008.

[10] REST, RESTful Web Services, http://en.wikipedia.org/wiki/Representational_state_transfer (Last visited on June 3rd, 2012)

[11] Silmaril, What is XML for? http://xml.silmaril.ie/whatfor.html (Last visited on June 3rd, 2012)

[12] Chris Herborth, Dealing with data in XML, IBM Research paper

[13] MVC Architecture, Wikipedia

http://en.wikipedia.org/wiki/Model-view-controller (Last visited on June 3rd, 2012)

[14] Java Documentation, JAXBContext (Last visited on June 3rd, 2012)

http://docs.oracle.com/javase/6/docs/api/javax/xml/bind/JAXBContext.html

[15] Gson, Library for JSON data format

http://code.google.com/p/google-gson/ (Last visited on June 3rd, 2012)

[16] Java Documentation, Class GZIPOutputStream (Last visited on June 3rd, 2012)

http://docs.oracle.com/javase/1.4.2/docs/api/java/util/zip/GZIPOutputStream.html

[17] Java Documentation, java.util.zip (Last visited on June 3rd, 2012)

http://docs.oracle.com/javase/1.4.2/docs/api/java/util/zip/package-summary.html

[18] Java Documentation, OutputStream (Last visited on June 3rd, 2012)

http://docs.oracle.com/javase/1.4.2/docs/api/java/io/OutputStream.html

[19] Base64 encoding, Wikipedia

http://en.wikipedia.org/wiki/Base64 (Last visited on June 3rd, 2012)

[20] Same Origin Policy

http://en.wikipedia.org/wiki/Same_origin_policy (Last visited on June 3rd, 2012)

[21] Alex Rodriguez, "RESTful Web services: The basics" IBM Research paper

[22] ODATA http://www.odata.org/ (Last visited on June 3rd, 2012)

ABBREVIATIONS

1. **AJAX** Asynchronous JavaScript + **XML**
2. **API** Application Programming Interface
3. **CRUD** Create Read Update Delete
4. **HTML** Hypertext Markup Language
5. **HTTP** Hypertext Transfer Protocol
6. **IDE** Integrated Development Environment
7. **JSON** JavaScript Object Notation
8. **REST** Representational State Transfer
9. **ROA** Resource Oriented Architecture
10. **RPC** Remote Procedure Call
11. **SOA** Service Oriented Architecture
12. **SOAP** Simple Object Access Protocol
13. **UI** User Interface
14. **URI** Uniform Resource Identifier
15. **URL** Uniform Resource Locator
16. **W3C** World Wide Web Consortium
17. **WWW** World Wide Web
18. **XHTML** Extensible Hypertext Markup Language
19. **XML** Extensible Markup Language