Passing Large Data-sets to a Web Service

An Technique of Sending Bulk Data over HTTP

(Proposal)

By

John Avera

Florida Institute of Technology

CIS 5080 – Projects in Computer Information Systems

For

Mr. Steve Johnson

May 22, 2011

Introduction 3

Web Service Basics 3

SOAP Message Structure 4

DAWS Value Proposition 4

Algorithms / Project Solution 4

Message Transmission Optimization Mechanism (MTOM) 5

Base64 Encoding 5

XML-binary Optimized Packaging (XOP) 6

Multipurpose Internet Mail Extensions (MIME) 6

Implementation 7

Platform 7

Development Tools 7

Programming Language and Code Libraries 7

Implementation Procedure 7

Schedule 8

References 9

# Introduction

Data Access Web Service (DAWS) will distribute large datasets across the network using the Hyper-Text Transfer Protocol.

There are many applications that need access to large data sets such as logistical applications, climate reporting and analysis systems, and location services. Many techniques and current protocols are platform dependent and proprietary. Web Services offers an ability to send data over open standards that are platform independent. The issue with sending data as a web services is Hyper-Text Transfer Protocol (HTTP) used in web services is a stateless protocol. It is intended to send character data across the network as a request to a response without maintaining a connection from client to server. It was not intended to send large datasets. There is a technique; however, that is simple and maximizes current standards already in place to send bulk data-sets inside Simple Object Access Protocol (SOAP) messages. This project will provide a technique with algorithms of sending large data-sets to a web service to be consumed by a client and persisted on the client machine. It will also demonstrate a potential event-driven distributed system using the proposed data access web service.

### Web Service Basics

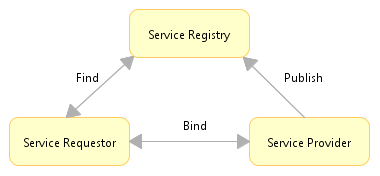
Web Services are typically Application Programming Interfaces (API) or Web APIs that can be accessed over a network, such as the Internet, and executed on a remote system hosting the requested services. A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-process able format (specifically Web Services Description Language (WSDL)). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically transmitted using HTTP with an Extensible Markup Language (XML) serialization in conjunction with other Web-related standards. A Service-Oriented Architecture (SOA) is a pattern realized thru web services, and is composed of three roles: service provider, service registry, and service requester that perform basic operations such as find, bind, and publish as shown in Figure 1.

Figure 1: Service-Oriented Architecture

The Service Provider is an organization (or individual) that provides services by creating, publishing, maintaining, and publishing their services. From a business perspective, the Service Provider owns the service, whereas from an architectural perspective, it is the platform that holds the implementation of the service.

The Service Registry provides a repository of service descriptions (WSDL). These descriptions are published by the service provider. Service Requesters will search the repository to identify the needed services and obtain the binding information for these services. A service registry can either be public or private.

The Service Requester is an organization (or individual) that looks for a service to fulfill its requirements. From a business perspective, it is a business that wants to consume a particular service. From a architectural perspective, it is an application that looks for and invokes a service.

### SOAP Message Structure

A SOAP message is encoded as an XML document consisting of an <Envelope> element, which contains an optional <Header> element, and a mandatory <Body> element. The <Fault> element, contained within the <Body>, is used for reporting errors (Figure 2).

The SOAP <Envelope> is the root element in every SOAP message and contains two child elements, an optional <Header> and a mandatory <Body>.

The SOAP <Header> is an optional sub-element of the SOAP envelope and is used to pass application-related information that is to be processed by SOAP nodes along the message path.

Figure 2: SOAP Message Structure

The SOAP <Body> is a mandatory sub-element of the SOAP envelope which contains information intended for the ultimate recipient of the message.

The SOAP <Fault> is a sub-element of the SOAP body which is used for reporting errors.

## DAWS Value Proposition

DAWS will consist of an open source HTTP Server, an open source web service framework, custom components that implement DAWS, and an open source database server. A user will be able to send a request for a data set that contains over 10,000 rows of data provided by DAWS, received the response containing the data, and parse and persist the data on the their client machine. The user will receive large data sets from a database server over HTTP using open standards without establishing and maintaining an open connection to the database server. This concept enables DAWS to service data requests from many clients on different platforms simultaneously with few database connections.

# Algorithms / Project Solution

This section describes the unique problem and the proposed solution that will be known as DAWS.

The most significant challenge in implementing a data access web service is actually sending the data inside a SOAP <Body> in the most efficient manner possible. SOAP messages are intended to be text based to be processed by an XML parser. Typically, clients connect to databases using a specific driver for that particular database. When a client queries the database using the driver, the database returns a result-set containing the data in binary form. To provide data as a web service using open standards, the result-set has to be embedded in or appended to a SOAP message. This requires the data in the result-set to be written out as XML with a pre-determined schema to describe the rows, columns, and data types. Once the data is converted to XML and embedded inside the SOAP <Body>, it is then sent over the network via HTTP to the service requestor to be consumed. With a result-set of over 10,000 rows of data, the body of the SOAP message can be very large which consumes bandwidth when publishing over the network.

Since the goal of DAWS is to provide a technique of sending large datasets in an efficient manner that is simple and maximizes current standards already in place, our technique will have to include compressing (deflate) the data in the SOAP body using a standard compression algorithm. The deflation algorithm DAWS will use is a variation of LZ77 (Lempel-Ziv 1977) which is used by gzip and zip. Most modern programming languages have APIs to support gzip and zip operations to include Java and C#. After compressing the XML into binary form, we will have to provide a method to include the data with our SOAP message.

## Message Transmission Optimization Mechanism (MTOM)

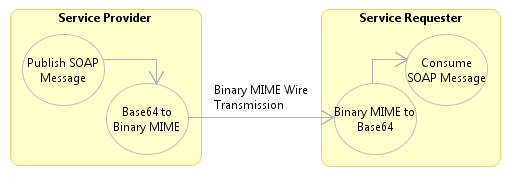
SOAP MTOM is a W3C Recommendation. It describes an abstract feature and a concrete implementation for optimizing the transmission and/or wire format of SOAP messages. The concrete implementation depends on Base64 Encoding, XOP, and MIME specifications for carrying SOAP messages. When implemented, MTOM provides a mechanism of efficiently transmitting binary data, such as images, video, PDF files, and compressed ZIP files between systems. These are the specifications, standards, and recommendations DAWS follows to efficiently send bulk binary data over HTTP. Figure 3 demonstrates the steps involved in transmitting data from Service Provider to Service Requester.

Figure 3: MTOM Transmission Steps

### Base64 Encoding

Base64 is an encoding scheme that represents binary data in an ASCII string format by translating it into radix-64 representation (65 textual characters). It is commonly used in email applications via MIME to store complex and binary data in XML. This encoding allows DAWS to include the compressed binary ZIP file of the large dataset in the SOAP message body. A sample SOAP Body with Base64Binary encoded element <data> is shown below:

<Envelope>

<Body>

<data>HdsVeRi0x93YxiOf7Yisjh09…</data>

</Body>

</Envelope>

We can stop here though, because base64 encoded data is approximately 33% larger than the binary data itself. This inflation of the data wastes processing time, network bandwidth, and transmission time. DAWS will need a method to optimize the base64binary encoded data to increase efficiency.

### XML-binary Optimized Packaging (XOP)

XOP is a specification that provides a means of more efficiently serializing XML Information Sets that have binary data. It also provides a mechanism to de-serialize the binary data back into the XML Information Set. The benefit of XOP is it allows the binary data part of an XML Infoset to be serialized without going through the XML serlializer. The XML serialization of XML Infoset is text based which means the binary data will have to be encoded using base64. The use of XOP avoids this by extracting the binary data out of the XML Infoset so it can be serialized in a different way. XOP reduces the size of the serialization since base64 encoding is known to increase binary data as much as 33%. This reduction in size increases efficiency for DAWS when transmitting large data sets.

### Multipurpose Internet Mail Extensions (MIME)

MIME is an internet standard that extends the format of e-mail to support text in character sets other than ASCII, non-text attachments, message bodies with multiple parts, and header information in non-ASCII character sets. MIME standards also define content types for communication protocols like HTTP. It will be used in our technique of implementing DAWS to define binary data appended to our SOAP message separated by a MIME boundary as shown below:

<Envelope>

<Body>

<data><xop:Include href=”cid:abc123”/></data>

</Body>

</Envelope>

--MIMEBoundary

Content-Type: application/zip

Content-ID: <abc123>

…binary data is here

…

--MIME Boundary

# Implementation

This section describes the resources that will be used to implement DAWS.

## Platform

DAWS will be a Java Web Service running under Apache Axis2 framework deployed on Apache Tomcat Java Web Server connected to an Apache Derby Database Server. All these components will be installed on an Intel platform running Windows 7 Operating System (OS). In actuality, the platform and OS are independent as long as they can support and run a Java Virtual Machine (JVM) version 6 or above. Information and downloads for Apache Axis2, Tomcat, and Derby can be found at <http://axis.apache.org>, <http://tomcat.apache.org/>, and <http://db.apache.org/derby>. Information and downloads for Java can be found at <http://www.java.com> .

## Development Tools

Development will be done on an Intel machine running Windows 7 OS using Eclipse IDE for Java EE Developers (Helios SR2 Packages). Information and downloads can be found at <http://www.eclipse.org/> .

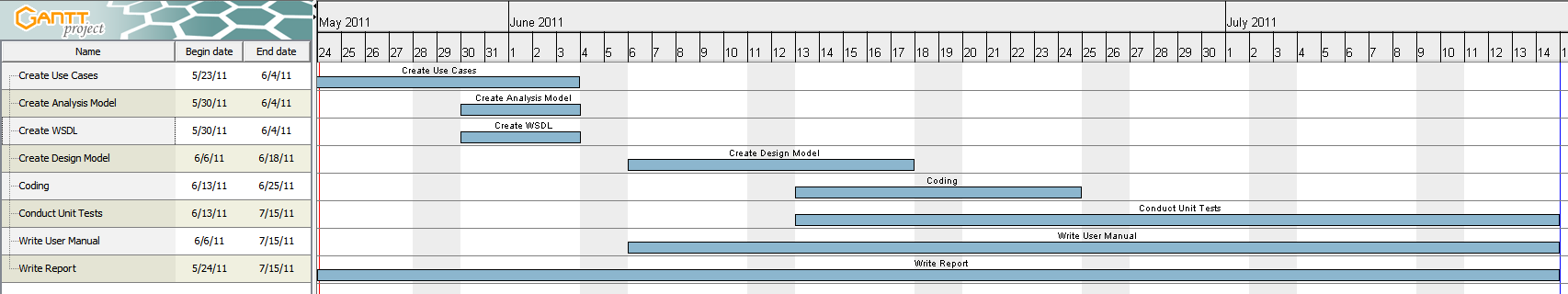
## Programming Language and Code Libraries

DAWS will be programmed in the Java programming language. The code libraries required are the Apache Axis2 libraries that come with the framework.

## Implementation Procedure

I plan to start by developing high-level use cases and sequence diagrams for the web service. Also during this phase I’ll design the interface using Web Services Description Language (WSDL). Next, I’ll create an analysis model using the modeling features in Eclipse for the Service Requestor and Service Provider which includes operations to query, compress, transmit, parse, and persist the data. Each operation will be modeled as a mediator function controlled by a single mediator object. The mediator object will basically be an aggregate of all the mediator functions. Once I feel I have a working version of the model, I will use the transform feature of Eclipse to generate the skeleton code and begin writing the implementation logic for each class and its methods. I will also use the Axis2 framework to generate the web service implementation classes from the WSDL design. When coding is complete, I will package DAWS as a Service Archive (SAR file) and deploy on the Tomcat server running Axis2 framework.

# Schedule



# References

SOAP Messaging Transmission Optimization Mechanism, Martin Gudgin, Noah Mendelsohn, Mark Nottingham, Herve Ruellan, et. al., Editors. World Wide Web Consortium, 25 January 2005. <http://www.w3.org/TR/soap12-mtom/>

SOAP Version 1.2 Part 1: Messaging Framework, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, et. al., Editors. World Wide Web Consortium, 24 June 2003. <http://www.w3.org/TR/soap12-part1/>

XML-binary Optimized Packaging, Mark Nottingham, Noah Mendelsohn, Martin Gudgin, and Hervé Ruellan, Editors. World Wide Web Consortium, 25 January 2005. <http://www.w3.org/TR/xop10/>

Jacob Ziv and Abraham Lempel; A Universal Algorithm for Sequential Data Compression, IEEE Transactions on Information Theory, 23(3), pp. 337–343, May 1977.

Tong Ka Iok, K. (2008). *Developing Web Services with Apache Axis2*. Sydney: TipTec Development.