MASTER'S THESIS SUBMISSION FORM

Type of Document	M. S. Thesis
Author	BIJU T MANIAMPADAVTHU
Author's e-mail	
Thesis Title	Message Optimization Enhanced Logger System
Degree	Master of Science
Subject	Computer Science
Keywords	XML SOAP XOP MTOM Logging "binary XML"
Committee Members	1. Prof. 2. Prof. 3. Prof. 4. Prof.
Date of Defense	
Availability	
Abstract	This document describes about extending the functionality of a Web services based logger system using SOAP Message Transmission Optimization (MTOM) recommendation from W3C. MTOM uses XML binary optimized packaging (XOP) recommendation from W3C to provide an optimized XML serialization for binary data in XML. This work adds binary transport and locale specific translation (localization) feature for the binary log messages stored in the log repository.

Signature_____ **Date:**_____

CSIS THESIS/DISSERTATION USE AND ELECTRONIC PUBLICATION CONSENT FORM

I, BIJU T MANIAMPADAVATHU, (the "Author") herby grant to Pace University the right to display on Pace's Web site my thesis titled "Message Optimization Enhanced Logger System" (the "Materials") alone or in combination with other thesis papers for use by researchers, students and faculty in any and all formats and by means of technology now known or hereafter to become known, and to reserve copies in its libraries in hard copy or electronic form.

Pace University shall give the Author credit in connection with the use of the Materials by identifying Author in a prominent manner.

Signature_____ **Date:**_____

Pace University

Date:

Student's Name: Biju T. Maniampadavathu

Date of Graduation: Spring 2005

Title of Thesis: Message Optimization Enhanced Logger System

This will certify that the above thesis has been approved in fulfillment of the thesis option for the Degree of Master of Science in Computer Science.

Student's Signature _____

Thesis Advisor _____

Committee Member _____

Committee Member _____

Committee Member _____

Department Chair _____

Dean _____

Date _____

Thesis Project for the degree of Master of Science in Computer Science

Message Optimization Enhanced Logger System

Pace University
School of Computer Science and Information Systems
Biju T. Maniampadavathu
Professor Mehdi Badii – Thesis Advisor
Fall 2004 – Spring 2005

Table of Contents

Table of Figures

Abstract

This document describes about extending the functionality of a Web services based logger system using SOAP Message Transmission Optimization (MTOM) recommendation from W3C. MTOM uses XML binary optimized packaging (XOP) recommendation from W3C to provide an optimized XML serialization for binary data in XML. This work adds binary transport and locale specific translation (localization) feature for the binary log messages stored in the log repository.

1 Introduction

The article titled “A Logger System based on Web services” [ref (§1)] describes about a general purpose set of component interfaces for logging data in a distributed, heterogeneous computing environment. This logger system serves as a manageable repository for the log records and intermediates between the log artifact producers and the log artifact consumers. The logger system also provides administrative functions such as creating new logs, purging obsolete records, and managing retention policies. The log records are persisted in a proprietary binary format, which are available for read through the read operation of LogBrowseSession interface. The current work described here, discusses about extending the logger system by providing localization feature for the user requested log artifacts. The binary log records requested by a user are filtered from the persistent storage are delivered to the requestor using a web service. The binary log records are serialized using the XOP [ref (§3)] specification and are transported over the HTTP protocol in a SOAP [ref (§2)] packet as described by the MTOM [ref (§4)] specification. At the user location, the binary log record is extracted from the SOAP packet and is translated to the local language.

The rest of this section gives a brief introduction all the technologies used in this work. All these technologies are described in detail in the following sections of this paper.

1.1 Web services

A web service is an interface that describes a collection of operations that are network accessible through a published URI, and whose public interfaces and bindings are defined and described using standardized XML. Web Services fulfill a specific task or a group of tasks. A web service is described using WSDL, a standard, formal XML notion, called its service description, which provides all the details necessary to interact with the service, including message formats (that detail the operations), transport protocols, and locations, Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

1.2 Extended Markup Language (XML)

Extensible Markup Language (XML) [ref (§5)] is a simple, very flexible text format derived from SGML (ISO 8879). Originally designed to meet the challenges of large-scale electronic publishing, XML is also playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere. The Extensible Markup Language (XML) is a subset of SGML that is completely described in this document. Its goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML.

XML documents are made up of storage units called entities, which contain either parsed or unparsed data. Parsed data is made up of characters, some of which form character data, and some of which form markup. Markup encodes a description of the document's

storage layout and logical structure. XML provides a mechanism to impose constraints on the storage layout and logical structure.

1.3 Simple Object Access Protocol (SOAP)

Simple Object Access Protocol (SOAP) is a W3C reccomendation which provides the definition for the XML-based information which can be used for exchanging structured and typed information between peers in a decentralized, distributed environment. SOAP uses XML technologies to define an extensible messaging framework, which provides a message construct that can be exchanged over a variety of underlying protocols. This framework has been designed to be independent of any particular programming model and other implementation specific semantics. SOAP is fundamentally a stateless, one-way message exchange paradigm, which can be used as a building block for creating more complex interaction patterns (e.g., one-way, request/response, notification, notification/responses, etc.) by combining one-way exchanges with features provided by an underlying protocol and/or application-specific information. SOAP does not dictate the semantics of any application-specific data it conveys, such as the routing of SOAP messages, reliable data transfer, firewall traversal, etc. However, SOAP provides the framework by which application-specific information may be conveyed in an extensible manner. Also, SOAP provides a full description of the required actions to be taken by a SOAP processor node on receiving a SOAP message.

1.4 XML and Binary Data

The desire to integrate XML with pre-existing data formats has been a longstanding and persistent issue for the XML community. Users often want to leverage the structured, extensible markup conventions of XML without abandoning existing data formats that do not readily adhere to XML 1.0 syntax. Often, users want to leave their existing non-XML formats as is, to be treated as opaque sequences of octets by XML tools and infrastructure. Such an approach allows widely used formats such as JPEG and WAV to peacefully coexist with XML. Currently two methods are used to include binary data inside the XML. These are embedding the binary data as content and referencing binary data from the XML. As XML is increasingly used as a message format (e.g., SOAP), the interest in integrating opaque data with XML has increased to the point where there are at least two competing proposals for doing so (SOAP with Attachments (SwA) and WS-Attachments). Both these specifications do not efficiently provide a mechanism to include binary data in the XML. W3C has come with a new recommendation called XML binary optimized packaging (XOP) to address this issue.

1.5 XML Binary Optimized Packaging (XOP)

XOP is the latest W3C recommendation for XML serialization. The W3C XML-binary Optimized Packaging format uses multi-part MIME to allow raw binary data to be included into an XML 1.0 document without resorting to base64 encoding. XOP serializes the XML infoset into a MIME multipart-related package with an XML document as the root part. That root part is very similar to the XML serialization of the document, except that base64-encoded binary data is replaced by a reference to one of the MIME parts, which isn't base64 encoded. This allows avoiding the bulk and overhead in the processing associated with encoding, which is the only way that you can fit binary

data directly into an XML world. XOP can be applied to any XML Infoset including SOAP messages.

1.6 Message Transmission Optimization (MTOM)

MTOM is a W3C recommendation which describes a mechanism for optimizing the transmission and/or wire format of a SOAP message by selectively re-encoding portions of the message while still presenting an XML Infoset to the SOAP application. MTOM is a description of how XOP is layered into the SOAP HTTP transport.

1.7 A Logger system based on Web services

The Logger System referred to in this paper serves as a manageable repository of log records. The term *log record* refers to the atomic unit read or written through the Logger System's interfaces. Components that write log records to the Logger System are referred to as *log artifact producers* or simply *producers*. Components that read from the Logger System are referred to as *log artifact consumers* or simply *consumers*. The Logger System serves as an intermediary, decoupling log artifact producers and log artifact consumers. Log artifacts written by log producers may or may not be read at a later time by log consumers. A component that manages a Logger System is referred to as a *log manager*. Log managers maintain the Logger System, performing a variety of administrative functions such as creating new logs, purging obsolete records, and managing retention policies. The Logger Systems defines four interfaces The `log:LogManager` interface, the `log:Log` interface, the `log:`LogConnection interface and the `log:LogBrowseSession` interface. The `log:Log` interface acts as a factory for all the `log:LogConnection` resources. The `createLogConnection()` method of the `log:Log` interface creates the `log:LogConnection` resources. The `destroy()` method of the `log:Log` interface is used to destroy the log resource. When a log resource is destroyed, then all log records associated with that log is also lost. All read and write access to the records in the log are made through the `log:LogConnection` interface. The createLogBrowseSession operation of the `log:LogConnection portType` acts as a factory for `log:LogBrowseSession` resources. The `log:LogConnection` resources can be explicitly destroyed using its inherited `destroy()` method. When a `log:LogConnection` resource is explicitly destroyed all the associated `log:LogBrowseSession` resources are also lost. The `log:LogBrowseSession` interface is used to read log records from a log.

2 Web Services

2.1 What is Web Services?

Web services are the fundamental building blocks in the move to distributed computing on the Internet. The first major shift in computing happened in the 80's with the introduction of Component Architecture, which enabled code reuse. Initially the code reuse was used for developing the graphical user interface (GUI) intensive applications. Towards the beginning of 90's the component architecture was used for network architectures also. This led to the development of distributed computing. Two main architectures have been popular in the distributed computing development. The Object Management Group (OMG) developed the first specification of CORBA (Common Object Request Broker Architecture) in 1991, and Microsoft developed the distributed version of their COM component architecture in 1995. Later when Sun released the Java platform, RMI (Remote Method Invocation) was introduced and was later enhanced with the J2EE (Enterprise Java) specifications. These distributed computing technologies where restricted to private networks and had no support for interoperability between them. This deficiency of the distributed computing is solved by the Web Services technology which made its entry with the beginning of the 21st century.

A Web service is a reusable piece of software that interacts by exchanging messages over the network through XML (Extensible Markup Language), SOAP (Simple Object Access Protocol), and other industry recognized standards allowing applications to communicate with each other independent of platform and programming language. A Web service is an interface that describes a collection of operations that are network-accessible through standardized XML messaging. A Web service is described using a standard, formal XML notion WSDL (Web Services Description Language), called its service description. It covers all the details necessary to interact with the service, including message formats (that detail the operations), transport protocols and location. The interface hides the implementation details of the service, allowing it to be used independently of the hardware or software platform on which it is implemented and also independently of the programming language in which it is written. This allows and encourages Web Services-based applications to be loosely coupled, component-oriented, cross-technology implementations. Web Services fulfill a specific task or a set of tasks. They can be used alone or with other Web Services to carry out a complex aggregation or a business transaction.

The Web Services Architecture working group of the World Wide Web Consortium (W3C – http://www.w3.org) manages the evolution of the Web Services specifications through and open standards process. W3C provides the following definition for a web service. "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-processable format (specifically WSDL). Other systems interact with the web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."

The wide spread adoption of the World Wide Web infrastructure during the internet boom period is widely attributed to the simplicity of the protocols used for its implementations. The WWW is based on HTTP (Hyper Text Transfer Protocol) and HTML (Hyper Text Markup Language). Every business organization has realized the potential of the internet and has developed adequate web infrastructures. People skilled in the web infrastructure development are also available. The fact that web services is developed based on open standards and can be layered over the existing web infrastructure leads to the easy adoption of web service technologies. Web services use the XML Infoset as the primary data representation format. The service description is provided as per the WSDL specifications. The message exchange for web service interaction is done using the SOAP specifications. The web services are defined independently of the underlying messaging transport mechanism in use. It allows the use of many alternative transports for message exchange, and allows both synchronous and asynchronous message transfer and processing. All these specifications which form the building blocks of the web services are developed using the open standards and managed by standards organizations like W3C and Oasis. The open standards development process helps the design of a simple, vendor neutral and interoperable web services standards. With web services acting as a ubiquitous integration infrastructure supported by most organizations, the task of building cross-organizational information systems becomes much easier and cheaper. The web services technologies help organizations develop service offerings based on the loosely-coupled component-based software development.

2.2 Web Services Architecture

The web services architecture is defined using Service Oriented Architecture (SOA) pattern. SOA is a component model that inter-relates the different functional units of an application, called *services*, through well-defined interfaces and contracts between these services. The interface is defined in a neutral manner that should be independent of the hardware platform, the operating system, and the programming language the service is implemented in. This allows services, built on a variety of such systems, to interact with each other in a uniform and universal manner. In SOA all software components or functional units that are available to be invoked are modeled as services.

This feature of having a neutral interface definition that is not strongly tied to a particular implementation is known as *loose coupling* between services. The benefit of a loosely-coupled system lies in its agility and the ability to survive evolutionary changes in the structure and implementation of the internals of each service, which make up the whole application. Tight-coupling on the other hand means that the interfaces between the different components of an application are tightly interrelated in function and form, thus making them brittle when any form of change is required to parts or the whole application.

The SOA defines three roles: a service requestor, a service provider and a service registry. The three main roles and the interaction between the roles are depicted in figure 1.

A *Service Provider* is responsible for creating a service description, deploying that service in a runtime environment that makes it accessible by other entities over the

network, publishing that service description to one or more service registries, and receiving Web service invocation messages from one or more service requestors.

A *service requestor* is responsible for finding a service description published to one or more service registries and is responsible for using service descriptions to bind to or invoke Web services hosted by service providers. Any consumer of a Web service can be considered as a service requestor.

A *service registry* is responsible for advertising Web services descriptions published to it by service providers and for allowing service requestors to search the collection of service descriptions contained within the service registry by replying to the queries from the service requestor on the availability of the service and the quality of service (QoS) provided by an available service. The service registry role is simple to be a match maker between service requestor and service provider. Once the service registry makes the match, it's no longer needed in the picture; the rest of the interaction takes place directly between the service requestor and the service provider for the Web service invocation.

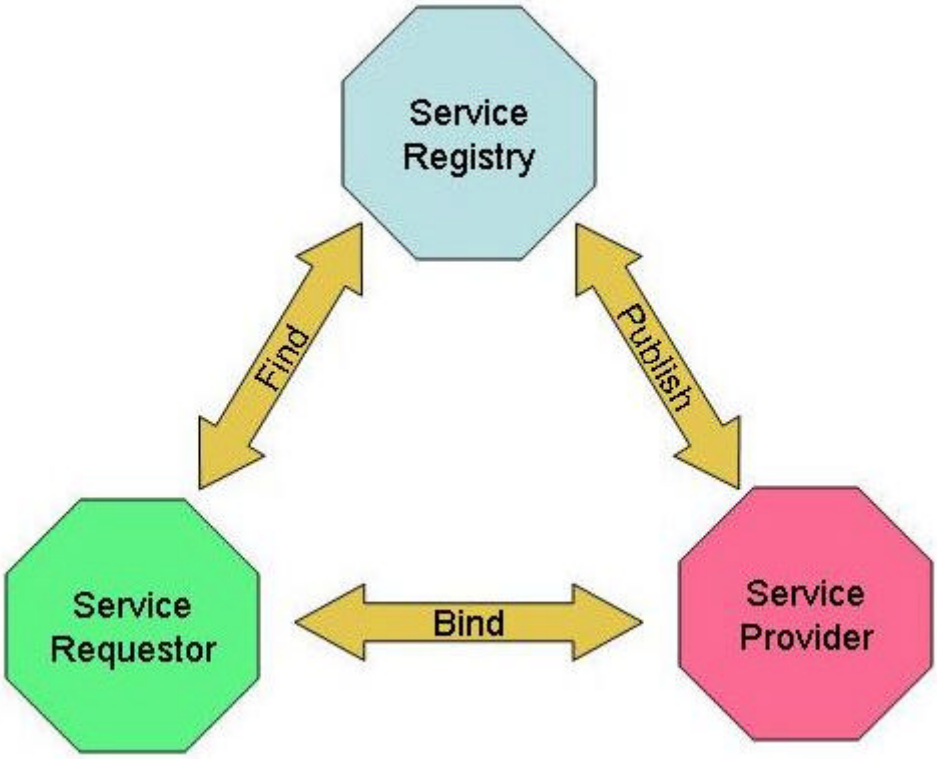

Figure 1 Service Oriented Architecture

For an application to take advantage of Web Services, three behaviors must take place: publication of service descriptions, lookup or finding of service descriptions, and binding or invoking of services based on the service description. These behaviors can occur singly or iteratively. These operations are called publish, find and bind.

The *publish* operation is an act of service registration or service advertisement. It acts a contract between the service provider and the service registry. When a Web service provider publishes its service description to a service registry it is advertising the details of an available Web service to the potential service requestors. Any action that makes a WSDL document available to a service requestor, at any stage of the service requestor's lifecycle, qualifies as service publication. The simplest, most static example at this layer is the service provider sending a WSDL document directly to a service requestor. This is called direct publication. E-mail is one vehicle for direct publication. Direct publication is useful for statically bound applications. Alternatively, the service provider can publish the WSDL document describing the service to a host's local WSDL registry, private UDDI (Universal Description Discovery and Integration) registry or the UDDI operator node.

The *find* operation is a contract between the service requestor and the service registry. With the find operation the service requestor searches the available services based on requirements like type of service, QoS provided etc. The result of the find operation is a list if service descriptions that match the find criteria. The find operation can be involved in two different lifecycle phases for the service requestor: at design time to retrieve the services interface description for the program development and at runtime to retrieve the service's binding and location description for invocation.

The *bind* operation is the contract between the service provider and the service requestor. The service requestor identifies the service it is going to use from the find operation. The service requestor then invokes or initiates an interaction with the service at runtime using the binding details in the service description to locate, contact and invoke the service.

The service description is the connecting factor that makes the SOA work. The service provider publishes its service description to the service registry using the publish operation. The service requestor queries the service registry for service descriptions using the find operation. The service registry provides the service requestor the service description of the matching service. The service requestor then uses the information provided in the service description to invoke the service from the service provider through the bind operation.

The following sections the basic building blocks of the Web services like the data capture methods, core messaging constructs and service description specifications are described in detail.

2.3 Web Services Resource Framework (WSRF)

The Web services architecture defines a service-oriented distributed computing model in which services interact by exchanging XML documents. The basic elements of the Web services architecture define the syntax for information exchange. Various efforts are now underway to augment this base architecture with additional conventions so that interacting services can accomplish more sophisticated behaviors such as authentication, transactions, and reliable messaging in standard ways.

A set of conventions intended to formalize interactions with state was introduced into the web services technology based on the realization that there are many ways of representing state in Web services, however there does not exist an agreed upon convention that would promote interoperability among Web services and their interactions with stateful resources. Even those Web service implementations commonly described as stateless frequently allow for the manipulation of state, i.e., data values that persist across, and evolve because of, Web service interactions. For example, an online airline reservation system must maintain state concerning flight status, reservations made by specific customers, and the system itself: its current location, load, and performance. Web service interfaces that allow requestors to query flight status, make reservations, change reservation status, and manage the reservation system must necessarily provide access to this state.

Web services must often provide their users with the ability to access and manipulate state, i.e., data values that persist across, and evolve as a result of, Web service interactions. And while Web services successfully implement applications that manage state today, it is desirable to define Web service conventions to enable the discovery of, introspection on, and interaction with stateful resources in standard and interoperable ways. WS-Resource Framework (WSRF) defines these conventions and does so within the context of established Web services standards.

The WS-Resource Framework is a set of six Web services specifications that define what is termed the WS-Resource approach to modeling and managing state in a Web services context. The WS-Resource approach, models state as stateful resources and codify the relationship between Web services and stateful resources in terms of the implied resource pattern, a set of conventions on Web services technologies, particularly XML, WSDL, and WS-Addressing. WS-Resource is described in terms of a stateful resource and an associated Web service. Modeling stateful resources with web services also describe an approach for making the properties of a WS-Resource accessible through its Web service interface, and for managing and reasoning about a WS-Resource's lifetime. The six WS-Resource Framework specifications are described below:

- *WS-ResourceLifetime* defines mechanisms for WS-Resource destruction, including message exchanges that allow a requestor to destroy a resource, either immediately or by using a time-based scheduled resource termination mechanism.

- *WS-ResourceProperties* defines how the data associated with a stateful resource can be queried and changed using Web services technologies. This allows a standard means by which data associated with a WS-Resource can be accessed by clients. The declaration of the WS-Resource's properties represents a projection of or a view on the WS-Resource's state. This projection represents an implied resource type which serves to define a basis for access to the resource properties through Web service interfaces.
- *WS-Notification* defines mechanisms for event subscription and notification using a topic-based publish/subscribe pattern and also includes three normative specifications.

- *WS-BaseNotification* defines the Web services interfaces for NotificationProducers and NotificationConsumers. It includes standard message exchanges to be implemented by service providers that wish to act in these roles, along with operational requirements expected of them. This is the base specification on which the other WS-Notification specification documents depend.

- *WS-BrokeredNotification* defines the Web services interface for the NotificationBroker. A NotificationBroker is an intermediary which, among other things, allows publication of messages from entities that are not themselves service providers. It includes standard message exchanges to be implemented by NotificationBroker service providers along with operational requirements expected of service providers and requestors that participate in brokered notifications.

- *WS-Topics* define a mechanism to organize and categorize items of interest for subscription known as "topics." WS-Topics defines three topic expression dialects that can be used as subscription expressions in subscribe request messages and other parts of the WS-Notification system. It further specifies an XML model for describing metadata associated with topics.

- *WS-RenewableReferences* defines a conventional decoration of a WS-Addressing endpoint reference with policy information needed to retrieve an updated version of an endpoint reference when it becomes invalid.

- *WS-ServiceGroup* defines a means by which Web services and WS-Resources can be aggregated or grouped together for a domain specific purpose. In order for requestors to form meaningful queries against the contents of the ServiceGroup, membership in the group must be constrained in some fashion. The constraints for membership are expressed by intension using a classification mechanism. Further, the members of each intension must share a common set of information over which queries can be expressed.

- *WS-BaseFaults* defines an XML Schema type for a base fault, along with rules for how this fault type is used by Web services. A designer of a Web services application often uses interfaces defined by others. Managing faults in such an application is more difficult when each interface uses a different convention for representing common information in fault messages. Support for problem determination and fault management can be enhanced by specifying Web services fault messages in a common way. When the information available in faults from various interfaces is consistent, it is easier for requestors to understand faults. It is also more likely that common tooling can be created to assist in the handling of faults.

The “implied resource pattern” is a convention on XML, WSDL and WS-Addressing that defines how a particular WS-Resource is referred to as a context for processing a Web service message. The term “implied” is used because the identity of the resource isn’t part of the request message, but rather is specified using the reference properties feature of WS-Addressing. The identity of the resource is implied by the context of the message and its association with an end-point reference, not directly in the signature of the message.

A Web services message that follows the implied resource pattern must be sent to a Web service referred to by an EndpointReference that follows the implied resource pattern, and must include the ReferenceProperties information from that EndpointReference, as specified by WS-Addressing.

A Web service that follows the implied resource pattern must use the ReferenceProperties information from a message that follows the implied resource pattern in order to identify the resource to associate with the execution requested by that message.

3 Extensible Markup Language (XML)

The Extensible Markup Language (XML) was originally designed as a language for defining new document formats for the World Wide Web. XML is derived from the Standard Generalized Markup Language (SGML), and can be considered to be a meta-language; a language for defining markup languages. SGML and XML are text-based formats that provide mechanisms for describing document structures using markup tags. Today XML is used for both describing document structure (e.g. HTML) and describing structured data like spread sheets, configuration files etc. The present ubiquitous presence of XML is mostly due to the fact that XML is extensible, platform-independent, and supports internationalization by being fully Unicode complaint. XML does not have a fixed vocabulary. Instead the developer can define vocabularies specific to a particular applications or domains using XML. The XML processor applications are adaptable to additive changes in the XML Structure. XML's Platform independence makes it very useful technology in achieving interoperability between different programming platforms and operating systems.

The XML 1.0 specification is a W3C Recommendation which defines the structure of an XML document by defining what "tags" and "attributes" are and how they should appear inside an XML document. The specification also defines a valid XML document and a well-formed XML document. An XML document is valid if it has an associated document type declaration and if the document complies with the constraints expressed in it. An XML document is well-formed if that document has one root element and all other elements are delimited by start and end-tags and nest properly within each other. Well formed-ness implies that XML processing software can read the document without any basic errors associated with parsing, such as invalid character data, mismatched start and end tags, multiple attributes with the same name etc. The XML specification mandates that if any well formed-ness constraint is not met, the XML parser must immediately generate a non recoverable error.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<?xml-stylesheet href="students.xsl"?>

<batch id="2003Fall">
  <student id="bm33753w">
    <first-name>John</first-name>
    <last-name>Smith</last-name>
    <address>
      <street>Woodbridge Road</street>
      <city>White Plains</city>
      <state>NY</state>
      <zip>10501</zip>
    </address>
  </student>
  <courses>
    <course id=”cs609”>
      <campus>WPGC</campus>
      <name>Operating Systems</name>
      <prof>Josh Newton</prof>
    </course>
```

```
    <course>
      <campus>WPGC</campus>
       <name>Principles of Compilers</name>
       <prof>Dennis Tidwell</prof>
    </course>
  </courses>
</batch>
```

The document begins with the optional XML declaration that specifies what version of XML is being used and character encoding used by the document. This is followed by an optional xml-stylesheet processing instruction, which is used to bind a style sheet containing formatting instructions to the XML document for use in rendering it in a more attractive manner in user applications such as Web browsers. The document type declaration if it appears MUST appear before the first element in the document. There should be only one root element or document element for every XML document. Inside the root element other elements can appear which are delimited by start and end-tags and nest properly within each other.

The term element is a technical name for the pairing for a start tag and end tag in an XML document. Every start tag must have a matching end tag and vice versa. Everything between these two tags is the content of the element. The content can be nested elements, text or comments. The element names in XML are case sensitive. Depending on the content of an element, the elements can be classified into three groups: element-only content, mixed content or empty content. Element only content consists entirely of nested elements. Mixed elements refer to any combination of text and nested elements. Empty elements contain no content. The start tag is immediately followed by an end tag. The XML document is generally viewed as a tree structure. The document element or the root element forms the root of the XML document tree. A parent element is an element which occurs immediately above, containing the element under consideration. Ancestors are all elements that occur above the given element in the hierarchy. Siblings are elements that are in the same level and have the same parent. Children elements are elements which have the given element as parent. Descendant elements are elements that have the given element as an ancestor. The start tags for an XML element can have zero or more attributes. An attribute is a name value pair where the value appears inside quotes.

3.1 XML Namespaces

The primary use of such names in XML documents is to enable identification of logical structures in documents by software modules such as query processors, stylesheet-driven rendering engines, and schema-driven validations. Consider the following example:

```
<section><title>Book-Signing Event</title>
<signing>
  <author title="Mr" name="Vikram Seth" />
  <book title="A Suitable Boy" price="$22.95" /></signing>
<signing>
  <author title="Dr" name="Oliver Sacks" />
  <book title="The Island of the Color-
Blind" price="$12.95" /></signing>
</section>
```

In this example, there are three occurrences of the name title within markup, and the name alone clearly provides insufficient information to allow correct processing by a software module. The problems to be encountered in processing this XML fragment can be classified into two: Recognition and Collision. Recognition is the problem faced by the XML processors in identifying between the section/title, signing/author title and signing/book title. Collision is the confusion faced when a software module is asked to update the attribute title. Both collision and recognition problems can be avoided by the use of XML Namespaces (http://www.w3.org/TR/REC-xml-names) which is a W3C published recommendation.

The problem of collision in composed XML documents arises because of the likelihood that elements with common names will be reused in different document types. This problem can be addressed by qualifying an XML element name with an additional identifier that is much more likely to be unique within the composed document. In XML documents conforming to the XML namespace specification, some names may be given as qualified names, defined as follows:

```
QName       ::=   (Prefix ':')? LocalPart
Prefix      ::=   NCName
LocalPart   ::=   NCName
```

The problem of recognition in composed XML documents arises because no good mechanism exists to identify all elements belonging to the same document type. Having namespace qualifiers, this problem is solved as all elements that have the same namespace identifier are considered together. For Prefixes XML namespaces uses Uniform Resource Identifiers (URIs). URI's can be locators (URLs) or names (URNs). Both URI and URN are globally unique. Both attributes and elements can have the namespaces associated with them. The example below defines two namespaces "bk" and "isbn".

```
<?xml version="1.0"?>
<!-- both namespace prefixes are available throughout -->
<bk:book xmlns:bk='urn:loc.gov:books'
         xmlns:isbn='urn:ISBN:0-395-36341-6'>
    <bk:title>Cheaper by the Dozen</bk:title>
    <isbn:number>1568491379</isbn:number>
</bk:book>
```

3.2 XML Schema

An XML schema language is used to describe the structure and content of an XML document. The W3C XML Schema Definition Language is an XML language for describing and constraining the content of XML documents. W3C XML Schema is a W3C Recommendation. The following example defines the skeleton of a schema definition.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```
    xmlns:sample="http://www.mysample.com/Sample">

  <xsd:annotation>
    <xsd:documentation xml:lang="en">
      Explaining how to define an XML schema
    </xsd:documentation>
  </xsd:annotation>

  …

</xsd:schema>
```

The schema is enclosed in the xsd:schema element. The content of this element is other schema elements that are used for element, attribute, and datatype definitions. The "<xsd:annotation>" and "<xsd:documentation>" elements can be used to attach additional information to a schema.

XML schema specification comes with a large number of predefined basic data types such as string, time, integer, date etc. A custom type can be created from base type, which can be an existing predefined schema types or the already defines simple type. This is achieved by using a <xsd:restriction> on the existing type by providing any required <xsd:pattern>. The XML simple types (<xsd:simpleType>) define the valid choices for character-based content such as attribute values and elements with character content. Complex content models, such as those of mixed elements are defined using <xsd:complexType>. Complex type definitions address the sequencing and multiplicity of child elements as well as the names of associated attributes and whether they are required or optional.

XML Schema helps web services achieve interoperability by helping define the structure and type for the XML documents used for message exchange (e.g. SOAP) and service description (e.g. WSDL).

3.3 XML Information Set

The XML Information Set (Infoset) defines a data model that describes an abstract representation of the important properties of a well-formed XML document. The data model describes the results of parsing an XML document, and it is this model that is manipulated by XML APIs. An XML document's information set consists of a number of information items. The information set of a well formed XML document will have at least a 'document' information item. An information item is an abstract description if some part of an XML document and each information item has a set of associated named properties. The current XML Information Set specification presents the information item in the form of a tree model which is represented by a document object model (DOM) interface. However the information set can also be represented by an event (SAX) based interface or query (XQuery) based interface. The XML infoset is primarily meant to act as a set of definitions used by XML technologies to formally describe what parts of an XML document they operate upon. With the specification for XML information set being available many recent W3C specifications and revisions for earlier specifications are being defined on XML Information set.

An XML document's Information Set can consist of up to eleven information items, with a minimum of one information item all of which are abstract representations of the components of an XML document. There are information items representing the document, its elements and attributes, processing instructions, unexpanded entity references, characters, comments, document type declarations, unparsed entities, notations and the namespaces. The infoset focus on capturing the data in XML document and is not too specific on the syntactical representation of the data. The infoset does not capture information like white space outside the document element, immediately following the target name of a processing instruction, with in a start tag if they are not significant and end tags. Infoset does not capture the kind of quotes used to quote an attribute value, and the specific combination used for line termination.

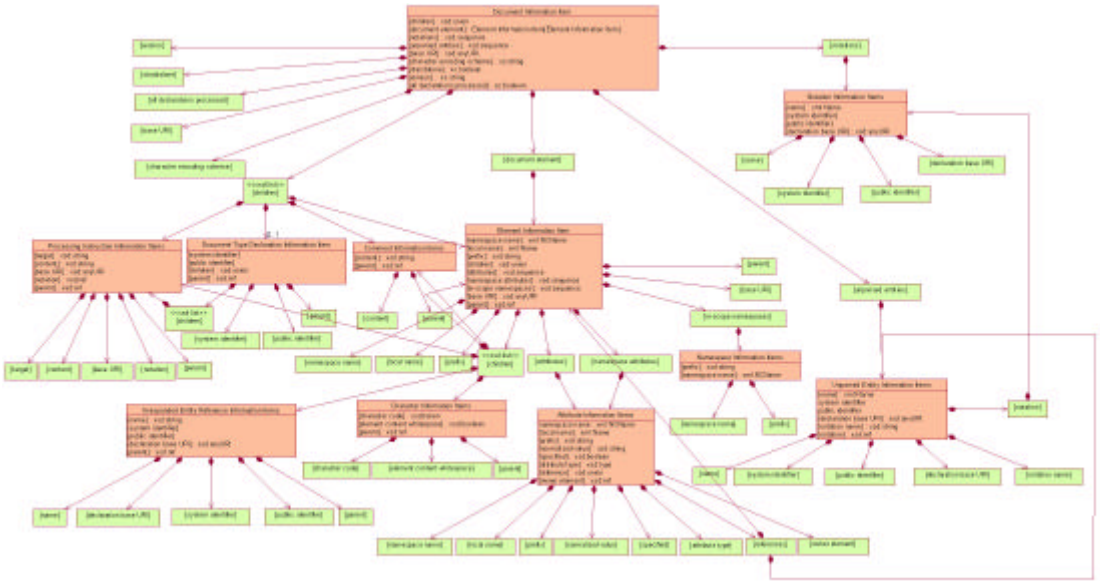

Figure 2 XML Infoset Model

Document Information Item: There is exactly one document information item in the information set and all other items are accessible from the properties of the document information item, either directly or indirectly through the properties of other information items. The document information item forms the root of the information set and has nine properties. The children properties constitute of an ordered list of child information items in document order. The list contains exactly only element information item and also a processing instruction information item and a comment information item. If there is a document declaration type associated with the XML document, then there is also a document type declaration information item in the ordered list. The document element information item corresponds to the root element of the XML document. The other properties are notations, unparsed entities, base URI, character encoding scheme, standalone, version and all declaration processed.

Element Information Item: There is an element information item corresponding to each element appearing in the XML document. One of the element information items is the

value of the document element property of the document information item. All other element information items can be accessed through the children property of the document element or root element of the XML Document. The element information item has nine properties. The namespace name, local name and the prefix properties together defines a unique name for the element information item. The children property is an ordered list of child information items, in document order. This list contains element processing instruction, unexpanded entity reference, character and comment information items, processing instruction, and reference to an unprocessed external entity, data character and comment appearing immediately within current element. If the element is empty then this has no members. The attributes property is an unordered set of attribute information items, one for each of the attributes of this element. The other properties are namespace attributes in-scope namespaces, baseURI, and parent.

Attribute Information Items: There is an attribute information item for each attribute of each element in the document, including those which are namespace declarations. The attribute information item has eight properties which include namespace name, local name, and prefix, normalized value, specified, attribute type, references and owner element.

Processing Instruction Information Items: Processing instructions allow documents to contain instructions for applications. There is a processing instruction information item for each processing instruction in the document. The processing instruction information item has five properties which are target, content, base URI, notation and parent.

Unexpanded Entity Reference Information Items: An unexpanded entity reference information item serves as a placeholder by which an XML processor can indicate that it has not expanded an external parsed entity. There is such an information item for each unexpanded reference to an external general entity within the content of an element. A validating XML processor or a non-validating processor that reads all external general entities will never generate unexpanded entity reference information items for a valid document. The unexpanded entity reference information item has the five properties, which are name, system identifier, public identifier, declaration base URI and parent.

Character Information Items: There is a character information item for each character that appears in the document, whether literally as a character reference or within a CDATA section. Each character is a logically separate information item, but XML applications are free to chunk characters into larger groups as necessary or desirable. The character information item has three properties which are character code which is the ISO 10646 character code of the character, element content white space and parent.

Comment Information Items: There is a comment information item for each XML comment in the original document, except for those appearing in the DTD, which are not represented in the information set. The comment has two properties, content which is the string representing the content of the comment and parent.

Document Type Declaration Information Item: If the XML document has a document type declaration, then the information set contains a single document type declaration information item. A document type declaration information item has the four properties, which are system identifier, public identifier, children and parent.

Unparsed Entity Information Items: There is an unparsed entity information item for each unparsed general entity declared in the DTD. An unparsed entity information item has six properties, which are name, system identifier, public identifier, declaration base URI, notation name and notation.

Notation Information Items: There is a notation information item for each notation declared in the DTD. The notation information item has four properties which are name, system identifier, public identifier and declaration base URI.

Namespace Information Items: Each element in the document has a namespace information item for each namespace that is in scope for that element. A namespace information item has two properties which are prefix and namespace name.

The following example defines an XML document and describes below its information set.

```
<?xml version=”1.0”?>
<student:name record:campus=”white plains”
            xmlns:record=”http://www.pace.edu/namespaces/record”
            xmlns:student=”http://www.pace.edu/namespaces/student”>
biju
</student:name>
```

The information set for this XML document contains the following information items:

- A document information item
- An element information item with namespace name “`http://www.pace.edu/namespaces/student`”, local part “`name`” and “`student`”
- An attribute information item with the namespace name “`http://www.pace.edu/namespaces/record`”, localpart “`campus`”, prefix “`record`” and normalized value “`white plains`”
- Three namespace information items for the “`http://www.w3.org/1998/namespace`”, “`http://www.pace.edu/namespaces/student`”, and “`http://www.pace.edu/namespaces/record`”, namespaces.
- Two attribute information items for the namespace attributes
- Four character information items for the character data.

3.4 XML Parsers

XML Parsing is a process that involves breaking the text of an XML document into small identifiable pieces (nodes). Parsers break documents into pieces such as start tags, end

tags, attribute value pairs, chunks of text content, processing instructions, comments and so on. Four parsing models are commonly used:

- *Pull parsing* – The application always has to ask the parser to give it the next piece of information about the document. It's as if the application has to "pull" the information out of the parser. Pull parsing is a recent trend in XML processing. The previously popular XML processing frameworks such as SAX and DOM were "push-based" which means the control of parsing was with the parser itself. This approach was not efficient in handing large XML documents since a complete memory model will be generated in the memory. Pull parsing inverts the control and hence the parser only proceeds at the users command. The user can decide to store or discard events generated from the parser.

- *Push parsing* – The parser sends notifications to the application about the types of XML document pieces it encounters during the parsing process. The notifications are send in the order in which the parser encounters them inside the XML document. Push parsing is also known as event based parsing. The most common API available for event parsing the SAX (Simple API for XML)

- *One-step parsing* – The parser reads the whole XML document and creates a data structure, mostly a tree, from the contents. The tree structure mimics the nesting of elements inside the XML Document. The W3C has published a DOM (Document Object Model) for XML. The XML DOM specifies the types of objects that are included in the parse tree, their properties, and their operations. Various implementation of the DOM specification is currently available. The biggest disadvantage of the DOM model is that the entire XML document is read to memory first and an object representing that document is returned, on which the available methods are called. For huge XML documents, memory availability will be a hindrance for performance.

- *Hybrid Parsing* – The hybrid parsing combines the best features of the other parsing techniques to provide a better performing parser. One example is combining the SAX parser with the pull parser. The application will get the impression that the whole document is in memory, but in effect, the document gets loaded into memory only on demand.

4 Web Services Description Language (WSDL)

Service description is a key feature within a service-oriented architecture. A service description is involved in each of the three operations of SOA: publish, find and bind. The service provider during the publish operation publishes the service description to the service registry. The service requestor searches the service descriptions available in the service registry during the find operation to identify a matching service. The service description defines the message format expected by the service provider so that service requestor can send the messages in this manner during the bind operation.

Web Services Description language (WSDL) is used to describe the syntax associated with the invocation and response of a Web Service. The WSDL standard is available from the W3C. A WSDL service description is a valid XML document conformant to the WSDL schema definition. WSDL defines answer to questions like what a service does, how a service is accessed, where a service is located etc.

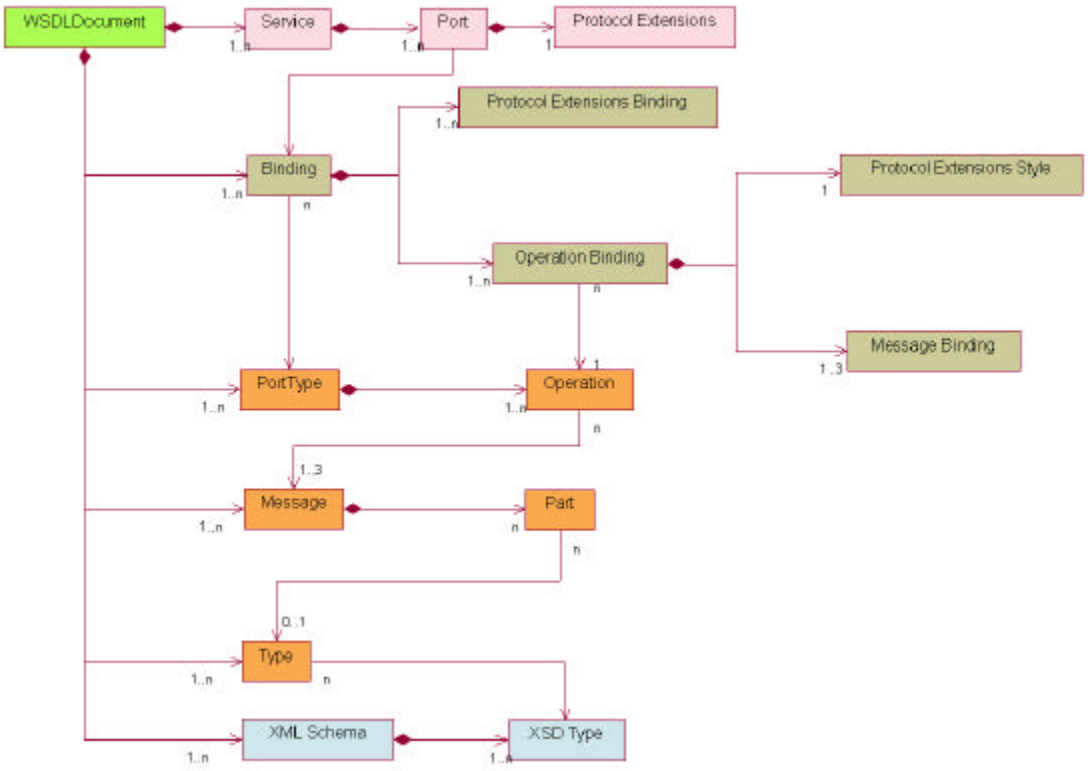

Figure 3 WSDL Information Model

A WSDL description has the following elements:
portType: A web service's abstract interface definition where each child operation element defines an abstract method signature.

message: Message element defines the format of the message, or a set of parameters, referred to by the method signatures or operations. A message is composed of parts.
types: The collection of all the datatypes used in the web services as referenced by the various message part elements is defined by the type element.
binding: Binding describes the implementation details of how the elements in a portType are converted into a concrete implementation in a particular combination of data formats and protocols.
port: The port element describes how a binding is deployed at a particular network endpoint
service: A service is a collection of concrete implementation endpoints of web services.

The WSDL file shown in the example below defines the StockQuoteService. This service has one operation available called getStockQuote(). The individual parts of this WSDL file is described in detail below.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="stockQuote"
      targetNamespace="http://www.stockquotes.org/quote"
      xmlns:impl="http://www.stockquotes.org/quote"
      xmlns:intf="http://www.stockquotes.org/quote"
      xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
      xmlns="http://schemas.xmlsoap.org/wsdl/">

   <wsdl:message name="getQuoteResponse">
      <wsdl:part name="getQuoteReturn"
             type="xsd:float"/>
   </wsdl:message>

   <wsdl:message name="getQuoteRequest">
      <wsdl:part name="symbol" type="xsd:string"/>
   </wsdl:message>

   <wsdl:portType name="StockQuote">
      <wsdl:operation name="getQuote" parameterOrder="symbol">
         <wsdl:input name="getQuoteRequest"
                message="impl:getQuoteRequest"/>
         <wsdl:output name="getQuoteResponse"
                 message="impl:getQuoteResponse"/>
      </wsdl:operation>
   </wsdl:portType>

   <wsdl:binding name="quotesSoapBinding"
             type="impl:StockQuote">
      <soap:binding style="document"
                 transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="getQuote">
         <soap:operation soapAction=""/>
         <wsdl:input name="getQuoteRequest">
            <soap:body use="literal"/>
         </wsdl:input>
         <wsdl:output name="getQuoteResponse">
```

```
            <soap:body use="literal"/>
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>

   <wsdl:service name="StockQuoteService">
      <wsdl:port name="quotes" binding="impl:quotesSoapBinding">
         <soap:address location="http://www.stockquote.com/quotes"/>
      </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```

The root element of a WSDL document is the definitions element which in turn contains all the other elements of the WSDL document. A definitions element can contain the following elements:

- Zero or more `documentation` elements
- Zero or more `import` elements
- Zero or one `types` element
- Zero or more `message` elements
- Zero or more `portType` elements
- Zero or more `binding` elements
- Zero or more `service` elements

The definitions element contains a name attribute which is typically the name of the web services. The definitions element also contains the target namespace and other namespace definitions which are used in the WSDL document. The definitions section of the stockQuote WSDL is shown below

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="stockQuote"
      targetNamespace="http://www.stockquotes.org/quote"
      xmlns:impl="http://www.stockquotes.org/quote"
      xmlns:intf="http://www.stockquotes.org/quote"
      xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
      xmlns="http://schemas.xmlsoap.org/wsdl/">
```

The portType element defines the interface to a Web service, which describes the operations available. The portType description of the stockQuote WSDL is shown below

```
   <wsdl:portType name="StockQuote">
      <wsdl:operation name="getQuote" parameterOrder="name">
         <wsdl:input name="getQuoteRequest"
                 message="impl:getQuoteRequest"/>
         <wsdl:output name="getQuoteResponse"
                  message="impl:getQuoteResponse"/>
      </wsdl:operation>
   </wsdl:portType>
```

The stockQuote portType defines one operation, getQuote. There can be zero or more portTypes defined in a WSDL file. Each portType's name attribute should have a different name.

An operation defines a method on a Web service, including the name of the method and the input parameters and the output parameters of the method. The StockQuote portType has one operation named getQuote. The getQuote operation takes as input parameter impl:getQuoteRequest and returns impl:getQuoteResponse.

A WSDL document can contain zero or more message elements. Each message has a name, which must be unique within the WSDL document and contains a collection of part elements. The message definitions of the stockQuote WSDL is shown below.

```
<wsdl:message name="getQuoteResponse">
   <wsdl:part name="getQuoteReturn"
          type="xsd:float"/>
</wsdl:message>

<wsdl:message name="getQuoteRequest">
   <wsdl:part name="name" type="xsd:string"/>
</wsdl:message>
```

The message getQuoteResponse wraps a xsd:float type which will contain the value of the returned stock quote. The message getQuoteRequest wraps an xsd:string which contains the value of the stock quote symbol which is send to the web service as an input parameter.
A part element is made up of two properties. The name property describes the name of the part and it should be unique inside the WSDL document. The kind property describes type which is an XSD type or element which refers to an element defined in the XML Schema. In the stockQuote example both the parts are defined as types, and they refer to the schema definition type of string and float.

The binding element in the WSDL document describes how to format the message in protocol specific manner. Each portType can have one or more binding elements associated with it. The binding element for the stockQuote service is described below.

```
<wsdl:binding name="quotesSoapBinding"
          type="impl:StockQuote">
   <soap:binding style="document"
               transport="http://schemas.xmlsoap.org/soap/http"/>
   <wsdl:operation name="getQuote">
      <soap:operation soapAction=""/>
      <wsdl:input name="getQuoteRequest">
         <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="getQuoteResponse">
         <soap:body use="literal"/>
      </wsdl:output>
   </wsdl:operation>
</wsdl:binding>
```

The port element in WSDL specifies the network address of the endpoint hosting the Web Service. The port associates a single protocol specific address to an individual binding. The port element does not stand alone. The port elements are children of the WSDL service element.

The service element defines the collection of ports available for the service described in the WSDL document. The service element contains one entry each for every concrete implementation end point of the service defined in the WSDL document.

```
<wsdl:service name="StockQuoteService">
   <wsdl:port name="quotes" binding="impl:quotesSoapBinding">
      <soap:address location="http://www.stockquote.com/quotes"/>
   </wsdl:port>
</wsdl:service>
```

4.1 WSDL 2.0 (Working Draft)

The WSDL specification is being standardized within the W3C working group. The revision for a new draft for WSDL called WSDL 2.0 is under progress in W3C. Some of the major changes in WSDL 2.0, from WSDL 1.1 are described below. The WSDL 2.0 specification is defined by three related specifications. They are Core Language, Message Patterns, and Bindings. The portType element is being replaced by interface. There is a new attribute called style for the operations element which defines the type of message exchange pattern for the operation. The abstract message is now defined using types and elements. WSDL 1.1 message and part elements are removed. Also overloading of methods in removed. The name attribute of the WSDl definition element is removed. The stockQuote WSDL file defined above is described in WSDL 2.0 below.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
      targetNamespace="http://www.stockquotes.org/quote"
      xmlns:impl="http://www.stockquotes.org/quote"
      xmlns:intf="http://www.stockquotes.org/quote"
      xmlns:soap=" http://www.w3.org/2003/11/wsdl/soap12"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:wsdl2="http://www.w3.org/2003/11/wsdl"
      xmlns="http://www.w3.org/2003/11/wsdl/>

   <!–Types definition ->
   <types>
      <xsd:schema
         targetNamespace=”http://www.stockquotes.org/quote"
         xmlns:xsd=”http://www.w3.org/2001/XMLSchema”>

      <xsd:element name=”symbol” type=”xsd:string” />
      <xsd:element name=”quote” type=”xsd:float” />

      <xsd:schema/>
   </types>

   <interface name="StockQuote">
      <operation name="getQuote" parameterOrder="symbol">
         <input message="impl:symbol"/>
         <output message="impl:quote"/>
      </operation>
   </interface>

   <binding name="quotesSoapBinding" type="impl:StockQuote">
```

```
      <soap:binding
      protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/"/>
   </binding>

   <service name="StockQuoteService"
            interface=”impl:StockQuoteService”>
      <endpoint name="quotes" binding="impl:quotesSoapBinding">
         <soap:address location="http://www.stockquote.com/quotes"/>
      </endpoint>
   </service>

</definitions>
```

5 Simple Object Access Protocol (SOAP)

5.1 What is SOAP?

SOAP provides a simple and lightweight mechanism for exchanging structured and typed information between peers in a decentralized, distributed environment using XML. A SOAP message is an XML document information item that contains three elements: **<Envelope>**, **<Header>**, and **<Body>**. All Web services messages are SOAP messages that take full advantage of the XML Information Set. Any software agent that sends or receives messages is called a SOAP node. The node that performs the initial transmission of a message is called the original sender. The final node that consumes and processes the message is called the ultimate receiver. Any node that processes the message between the original sender and ultimate receiver is called an intermediary. Intermediaries are used to model the distributed processing of an individual message. The collection of intermediary nodes traversed by the message and the ultimate receiver are collectively referred to as the message path.

Simple Object Access Protocol (SOAP) is a W3C reccomendation which provides the definition for the XML-based information which can be used for exchanging structured and typed information between peers in a decentralized, distributed environment. SOAP uses XML technologies to define an extensible messaging framework, which provides a message construct that can be exchanged over a variety of underlying protocols. This framework has been designed to be independent of any particular programming model and other implementation specific semantics. SOAP is fundamentally a stateless, one-way message exchange paradigm, which can be used as a building block for creating more complex interaction patterns (e.g., one-way, request/response, notification, notification/responses, etc.) by combining one-way exchanges with features provided by an underlying protocol and/or application-specific information. SOAP does not dictate the semantics of any application-specific data it conveys, such as the routing of SOAP messages, reliable data transfer, firewall traversal, etc. However, SOAP provides the framework by which application-specific information may be conveyed in an extensible manner. Also, SOAP provides a full description of the required actions to be taken by a SOAP processor node on receiving a SOAP message.

SOAP defines a simple extensible way to move XML messages from a sender to a receiever. This is achieved by providing an XML-based messaging framework that is extensible and usable over a variety of networking protocols and independent of programming models. Simplicity is one of primary design goals for SOAP. SOAP provides a structure to contain XML messages. The additional features of a distributed interoperable webservices application like security, routing, reliabilty etc can be added as layered extensions of SOAP. SOAP can be used over any transport protocol like HTTP, SMTP, TCP, UDP etc. Inorder to maintain interoperability, standard protocol bindings need to be specified that outline the rules for each environment. The SOAP specification provides a flexible framework for defining arbitrary protocol bindings and provides an explicit binding for todays most widely used HTTP protocol. The SOAP processing

model is designed to allow extensibility and versioning. The mustUnderstand attribute controls whether the introduction of a new header block is a breaking or non-breaking change. Adding optional headers blocks (e.g., headers marked mustUnderstand="false") is a non-breaking change, as any SOAP node is free to ignore it. Adding mandatory headers blocks (e.g., headers marked mustUnderstand="true") is a breaking change, in that only SOAP nodes that are aware of the header block's syntax and semantics are able to process the message. The role and relay attributes compose with mustUnderstand to distribute this processing model along a message path.

W3C SOAP specification consists of three parts. The SOAP 1.2 Part 0 Primer is a non-normative description on the document intended to provide an easily understandable tutorial on the features of the SOAP Version 1.2 specifications. Primer's purpose is to help a technically competent person understand how SOAP may be used, by describing representative SOAP message structures and message exchange patterns. In particular, Primer describes the features through various usage scenarios, and is intended to complement the normative text contained in Part 1 and Part 2 of the SOAP 1.2 specifications. The SOAP 1.2 Part 1 Messaging Framework is a normative definition of the SOAP messaging framework. SOAP specification part 1 defines the SOAP envelope, which is a construct that defines an overall framework for representing the contents of a SOAP message, identifying who should deal with all or part of it, and whether handling such parts are optional or mandatory. It also defines a protocol binding framework, which describes how the specification for a binding of SOAP onto another underlying protocol may be written. The SOAP 1.2 Part 2 Adjuncts is a normative definition of the set of adjuncts that can be used in connection with the SOAP messaging framework. SOAP Part2 defines a data model for SOAP, a particular encoding scheme for data types which may be used for conveying remote procedure calls (RPC), as well as one concrete realization of the underlying protocol binding framework defined in SOAP Part 1. This binding allows the exchange of SOAP messages either as payload of a HTTP POST request and response, or as a SOAP message in the response to a HTTP GET.

The structure of a SOAP envelope is described in the example skeleton below.

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  <env:NotUnderstood …/>
  …
 </env:Header>
 <env:Body>
  …
    <env:Fault>
      …
      <env:Code>
        <env:Value> … </env:Value>
        <env:Subcode> … </env:Subcode>
      </env:Code>
      <env:Reason> … </env:Reason>
      <env:Detail> … </env:Detail>
      <env:Node> … </env:Node>
      <env:Role> … </env:Role>
    </env:Fault>
```

```
 </env:Body>
</env:Envelope>
```

The SOAP specification defines two soap specific child elements inside the SOAP Envelope, `env:Envelope`. These are the header element `env:Header` and the body element `env:Body`. The contents of these elements are defined by the applications using them and not by the SOAP specifications. The specification marks the `env:Header` element as optional. The `env:Header` element is used as an extension mechanism to provide information to the SOAP processing nodes, which are also called soap intermediaries, about how the `env:Envelope` should be processed. The `env:Body` element is a required element in the SOAP envelope. The application specific content is contained in the `env:Body` element and is understood by the sender and the receiver nodes. The `env:Body` element may also contain the optional SOAP Fault element `env:Fault`. The `env:Fault` element provides extensive provisions to convey error situations that might arise from the soap specific and application specific processing of the SOAP message. The `env:Fault` element contains two mandatory child elements `env:Code` and `env:Reason`. In addition `env:Fault` element may also contain `env:Detail, env:Node` and `env:Role` elements as optional sub elements. The soap specific error information is contained in the `env:Code` and `env:Reason` sub elements while the `env:Detail` element will contain the application specific information. The `env:Node` element provides the URI of the node which created the error and the `env:Role` element provides the role being played by the node which created the fault. The `env:Code` sub element is itself comprised of a mandatory `env:Value` sub element and an optional `env:Subcode` sub element. The SOAP encodingStyle attribute is used to define the data types used in the document. This attribute may appear on any SOAP element, and it will apply to that element's contents and all child elements. A SOAP message has no default encoding.

The body of a SOAP envelope is always targeted at the ultimate receiver. In contrast, SOAP headers may be targeted at intermediaries or the ultimate receiver. To provide a safe and versioned model for processing messages, SOAP defines three attributes that control how intermediaries and the ultimate receiver process a given header block—**role**, **relay**, and **mustUnderstand**. The role attribute is used to identify which node the header block is targeted at. The mustUnderstand attribute indicates whether that node may ignore the header block if it is not recognized. Header blocks marked mustUnderstand="true" are called mandatory header blocks. Header blocks marked mustUnderstand="false" or that have no mustUnderstand attribute are called optional header blocks. The relay attribute indicates whether that node should forward unrecognized optional headers or discard them.

Every SOAP node must use these three attributes to implement the SOAP processing model. The following steps define that model:

1. Identify all header blocks of the SOAP message intended for the current SOAP node using the role attribute (the absence of this attribute implies the header block is for the ultimate receiver).

2. Verify that all mandatory header blocks identified in Step 1 can be processed by the current SOAP node using the SOAP mustUnderstand attribute. If a mandatory header block cannot be processed by the current SOAP node, the message must be discarded and a distinguished fault message must be generated.
3. Process the message. Optional message elements may be ignored.
4. If the SOAP node is not the ultimate receiver of the message, all header blocks identified in Step 1 that are not relay-able are removed and the message is then relayed to the next SOAP node in the message path. The SOAP node is free to insert new header blocks into the relayed message. Some of these header blocks may be copies of header blocks identified in Step 1.

The following figure gives the pictorial representation of the SOAP envelope.

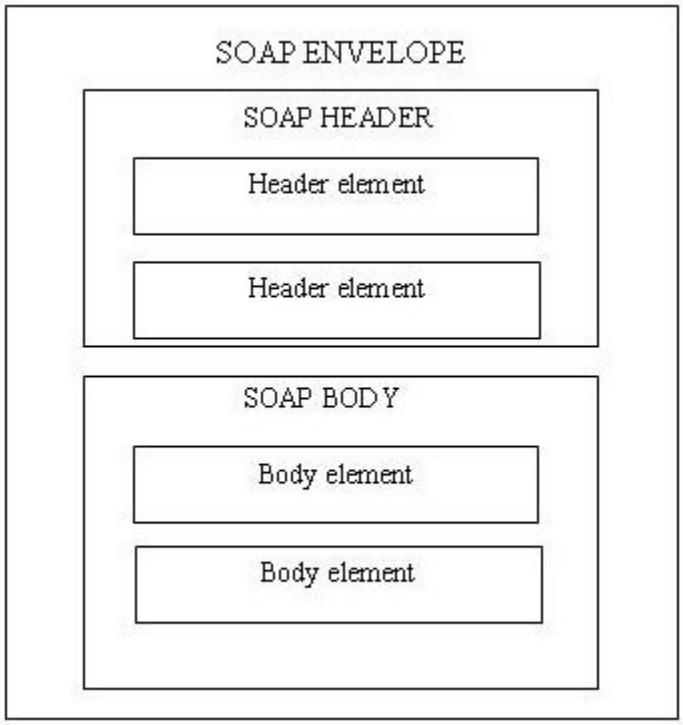

Figure 4 SOAP Envelope

The examples below show how the SOAP envelope is populated with different information.

This SOAP envelope describes the information request for course information for a student.

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
<env:Body>
  <p:courseInfoRequest
    xmlns:p="http://www.pace.edu/student ">
   <p:name>Biju</p:name>
```

```
   <p:studentId>bm33753w</p:studentId>
  </p:courseInfoRequest>
 </env:Body>
</env:Envelope>
```

This SOAP envelope describes the response message for course information request.

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
<env:Body>
  <p:courseInfoResponse
    xmlns:p="http://www.pace.edu/sars/grade">
   <p:course>
     <p:courseCode>CS 605</p:courseCode>
     <p:courseName>Compilers</p:courseName>
     <p:courseDate>2004 Fall</p:courseDate>
     <p:courseCampus>White Plains</p:courseCampus>
     <p:courseProfessor>M Badii</p:courseProfessor>
   </p:course>
   <p:grade>
     <p:gradeCode>A</p:gradeCode>
     <p:gradeType>Excellent</p:gradeType>
   </p:grade>
  </p:courseInfoResponse>
 </env:Body>
</env:Envelope>
```

This SOAP envelope describes how SOAP Fault element will capture the exceptions which occurred during the SOAP processing.

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env=”http://www.w3.org/2003/05/soap-envelope”
              xmlns:rpc=”http://www.w3.org/2003/05/soap-rpc” >
<env:Body>
   <env:Fault>
     <env:Code>
       <env:Value>env:Sender</env:Value>
       <env:Subcode>
        <env:Value>rpc:BadArguments</env:Value>
       </env:Subcode>
     </env:Code>
     <env:Reason>
      <env:Text xml:lang="en-US">Processing error</env:Text>
     </env:Reason>
     <env:Detail>
      <e:myFaultDetails
        xmlns:e="http://www.pace.edu/sars/faults">
        <e:message>No student by this studentId exists</e:message>
        <e:errorcode>404</e:errorcode>
      </e:myFaultDetails>
     </env:Detail>
   </env:Fault>
</env:Body>
</env:Envelope>
```

This SOAP envelope describes how mustUnderstand attribute is specified inside the header element of a SOAP envelope.

```
<?xml version="1.0" ?>
 <env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
   <env:Header>
     <a:log xmlns:a="http://www.pace.edu"
           env:role="http://www.pace.edu/Log"
               env:mustUnderstand="true">
     ...
     ...
     </a:log>
     <b:authenticate xmlns:b="http://www.pace.edu/Authenticate"
       env:role="http://www.w3.org/2003/05/soap-envelope/role/next">
     ...
     ...
     </b:authenticate>
     <c:charge xmlns:c="http://www.pace.com/Charge">
     ...
     ...
     </c:charge>
   </env:Header>
   <env:Body >
     ...
     ...
   </env:Body>
 </env:Envelope>
 </env:Body>
```

SOAP defines a processing model that outlines rules for processing a SOAP message as it travels from a SOAP sender to a SOAP receiver. The processing model, allows for architectures as shown in Figure 4, which contains an initial sender node, multiple intermediary nodes, and a final receiver node.

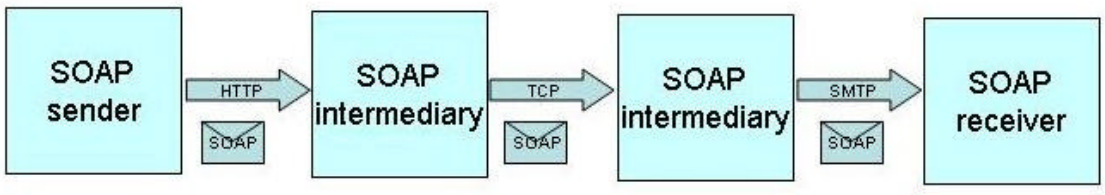

Figure 5 SOAP Processing Model

An intermediary sits between the initial sender and the ultimate receiver and intercepts SOAP messages. An intermediary acts as both a SOAP sender and a SOAP receiver at the same time. While processing a message, a SOAP node assumes one or more roles that influence how SOAP headers are processed. Roles are given unique names (in the form of URIs) so they can be identified during processing. When a SOAP node receives a message for processing, it must first determine what roles it will assume. It may inspect the SOAP message to help make this determination.

Once it determines the roles in which it will act, the SOAP node must then process all mandatory headers (marked mustUnderstand="1") targeted at one of its roles. The SOAP node may also choose to process any optional headers (marked mustUnderstand="0") targeted at one of its roles.

SOAP 1.1 only defines a single role named http://schemas.xmlsoap.org/soap/actor/next (next, for short). Every SOAP node is required to assume the next role. Hence, when a SOAP message arrives at any given SOAP node, the node must process all mandatory headers targeted at the next role, and it may choose to process optional headers also targeted at the next role. If the SOAP node wasn't designed to understand a mandatory header targeted at one of its role, it is required to generate a SOAP fault, with a soap:MustUnderstand status code, and discontinue processing. The SOAP Fault element provides the faultactor child element to specify who caused the fault to happen within the message path. The value of the faultactor attribute is a URI that identifies the SOAP node that caused the fault. If a SOAP node successfully processes a header, it's required to remove the header from the message. SOAP nodes are allowed to reinsert headers, but doing so changes the contract parties—it's now between the current node and the next node the header targets. If the SOAP node happens to be the ultimate receiver, it must also process the SOAP body.

There are here are two fundamental styles of SOAP messaging which are document style and RPC style. Document style indicates that the SOAP body simply contains an XML document whose format the sender and the receiver must agree upon. RPC style, on the other hand, indicates that the body contains an XML representation of a method call according to the SOAP encoding rules.

There are also two techniques for deciding how to serialize the data into the body: using literal XML Schema definitions and using the SOAP encoding rules. With the former approach, the schema definition literally defines the XML format for the body without ambiguity. With the latter approach, however, the SOAP processor must walk through the various SOAP encoding rules at runtime to determine the proper serialization of the body. This technique is obviously more prone to errors and interoperability issues. It's most common to use document style with literal schema definitions (known as document/literal) and RPC style with the SOAP encoding rules (known as rpc/encoded). Document/encoded and rpc/literal are possible but they're not common and don't make much sense.

6 SOAP Binary XML and Opaque data

The success of XML is evident from its global acceptance for application integration. The primary reason for the popularity of XML is its flexibility and simplicity from textual encoding. Web service which is the current technology for interoperability and distributed computing across multiple platforms is based on XML. XML was introduced as a structured data representation of textual data. As the usage of XML increased, there was demand for representing other data formats in XML. However the serialization of any structured data into XML does not make sense. Media data and data that include digital signatures are two examples. The media data formats are highly standardized formats with years of work behind them. They are also highly compressed formats. Changing the media formats to XML representation will make them loose the compression efficiency and additional applications has to be developed which can read the new formats, which may not happen. For digital signatures the binary integrity won't be retained if the structure of the data is changed. This sort of binary data is often referred to as opaque data which presents an inherent problem in the web services world. The web services infrastructure is standardized based on SOAP messaging, and sending the opaque data within the SOAP envelope is not efficient for this infrastructures working.

The opaque data is structurally not textual and hence does not seamlessly coexist with the XML structure. However traditionally two techniques are used for dealing with opaque data in XML. These are the "by value" and "by data" techniques. The "by value" method is achieved by embedding opaque data as an element or attributes content. XML supports embedding of opaque content through the use of either base64 or hexadecimal text encoding, which is specified by the XML schemas binary representation types, xs:base64Binary and xs:hexBinary. The representation of the xs:hexBinary is a simple hexadecimal character sequence and that of xs:base64Binary is by use of the base64 algorithm. An ordered sequence of octets is the final representation of both the types. The example below, the frontpage, key and hash elements, each contain a base64 string, which is the representation of the octet sequence for the same data which is provided below.

```
<m:data xmlns:m='http://sample.org/book' >
  <frontpage>/aWKKapGGyQ=</frontpage>
  <key>sdcfo2JTiXE=</key>
  <hash>Faa7vROi2VQ=</hash>
</m:data>
```

```
fd a5 8a 29 aa 46 1b 24 (frontpage)
b1 d7 1f a3 62 53 89 71 (key)
15 a6 bb bd 13 a2 d9 54 (hash)
```

Using base64 encoding, binary data can be textually encoded into an XML document. However the biggest disadvantage of using base64 encoding is that base64 encoded data expands by a factor of 1.33x original size, and that hexadecimal encoded data expands by a factor of 2x (assuming an underlying UTF-8 text encoding in both cases; if the

underlying text encoding is UTF-16, these numbers double). Also of concern is the overhead in processing costs (both real and perceived) for these formats, especially when decoding back into raw binary. These performance concerns have discouraged many developers from using embedded data in XML.

The second technique for dealing with binary data in XML document is “by reference”. XML 1.0 explicitly supports referencing external opaque data as external unparsed general entities. This is a fairly unused feature of XML and hence unparsed entities are not widely used. The primary obstacle to using unparsed entities is their heavy reliance on DTDs, which reduces modularity as well as use of XML namespaces. They are also not available to SOAP, which explicitly prohibits document type declarations in messages. A more common way to reference external opaque data is to simply use a URI as an element or attribute value. XML Schema supports this explicitly through the xs:anyURI type.

```
<?xml version="1.0" ?>
<data>
  <frontpage data="http://example.org/cover.jpg" />
  <key data="http://example.org/key.id" />
  <hash data="http://example.org/my.hsh" />
</data>
```

An XML schema can describe the content of the data attribute:

```
<xs:attribute name="data" type="xs:anyURI" use="required" />
```

Referencing opaque data avoids some of the performance and bloat issues associated with base64/hex encoding, but introduces its own problem; because the data is external to the document, it isn't part of the message’s information set.

Both “by value” and “by reference” methods have advantages and disadvantages making neither of them a favorite choice for dealing with binary data in XML document. The “by value” method seems to be potentially more appropriate as it treats the whole data as single information set, though it bloats the data.

With the development of the XML technology, the requirement for having a mechanism for transporting binary data developed new technologies which are discussed below:
Base64 encoding: With XML representation, replace any binary data with a structured XML representation (e.g. Base64 encoding) and include it inside the message. The deficiency of this method is the bloating of data while representing binary data in hexadecimal or base64 encoding.

SOAP with Attachments (SwA): “SOAP with Attachments” method sends binary data with your SOAP message in a MIME multipart message. There is no encoding and decoding issues associated, as data is sent in raw binary format. However there are potential issues with buffering large incoming binary data segments.

WS-Attachments with DIME: WS-Attachments and DIME is a faster and more efficient solutions for binary data attachment compared to SOAP with Attachments solution.

Message Transmission and Optimization Mechanism (MTOM): The MTOM mechanism incorporates the advantage of a SOAP infrastructure with the transport efficiencies provided by the SwA solution.

The current standard solution for the opaque data problem with Web services is the SOAP Message Transmission Optimization mechanism (MTOM) which is available as a candidate recommendation from the W3C standards organization. MTOM treats the opaque data part as part of the SOAP envelope at the Infoset level, but serializes the data for efficient transport over MIME messages.

6.1 Base64 Encoding

The base64 encoding scheme was originally devised to reliably transmit eight-bit data through transmission systems constrained to handle only seven-bit data. The base64 encoding is defined in RFC 1521. The RFC definition says the base64 as a message representation protocol which specifies considerable detail about message headers, but which leaves the message content, or message body, as flat ASCII text. This document defines the format of message bodies to allow multi-part textual and non-textual message bodies to be represented and exchanged without loss of information.

The base64 encoding takes the eight-bit data values and map them into a 65 character subset of the US-ASCII code enabling subgroups of 6 bits each to be represented by 64 different printable characters. The extra 65 character "=" is a special character called the pad character when the number of input bytes is not evenly divisible by three and therefore doesn't produce a number of output characters that is evenly divisible by four. Special processing is performed if fewer than 24 bits are available at the end of data being encoded. A full encoding quantum is always completed at the end of a body. When fewer than 24 input bits are available in am input group, zero bits are added (on the right) to form an integral number of 6-bit groups. Padding at the end of the data is performed using the "=" character.

The base64 encoding process causes 24-bit groups, each representing three eight-bit data values, to be represented as output groups of four encoded characters that are derived from a base64 alphabet. Proceeding from let to right, a 24-bit input group is formed by concatenating three 8-bit input groups. These 24 bits are then treated as 4 concatenated 6-bit groups, each of which is translated into a single character in the base64 alphabet. The input bit stream must be ordered with the most significant bit first. The first bit in the stream must be the high order bit in the first byte, and the eighth bit must be the low order bit in the first byte.

The table below shows the base64 alphabet. Whenever the value of a six-bit group matches one of the values in the Value columns in the table, that value is replaced by the seven-bit ASCII value of the corresponding character shown in the Char column to the right of the Value column.

Value	Character	Value	Character	Value	Character	Value	Character
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v	pad	=
14	O	31	F	48	W		
15	P	32	G	49	X		
16	Q	33	h	50	y		

Figure 6 The Base64 alphabet

7 SOAP Attachments History

Technologies for web services attachments deal with binary data associated with an XML Information set. The first technology which came into existence for the binary data attachment problem was the SOAP Messages with Attachment (SwA). SOAP Messages with Attachment being the first available solution addressing the binary attachment problem had a wide range of support. Though this was not a complete solution for the attachment problem, the SwA specification was successful in creating the foundation for supporting web services attachment based on SOAP and MIME. Following SwA, other solutions came into existence which tried to solve the deficiencies shown by SwA. The second major approach for the web services attachment problem was the DIME and WS-Attachment specification proposed together by Microsoft and IBM. The primary focus of this specification was to fix the performance problem of SOAP Messages with Attachment by proposing a totally new packaging mechanism for web services attachments. However the DIME and WS-Attachments specification was not able to gather enough support and faded away, with Microsoft not supporting DIME anymore. Being the first available specification SwA had reasonable support from the XML web services community. However the clarity issues resulting from the SwA and WSDL 1.1 specification resulted in lot of interoperability issues among the SOAP Processing Engines from different vendors. In order to address this issue, the WS-I (Web Services Interoperability) working group started working on the WS-I Attachment profile. The proposed Infoset Addendum to SOAP Messages with Attachments (PASwA) was constituted to define a consistent SOAP processing model for web services attachments by introducing an XML Infoset-based common logical view of the whole data set. The Infoset-based approach introduced by PASwA gained support form the industry, and eventually resulted in the current W3C Candidate Recommendation on XML-Binary optimized packaging (XOP), SOAP Message Transmission Optimization Mechanism (MTOM) and the related specifications.

Many web services have common application requirement that they carry binary data along with the XML document data. This combined data set have different requirements than simple XML documents for processing by applications or system level services, and for transmitting from one service to another service. The two main requirements are an infoset view of the complete document and packaging and transmitting the data without loosing relationship information.

7.1 SOAP with Attachments (SwA)

SOAP with Attachments was the first attempt at solving the data attachment problem. This was submitted by Microsoft and HP. SwA defines an extension to SOAP1.1 transport binding mechanism. The extension allows a SOAP1.1 message to be carried within a MIME multipart/related package without having to change the SOAP1.1 processing rules. In SwA, the combined data set was packaged as a MIME multipart/related package, and the root part of this package must be a SOAP 1.1 envelope. Related binary data and other textual data may be packaged as non-root MIME parts. The whole MIME package is transmitted through HTTP. The root part soap message may reference other MIME part using a href attribute defined by SOAP Encoding. The href

attribute value is a URI pointing to a content-id or current-location of a MIME part or the URL of some external web source. The SwA specification only addresses how to package and transmit a SwA message at runtime. It does not fully define how to describe the SwA message in WSDL. The commonly used description mechanism for SwA attachments is WSDL 1.1 MIME binding. Using the WSDL 1.1 approach, the SOAP body and the attachments are defined first as an XML schema element or type, associated to a message part of a WSDL portType, and then bound to a MIME binding. The MIME binding indicated which abstract message parts go where in the MIME package and provides media types for message parts.

The main focus of SwA was to package and transmit a SOAP envelope with binary attachments using MIME over HTTP. SwA had a couple of problems, one of which was related to the use of a text string to delineate boundaries between multiple parts. This required reading the whole document to memory and then scanning for the delimiter string to find the boundary. The second problem with SwA was the non availability of a single view of the combined data set which describes the type of the binary data along with the XML data.

7.1.1 SOAP with Attachments API for Java (SAAJ)

SAAJ provides a convenient API for constructing SOAP messages, in a generic way and to send and receive SOAP messages from one node to another across a network. SAAJ was originally part of the Java API for XML Messaging (JAXM), which was developed by the JSR 67 expert group. SAAJ was later separated into a separate API. SAAJ is a required component of J2EE (Java 2 Platform, Enterprise Edition) 1.4. SAAJ defines the namespace `javax.xml.soap`.

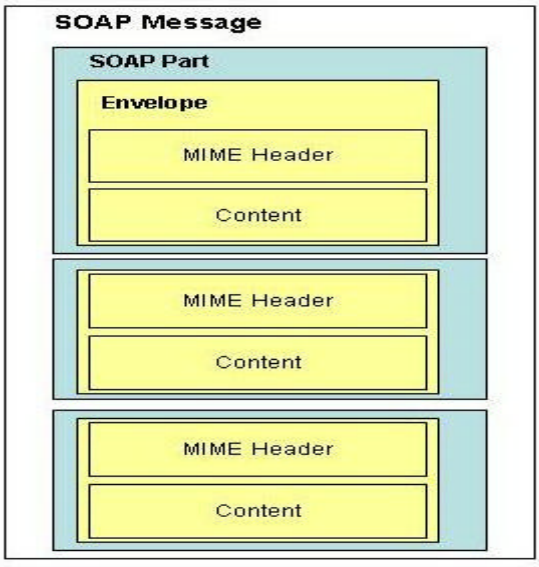

Figure 7 SOAP Message with 2 attachments

SOAP's attachment feature extends a SOAP message to include, in addition to the regular SOAP part, zero or more attachments, as above figure shows. Each attachment is defined by a MIME type and can assume any content represented as a byte stream.

The W3C Note on SOAP messages with attachment does not define any new structure; rather it defines how to take advantage of the MIME type to add multiple attachments to the SOAP message structure.

A SOAP message with attachments will be represented on the wire as shown in the example below

```
POST /studentRecord HTTP/1.1
Host: www.pace.edu
Content-Type: Multipart/Related; boundary=MIME_boundary; type=text/xml;
start="<studentRecord@pace.edu>"
Content-Length: 12FG

----MIME_boundary
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: <studentRecord@pace.edu>

<?xml version='1.0'?>
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<pace:studentRecord id=”bm37553w”
      xmlns:pace=”http://www.pace.edu”>
 <name>John Smith</name>
 <address>
    <street>8, Main St </street>
    <city>White Plains </city>
    <state>NY</state>
    <zip>10601</zip>
  </address>
  <photo href=”bm37553w.jpg@pace.edu”/>
</pace:studentRecord>
</SOAP-ENV: Body>
</SOAP-ENV: Envelope>

----MIME_boundary
Content-Type: image/jpeg
Content-ID:<bm37553w.jpg@pace.edu>

… octet stream …
----MIME_boundary
```

The above multipart message contains a series of MIME-headers and related data. At the root of the document is the SOAP body. The SOAP body contains only XML data and the MIME type of the entire message is text/xml. Following the SOAP envelope is the image attachment. A content ID identifies each attachment which is used to reference the attachments.

SAAJ API lets create and edit any part of a SOAP message. The SAAJ API is build on top of the XML DOM (Document Object Model) API for XML processing. The base class of all constructs of the SAAJ is the javax.xml.soap.Node interface which extends the org.w3c.dom.Node interface. The SAAJ API provides a factory pattern for creating a SOAP message. With the SAAJ API, you can create XML messages that conform to the SOAP 1.1 and WS-I Basic Profile 1.0 specifications simply by making Java API calls.

SAAJ API provides a factory pattern to create a SOAP message. Create an instance of `javax.xml.soap.MessageFactory` using its `newInstance()` method. The `SOAPMessage` object is created from the `MessageFactory` instance using the `createMessage()` method. The `SOAPHeader` and `SOAPBody` are obtained from the `SOAPMessage` object using the `getSOAPHeader()` and `getSOAPBody()` methods respectively.

```
MessageFactory factory =
  MessageFactory.newInstance();
SOAPMessage message = factory.createMessage();

SOAPHeader header = message.getSOAPHeader();
SOAPBody body = message.getSOAPBody();
```

All SOAP messages are sent and received over a connection. With the SAAJ API, the connection is represented by a `SOAPConnection` object, which goes from the sender directly to its destination. This kind of connection is called a *point-to-point* connection because it goes from one endpoint to another endpoint. Messages sent using the SAAJ API are called *request-response* messages. They are sent over a `SOAPConnection` object with the call method, which sends a message (a request) and then blocks until it receives the reply (a response).

```
SOAPConnectionFactory soapConnectionFactory =
  SOAPConnectionFactory.newInstance();
SOAPConnection connection =
  soapConnectionFactory.createConnection();
SOAPFactory soapFactory =
  SOAPFactory.newInstance();
//…
SOAPMessage response = connection.call(soapRequest, endpointURL);
```

The following UML Diagram represents the class design of the SAAJ API. The SOAPElement interface which is the base class for all SOAP classes extends from the Node class. The SAAJ API design follows the DOM design pattern forming a tree structure of the nodes. The SOAPMessage class contains the SOAPPart and AttachmentPart. The SOAPPart acts as the container for the SOAPBody which contains the SOAPEnvelope. The AttachmentPart contains all the binary attachments to this SOAP Envelope.

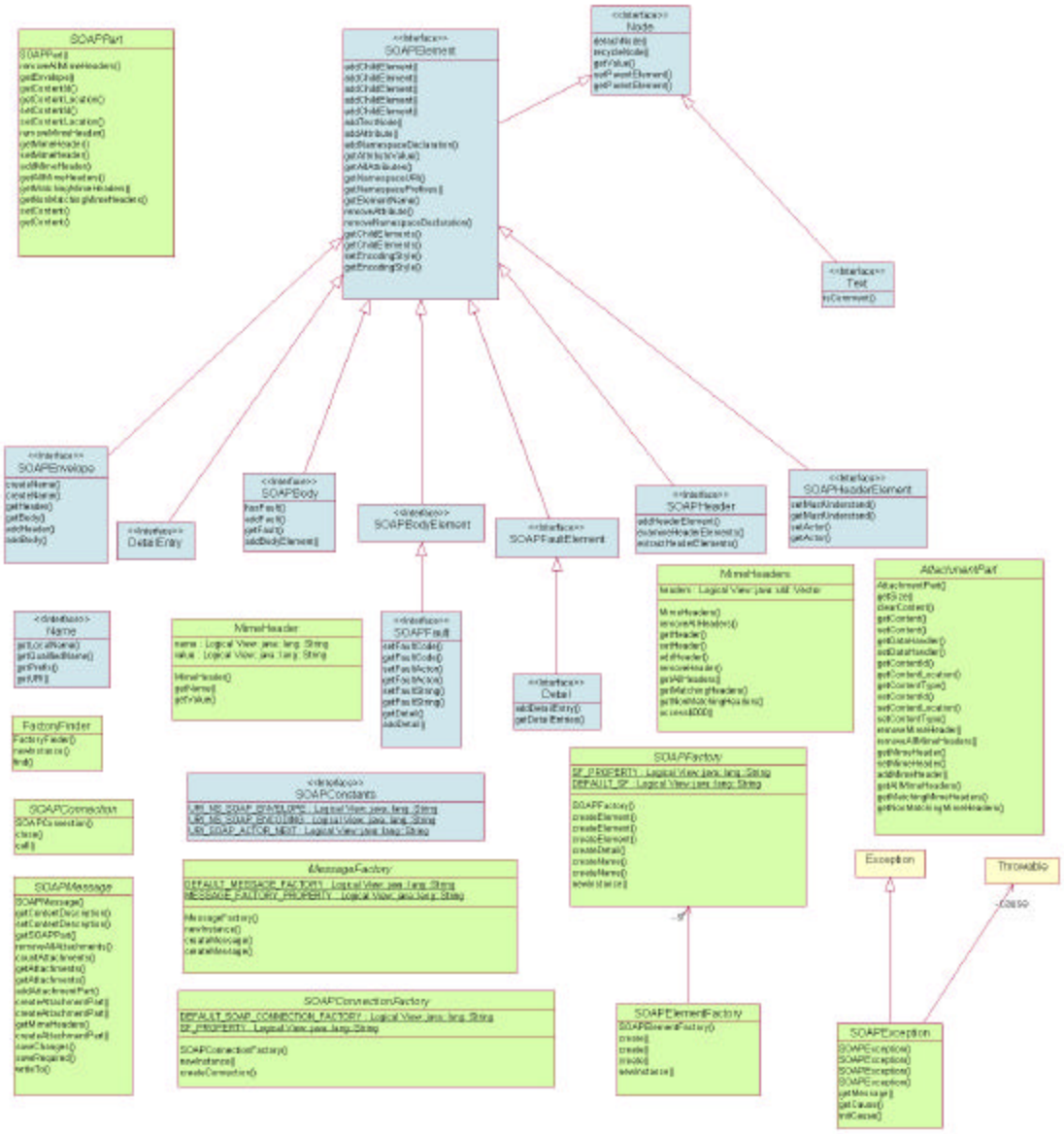

Figure 8 SAAJ Class Design

7.2 DIME and WS-Attachment

Microsoft and IBM submitted a proposal called DIME (Direct Internet Message Encapsulation) together with the WS-Attachments specification which uses DIME as the packaging mechanism for web service attachments. DIME effectively solves the text string boundary problem of MIME by defining length for each part of the combined data set, so a part can be easily skipped in a stream without having to search for the boundary string. DIME and WS-Attachment specifications did provide better performance than SOAP with Attachments, but the interoperability for DIME and WS-Attachments was not good. Now with Microsoft also not supporting DIME, there isn't any practical value associated with the DIME and WS-Attachment specifications.

7.3 WS-I Attachment Profile

There were interoperability issues which resulted from the underspecified nature of the SwA and WSDL 1.1. To address these interoperability issues, a group was formed (WS-Interoperability) who produced an Attachment Profile 1.0 (AP 1.0) which addresses the interoperability concerns. AP 1.0 defines two new mechanisms to address the interoperability concerns. The first one was the definition of XML Schema Type – swaRef. swaRef is simple an xsd:anyURI type, but AP1.0 gives it some special semantics: when a WSDL message part uses the swaRef type in its schema definition, any node of type swaRef in the runtime XML instance of this message part should contain a URI reference to an attachment part in the same MIME package. swaRef enables WSDL abstract interface to suggest about the presence of attachments at run time. AP 1.0 also defines a concrete naming convention for content-id of runtime MIME parts to address the problem of mapping MIME parts to corresponding WSDL message parts.

7.4 Proposed Infoset Addendum to SOAP with Attachments (PaSwA)

PaSwA was developed to address the drawbacks of the SwA. PaSwA built on top of SwA tries to meet both the key requirements of providing a common logical view of the combined data set and providing a serialization for packaging and transmitting the combined data set. PaSwA inherits from SwA the MIME over HTTP based solution for packaging and transmission enhancements. The key value of PaSwA is its support for XML Infoset based single logical view of the whole combined data set. The PaSwA work defines new constructs for use in runtime SOAP message to indicate presence of attachments and also to describe attachments in XML Schema and WSDL. The PaSwA work did not make it to any standards body, but rather it laid the foundation for the MTOM specification which has now become a candidate recommendation from W3C. MTOM borrows many concepts without change from the PaSwA.

8 MTOM & XOP

The World Wide Web Consortium (W3C) has published three new recommendations: XML-binary Optimized Packaging (XOP), SOAP Message Transmission Optimization (MTOM), and Resource Representation SOAP Header Block (RRSHB) to provide a way to efficiently package and transmit binary data included or referenced in a SOAP 1.2 message. These three new recommendations will help speed up web services by enabling a more efficient way of serializing and transmitting a SOAP message and by sending all the data needed to process the message, even when the data would not be readily available.

XML-binary Optimized Packaging (XOP) provides a standard method for applications to include binary data, as is along with an XML document in a package. As a result, applications need less space to store the data and less bandwidth to transmit it. XOP works at the XML Information Set level, allowing the same abstract representation of a XML document to be serialized in different ways.

The Message Transmission Optimization Mechanism (MTOM) uses the features provided by XOP to address SOAP messages. MTOM defines a “Transmission Optimization” feature that enables SOAP bindings to optimize the transmission and/or the wire format used to transfer a SOAP message. It also defines a concrete implementation of this feature, using HTTP and XOP to send the various binary parts as well as the SOAP message in a MIME envelope, reducing the bandwidth and the time used to encode/decode such data.

The Resource Representation SOAP Header Block (RRSHB) functionally allows SOAP message recipients to access cached representations of external resources, when access to the referenced file is not available. RRSHB provides an option to the recipient to use either the original file that may be identified by the URI, or use a cached copy that accompanies the actual SOAP message. Used with MTOM, it greatly enhances the processing speed as the external data is already present when the recipient is starting processing the message.

8.1 Message Optimization Use cases

The need for optimizing messages passed through a SOAP message is discussed in detail at the XMLP WG (XML Protocol working group) of the W3C. The following use cases are taken from the document titled “SOAP Optimized Serialization Use Cases and Requirements” which is a working draft from W3C.

- *Resource “bundled” with message* - An application wishes to send a URI in a message, and the receiver will dereference the URI. The sender chooses to “bundle” a representation of the resource in the message as well, so that the receiver may choose this representation when it dereferences the URI. This is useful in cases when the resource identified by the URI is inaccessible to the receiver, or the receiver wants to avoid the overhead of dereferencing the resource over the network, etc.

- *Message containing "Redundant" pieces of binary data* - An application wishes to send a SOAP message that contains redundant pieces of binary data. Each such piece of binary data is actually contained in several locations of the SOAP message (for example, the SOAP message could contain an XML document which includes in several locations the PNG logo of the company which authored the document). For design reasons, this data cannot be referenced externally, but must be transported with the message. Doing so by copying each piece of binary data as many times as it is contained in the SOAP message involves an unacceptable message size, due to bandwidth, latency and/or storage requirements.
- *Undesirable Message Bloat* - An application wishes to send a SOAP message that contains one or more large binary pieces of data; e.g., a JPG image, binary executable or other sizeable, non-textual data. For design reasons, this data cannot be referenced externally, but must be transported with the message. Doing so using base64Binary or similar encoding involves an unacceptable message size, due to bandwidth, latency and/or storage requirements.
- *Undesirable Processing Overhead* - An application wishes to send a SOAP message that contains one or more large binary pieces of data; e.g., a JPG image, binary executable or other sizeable, non-textual data. For design reasons, this data cannot be referenced externally, but must be transported with the message. Doing so using base64Binary or similar encoding involves unacceptable message generation and/or parsing overhead, due to throughput requirements, device limitations and/or other considerations.
- *Focused Intermediaries* - An intermediary wishes to process a large message, but only needs to access a small section of the message. For efficient operation, it needs to be able to construct the relevant parts of the message information set without actually reading and parsing the rest of the message, whilst still being able to successfully forward the entire message.

8.2 SOAP Message Transmission Optimization Mechanism (MTOM)

The W3C MTOM standard selectively encodes portions of a SOAP message using XOP to efficiently serialize XML Infosets containing binary data. MTOM streams binary data as MIME message parts. At the Infoset level, before final message serialization, the binary data can be accessed as canonical bas-64-encoded text so that other WS-* mechanisms, such as WS-Security signatures, continue to work properly. This allows seamless integration of MTOM with other WS-* protocols while still reducing the bandwidth required to send binary data over the wire.

The MTOM specification has three parts. The first part, Abstract SOAP Transmission Optimization Feature, describes an abstract feature for optimizing the transmission and/or wire format of a SOAP Message while selectively encoding portions of the message, while still presenting an XML Infoset to the SOAP Application. Use of the Abstract SOAP Transmission Optimization Feature is a hop-by-hop contract between a SOAP node and the next SOAP node in the SOAP message path, providing no mandatory convention for optimization of SOAP transmission through intermediaries. The feature does provide optional means by which binding implementations MAY choose to

facilitate the efficient pass-through of optimized data contained within headers or bodies relayed by an intermediary.

The second part, An Optimized MIME Multipart/Related Serialization of SOAP Messages, describes an Optimized MIME Multipart/Related Serialization of SOAP Messages implementing the Abstract SOAP Transmission Optimization Feature in a binding independent way using the XML-binary Optimized Packaging mechanism.

The third part, HTTP SOAP Transmission Optimization Feature, uses this Optimized MIME Multipart/Related Serialization of SOAP Messages for describing an implementation of the Abstract Transmission Optimization Feature for the SOAP 1.2 HTTP binding.

8.2.1 Abstract SOAP Transmission Optimization Feature

The Abstract SOAP Transmission Optimization Feature enables SOAP bindings to optimize the transmission and/or wire format of a SOAP message by selectively encoding portions of the message, whilst still presenting an XML Infoset to the SOAP application. Optimization is available only for element content that is in a canonical lexical representation of the `xs:base64Binary` data type.

The purpose of the Abstract SOAP Transmission Optimization Feature is to optimize the transmission of base64 encoded data. To be optimized, the characters comprising the children of an *element information item* MUST be in the canonical form of xs:base64Binary and MUST NOT contain any whitespace characters, preceding, inline with or following the non-whitespace content. The means of identifying *element information items* that contain base64 encoded data in canonical lexical form are implementation dependent. Some implementations can identify base64encoded *element information item* by construction using introspection of the API, others may check the characters prior to sending or may check for the presence of xmlmime:expectedMediaType schema annotation. When sending a SOAP Message, if the Abstract Transmission Optimization feature is used in combination with the SOAP Request-Response Message Exchange Pattern or the SOAP Response Message Exchange Pattern, the http://www.w3.org/2003/05/soap/mep/OutboundMessage property is set as the Infoset of the SOAP message to be sent.

When receiving a SOAP message optimized using an implementation of the Abstract SOAP Transmission Optimization Feature, a SOAP node SHOULD generate a fault if it does not support the implementation used or the Abstract SOAP Transmission Optimization Feature. Upon reception of an optimized SOAP message, the receiving node MUST reconstruct an Envelope Infoset from the optimized SOAP message. Then, the receiving node MUST perform SOAP processing on the reconstructed Infoset. Implementations are free to reconstruct only those portions actually needed for processing, or to present information from the message in a form convenient for efficient processing. For example, a value sent in an optimized form (e.g., binary) MAY be made available in that form as well as in the base64 encoded character form. When this feature is used in combination with the SOAP Request-Response Message Exchange Pattern or

the SOAP Response Message Exchange Pattern, the Infoset contained in the http://www.w3.org/2003/05/soap/mep/InboundMessage property is the Infoset of the reconstructed SOAP Envelope.

Use of the Abstract SOAP Transmission Optimization Feature is a hop-by-hop contract between a SOAP node and the next SOAP node in the SOAP message path. Therefore, no changes or restrictions to the SOAP processing model are introduced by this feature at an intermediary. However a SOAP intermediary implementing the Abstract Transmission Optimization Feature MUST still follow the rules related to the usage of an implementation of the Abstract Transmission Optimization Feature when receiving the message and those related to the usage of an implementation of the Abstract SOAP Transmission Optimization Feature when sending the message. A SOAP intermediary may be called upon to relay intact certain headers, or to reinsert headers identical to those received and removed for processing. Furthermore, many intermediaries will relay unmodified the contents of the SOAP body. In all these cases, portions of the relayed message have content identical to corresponding portions of the inbound message. The Abstract SOAP Transmission Optimization Feature does not require any particular correspondence between the optimization of the inbound message and the outbound message, even when optimized portions of the inbound message are relayed intact, or reinserted in identical form in the envelope Infoset. Nonetheless, the implementations of the receiving binding and the binding used to transmit the relayed message MAY cooperate to provide efficient relay. The Abstract SOAP Optimizaation Feature does not mandate any strict rules about the level of optimizations to be followed by the SOAP intermediary nodes. The optimizations done by the intermediaries on the messages transmitted through them are based on the capability for optimzation of the intermediary nodes.

8.2.2 An Optimized MIME Multipart/Related Serialization of SOAP Messages

The Optimized MIME Multipart/Related Serialization provides the basis of an implementation of the Abstract SOAP Transmission Optimization Feature bye describing how to serialize a SOAP envelope infoset in an optimized way using the XOP format and a MIME Multipart/Related packaging.

When sending a SOAP message using the MIME Multipart/Related Serialization, the SOAP envelope Infoset is serialized as follows:

- The content-type of the outer package MUST be `multipart/related`.
- The type parameter of the content-type header of the outer package MUST have a value of "`application/xop+xml`"
- The startinfo parameter of the content-type header of the outer package MUST specify a content-type for the root part of "`application/soap+xml`".
- The content-type of the root part MUST be "`application/xop+xml`"
- The type parameter of the content-type header of the root part MUST specify a content-type of "`application/soap+xml`".

The serialized SOAP Envelope is a Multipart/Related XOP package with one body part, the root, containing an XML representation of the modified SOAP envelope, with an

additional part used to contain the binary representation of each element that was optimized.

When receiving a SOAP message using this Optimized MIME Multipart/Related Serialization, the SOAP Envelope Infoset is reconstructed from the MIME Multipart/Related XOP Package as follows:

1. Construct an XML Infoset by parsing the root part of the package as an XML document. The document must be parsed according to the level of the XML Recommendation identified by the XML declaration of that document. If no XML declaration is present, then the document MUST be parsed per [XML 1.0].
2. Using that XML Infoset, for each *element information item* , E, which has, as the sole member of its [children] property, a `xop:Include` *element information item* (as defined in `xop:Include` element information item):
    a. Locate the part of the package corresponding to the URI in the `href` *attribute information item* of the `xop:Include` *element information item* (i.e, corresponding to the URI encoded in the *attribute information item's* [normalized value]).
    b. Replace the `xop:Include` *element information item* that appears in the [children] property of E with *character information items* representing the canonical base64 encoding of the entity body of the identified package part (i.e, effectively replace the `xop:Include` *element information item* with the data reconstructed from the package part.

8.2.3 HTTP SOAP Transmission Optimization Feature

The HTTP SOAP Transmission Optimization Feature is a binding-level feature providing a concrete implemention for the Abstract SOAP Transmission Optimization Feature in an HTTP binding as described by An Optimized MIME Multipart/Related Serialization of SOAP Messages. The HTTP SOAP Transmission Optimization Feature builds upon the SOAP HTTP Binding enhancing it with support of the Abstract SOAP Transmission Optimization Feature.

The sender serializes the SOAP Message as described by the Optimized MIME Multipart/Related Serialization and puts the MIME headers of the resulting MIME Multipart/Related XOP Package in as HTTP headers an d the rest of the package into the HTTP body. The receiver extracts the MIME headers from the HTTP headers and the rest of the MIME Multipart/Related XOP Package from the HTTP body and deserializes.

The XOP package constructed by the sender follows these restrictions:

- The XOP Infoset MUST be serialized as `application/xop+xml` in the root part of the package.
- Each optimized *Node* MUST generate exactly one extracted binary part in the resulting package.
- Each MIME part that is refered to by `xop:Include` MUST have a Content-Transfer-Encoding header field.

Implementations of this binding MUST enforce the restriction that XOP is not to be used with Infosets that contain *element information items* of name `xop:Include`. If a SOAP envelope containing such an *element information item* is to be sent, the binding MUST do one of the following:

- Fall back to use the `application/soap+xml` media type or any other suitable media type.
- Generate a binding-dependent SOAP fault.

When receiving a SOAP message, an implementation of the SOAP HTTP Binding will determine whether the HTTP SOAP Transmission Optimization Feature is used by checking the presence of the `application/xop+xml` media type. If the media type of the HTTP message is "`multipart/related`" and the media type of the root part of the MIME Multipart/Related package is `application/xop+xml`, and the start-info parameter indicates a content type of "`application/soap+xml`" then the received SOAP message was transmitted using the HTTP SOAP Transmission Optimization Feature and MUST be processed accordingly.

The HTTP SOAP Transmission Optimization Feature changes the behavior of SOAP HTTP Binding for the reception of a SOAP Message by deserializing the SOAP Infoset contained in the http://www/w3.org/2003/05/soap/mep/InboundMessage as described by the Optimized MIME Multipart/Related Deserialization of SOAP Messages.

8.3 XML Binary Optimized Packaging (XOP)

This XOP specification which is now a candidate reccomendation from W3C defines the XML-binary Optimized Packaging (XOP) convention, which is a means of more efficiently serializing XML Infosets that have certain types of content.

A XOP package is created by placing a serialization of the XML Infoset inside of an extensible packaging format such a MIME Multipart/Related. Then, selected portions of its content that are base64-encoded binary data are extracted and re-encoded (i.e., the data is decoded from base64) and placed into the package. The locations of those selected portions are marked in the XML with a special element that links to the packaged data using URIs.

In a number of important XOP applications, binary data need never be encoded in base64 form. If the data to be included is already available as a binary octet stream, then either an application or other software acting on its behalf can directly copy that data into a XOP package, at the same time preparing suitable linking elements for use in the root part; when parsing a XOP package, the binary data can be made available directly to applications, or, if appropriate, the base64 binary character representation can be computed from the binary data.

However, at the conceptual level, this binary data can be thought of as being base64-encoded in the XML Document. As this conceptual form might be needed during some processing of the XML Document (e.g., for signing the XML document), it is necessary to have a one to one correspondence between XML Infosets and XOP Packages.

Therefore, the conceptual representation of such binary data is as if it were base64-encoded, using the canonical lexical form of XML Schema base64Binary datatype. In the reverse direction, XOP is capable of optimizing only base64-encoded Infoset data that is in the canonical lexical form.

Only element content can be optimized; attributes, non-base64-compatible character data, and data not in the canonical representation of the base64Binary datatype cannot be successfully optimized by XOP.

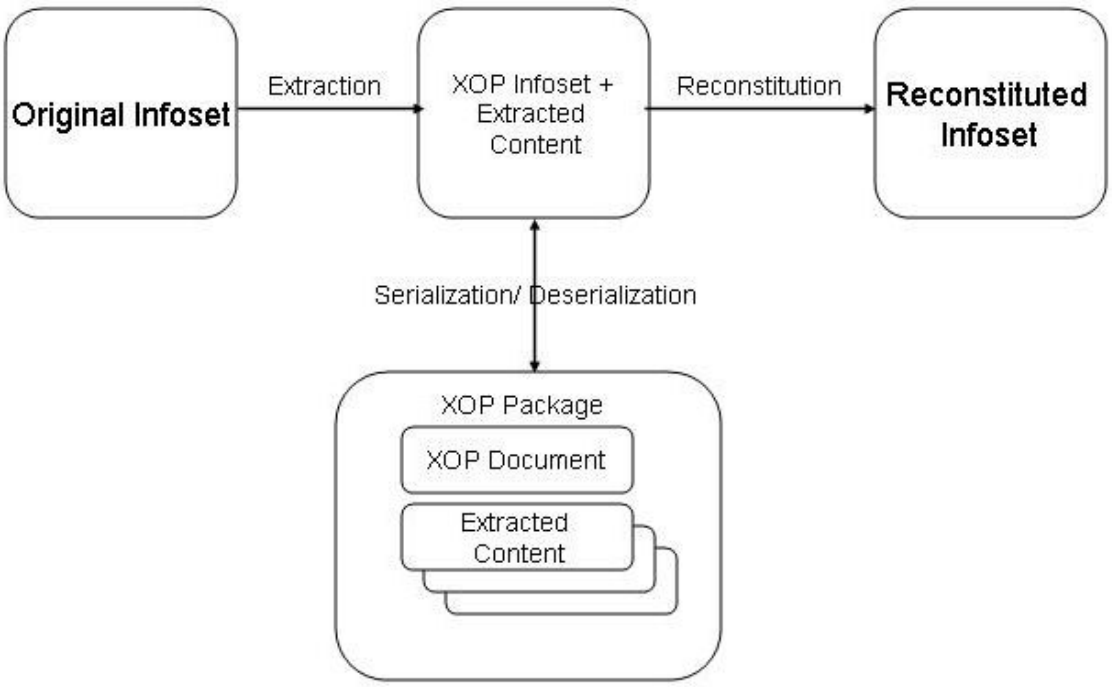

Figure 9 XOP Framework Architecture

The following example shows how an XML Infoset containing binary data can be serialized using the XOP format in a MIME Multipart/Related package.
The original XML Infoset

```
<m:data xmlns:m='http://example.org/stuff'>
  <m:photo>/aWKKapGGyQ=</m:photo>
  <m:sig>Faa7vROi2VQ=</m:sig>
</m:data>
```

The XOP serialized Infoset

```
MIME-Version: 1.0
Content-Type: Multipart/Related;boundary=MIME_boundary;
    type="application/xop+xml";
    start="<mymessage.xml@example.org>";
    start-info="text/xml"
Content-Description: An XML document with my pic and sig in it

--MIME_boundary
```

```
Content-Type: application/xop+xml;
    charset=UTF-8;
    type="text/xml"
Content-Transfer-Encoding: 8bit
Content-ID: <mymessage.xml@example.org>

<m:data xmlns:m='http://example.org/stuff'>
  <m:photo><xop:Include
  xmlns:xop='http://www.w3.org/2004/08/xop/include'
  href='cid:http://example.org/me.png'/></m:photo>
  <m:sig><xop:Include
  xmlns:xop='http://www.w3.org/2004/08/xop/include'
  href='cid:http://example.org/my.hsh'/></m:sig>
</m:data>

--MIME_boundary
Content-Type: image/png
Content-Transfer-Encoding: binary
Content-ID: <http://example.org/me.png>

// binary octets for png

--MIME_boundary
Content-Type: application/pkcs7-signature
Content-Transfer-Encoding: binary
Content-ID: <http://example.org/my.hsh>

// binary octets for signature

--MIME_boundary—
```

XOP operates by extracting the Optimized Content from the Original Infoset to create the XOP Infoset. In particular, the *character information item* children of *element information items* to be optimized are removed and replaced with an element information item named `xop:Include`. The `xop:Include` element information item contains an *attribute* information item with a link to the part of the XOP Package that carries a binary representation of the data removed from the original element information item.

The Infoset used as input to XOP processing MUST NOT contain any *element information item* with a namespace name property of "http://www.w3.org/2004/08/xop/include" and a local name property of `Include`. Infosets containing such *element information items* cannot be serialized using XOP. This is because during infoset reconstruction a processor is unable to differentiate between `xop:Include` *element information items* inserted during XOP package construction and those that were part of the original infoset.

The XOP Processing Model defines how to serialize binary data to XOP packages and also to deserialize XOP packages and extract the binary data from it.

To create a XOP Package from an Original XML Infoset:

2. Ensure that the original XML Inforset contains no *element information item* with a namespace name of "http://www.w3.org/2004/08/xop/include" and a local name of

`include`. As discussed in 'http://www.w3.org/TR/2004/CR-xop10-20040826' XOP Infoset Constructs. XML Infosets with such *element information items* cannot be represented using XOP.

3. Create an empty package.
4. Identify within the original XML Infoset the *element information items* to be optimized. To be optimized, the characters comprising the children of the *element information item* MUST be in the canonical form of `xs:base64Binary` and MUST NOT contain any whitespace characters, preceding, inline with or following the non-whitespace content.
5. Create a XOP Infoset which is a copy of the Original XOP Infoset, but with the children of each *element information item* identified in the previous step replaced by a `xop:Include` *element information item* construcuted as follows:
    a. Transform the replaced characters into binary data by processing them as base64-encoded data.
    b. Serialize the binary data into a new part of the package, with appropriate metadata corresponding to the normalized value of the `href` *attribute information item* of the `xop:Include` *element information item.*
    c. If the *element information item* being optimized (i.e. the parent of the newly inserted `xop:Include` *element information item*) has a `xmlmime:contentType` *attribute information item*, its value SHOULD be reflected appropriately in the metadata for the part.
6. Serialize the resulting XOP Infoset into the package using any W3C recommendation-level version of XML and identify it as the root part according to the packaging mechanism's convention, labeling it with the application/xop+xml media type.

To create a Reconstituted XML Infoset from a XOP Package:

1. Construct an XML Infoset by parsing the root part of the package as an XML document. The document must be parsed according to the level of the XML Recommendation identified by the XML declaration of that document. If no XML declaration is present, then the document MUST be parsed per [XML 1.0].
2. Using that XML Infoset, for each *element information item* , E, which has, as the sole member of its [children] property, a `xop:Include` *element information item* (as defined in `xop:Include` element information item):
    a. Locate the part of the package corresponding to the URI in the `href` *attribute information item* of the `xop:Include` *element information item* (i.e, corresponding to the URI encoded in the *attribute information item's* [normalized value]).
    b. Replace the `xop:Include` *element information item* that appears in the [children] property of E with *character information items* representing the canonical base64 encoding of the entity body of the identified package part (i.e, effectively replace the `xop:Include` *element information item* with the data reconstructed from the package part.

8.4 Resource Representation of SOAP Header Block

The Representation header block is designed to allow applications to carry a representation of a Web resource in a SOAP message. Applications of this header include cases where the receiver has limited ability to get the representation using other means, for example because of access restrictions or because the overhead would be unacceptable. The Representation header block is also useful when multiple references to the same resource are required but duplication of the resource is undesirable.

The meaning of the Representation header block, when present in a SOAP message, is to make available the contained representation of the resource it carries to the processing SOAP node. The SOAP node MAY use this representation when dereferencing the URI of the resource instead of making a network request to obtain a representation of the resource. Note that implementations MAY need to process a Representation header block before processing other header blocks that require dereferencing of a URI whose representation is carried in the Representation header block.

Multiple occurrences of the Representation header block MAY be present in the same SOAP Message to carry representation of multiple Web resources or multiple representations of the same Web resource.

Several occurrences of the Representation header block having the same value for the role and resource attribute information item MAY be present in the same SOAP Message. Such Representation header blocks SHOULD NOT have the same metadata (such as media-type). If such Representation header blocks have the same metadata then any one of them may be used.

URIs that are character for character identical MUST be considered equal when using a representation header to resolve a web reference; URIs that are considered equal according to the URI scheme of the URI SHOULD be considered equal.

An example SOAP Envelope using the Representation header block is given below.

```
<soap:Envelope xmlns:soap='http://www.w3.org/2003/05/soap-envelope'
               xmlns:rep='http://www.w3.org/2004/08/representation'
               xmlns:xmlmime='http://www.w3.org/2004/11/xmlmime'>
  <soap:Header>
    <rep:Representation resource='http://example.org/me.png'>
      <rep:Data xmlmime:contentType='image/png'>/aWKKapGGyQ=</rep:Data>
    </rep:Representation>
  </soap:Header>
  <soap:Body>
    <x:MyData xmlns:x='http://example.org/mystuff'>
      <x:name>John Q. Public</x:name>
      <x:img src='http://example.org/me.png'/>
    </x:MyData>
  </soap:Body>
</soap:Envelope>
```

The `rep:Representation` *element information item* has:

- A [local name] of `Representation`.
- A [namespace name] of "http://www.w3.org/2004/08/representation".
- One or more *attribute information items* amongst its attributes as follows:
    - A mandatory `resource` *attribute information item*.
    - An optional `reinsert` *attribute information item*.
    - Zero or more namespace qualified *attribute information items* whose namespace name is not "http://www.w3.org/2004/08/representation".
- One or more *element information items* in its children property in order as follows:
    1. A mandatory `Data` *element information item* (see 4.2.4 rep:Data element).
    2. Zero or more *element information items* whose [namespace name] is not "http://www.w3.org/2004/08/representation".

The `rep:Representation` *element information item* contains a representation of a Web resource. The value of the resource *attribute information item* is the identifier of the Web resource. The value of the `rep:Data` *element information item* is a base64-encoded representation of the Web resource.

The type of the `resource` *attribute information item* is `xs:anyURI`. The value of the `resource` *attribute information item* identifies the Web resource whose representation is carried in the `rep:Representation` *element information item* parent of the `resource` *attribute information item*.

The type of the `reinsert` *attribute information item* is `xs:boolean`. When this attribute is specified on the Representation header block with a value of "true", it indicates that a SOAP forwarding intermediary node processing the header block must reinsert the header block. This means that when used in conjunction with the `relay` attribute, with a value of "true", the Representation header block will always be relayed by a SOAP forwarding intermediary. When this attribute is specified on the Representation header block with a value of "false", the behavior of the SOAP node processing the header block is the same as that when the attribute is not specified, and normal SOAP processing rules apply. The presence of this attribute has no effect on the processing of a Representation header by a SOAP endpoint.

The type of a `rep:Data` *element information item* is `xs:base64Binary`. The value of this *element information item* is a base64-encoded representation of the Web resource carried in the `rep:Representation` *element information item* parent of the `resource` *attribute information item.*

The Representation header block is built to be extensible. Attributes defined in SOAP messaging framework for SOAP message construct for the SOAP header block may be used with the Representation header block.

Adding a `env:mustUnderstand` *attribute information item* with a value of "true" in the attributes property of the `rep:Representation` *element information item* ensures that the SOAP receiver is aware that the Web resource representation is available to it.

A `env:role` *attribute information item* in the attributes property of the `rep:Representation` *element information item* indicates the SOAP node for which the Web resource representation is intended.

An `xmlmime:contentType` *attribute information item* MAY be used to convey the media type of the representation conveyed by a header. Media type information can be useful in determining whether a given representation is suitable for processing and if it is, how best to interpret the representation provided. If used, the `xmime:contentType` *attribute information item* MUST appear within the attributes property of the `rep:Data` *element information item*. If the media type identified by the value of an `xmime:contentType` *attribute information item* is a text based media type then the value of the `xmime:contentType` *attribute information item* SHOULD include a charset parameter.

A receiving SOAP node MAY act as a resolver, with all the rules pertaining to HTTP caches, for some or all of the *http:* scheme URIs for which representations have been provided. To enable this, one or more element information items MAY be added to the [children] property of the `rep:Representation` *element information item* to transmit the information needed at the HTTP level.

To avoid requiring that all SOAP senders understand the HTTP caching mechanism, all the data required by a processor that wants to act as a local cache needs to be carried along with the message. This includes the complete request, reply as well as the time the original HTTP request has been sent and the time the HTTP response has been received.

8.5 Assigning Media Types to Binary Data in XML

Data sent and received over the Web typically uses the MIME media, as the type system. For example, "image/jpeg", "application/pdf". There is a need to indicate the media type of the XML element content, for example, in messages sent and received by Web services. There is also a need to express the media type information using XML Schema: Datatypes, which is the type system used by WSDL 2.0 Part 1. This would allow XML-based applications, such as Web services, to utilize the widely deployed and supported MIME media type infrastructure.

XOP and MTOM enables serialization of binary content (element content that is in a canonical lexical representation of the xs:base64Binary type) in an optimized way using MIME packaging. These developments to a desire to specify the media type information of such binary element content in a standard way in the XML Information Set and not just in the optimized serialization of that Infoset. This desire was documented by the W3C working group in the form of the following requirements:

1. Define how to indicate the media type of an XML element content whose type is *xs:base64Binary* or *xs:hexBinary*. This is meta-data that may be, but not required to, used by tools to infer the specific media type of binary data.
2. Define how to indicate the expected media type(s) of XML element content whose type is *xs:base64Binary* or *xs:hexBinary* in XML Schema. This information is needed to define the set of media types that a binary data may have. For example, a Web services application may be willing to indicate that the binary

data represents an image, but leaves it to a document to further specify whether it is "jpeg", "gif", etc. This meta-data is not required to be present.

3. Define the acceptable format of media type values.
4. Define the relationship between the expected and the actual value of the media types declared for binary data in XML documents.

The Assigning Media Types to Binary Data in XML specification is a W3C effort to provide a solution for the above mentioned requirements. In this direction this specification defines two attributes *xmlmime:contentType* and *xmlmime:expectedMediaType*. The *xmlmime:contentType* attribute is used to indicate the media type of an XML element content whose type is *xs:base64Binary* or *xs:hexBinary*. The value of the attribute is the name of a IANA media type token (e.g., "text/xml; charset=utf-16"). This attribute specifies the media type of the content of an element on which it occurs. The XML Schema annotation attribute *xmlmime:expectedMediaType* is used to indicate, in XML Schema, the expected media type(s) for an element content whose type is *xs:base64Binary* or *xs:hexBinary*.

The XML Schema annotation, *xmlmime:expectedMediaType*, specifies the expected range of values for the *xmlmime:contentType* attribute and the expected range of media types for the binary element content.

XML Documents that want to specify additional media type information for binary data SHOULD denote this by using a *binary element information item*. A *binary element information item* is an *element information item* defined with the following additional constraints:

- An OPTIONAL `contentType` *attribute information item*
- The *character information items* comprising the children of the *element information item* MUST conform to the lexical constraints of *xs:base64Binary* or *xs:hexBinary.*

The normalized value of the `contentType` *attribute information item* MUST be the name of a IANA media type token, e.g., "image/png", "text/xml; charset=utf-16" and indicates the media type of the owner element. The following example shows the instance of a document having binary contentType attribute whose value is “image/jpg”

```
<?xml version="1.0" ?>
<Image xmlns="http://www.pace.edu/studentImage"
           xmlns:xmlmime="http://www.w3.org/2004/11/xmlmime"
           xmlmime:contentType="image/jpg">/bXLMcoWPzS=</Image>
```

The `contentType` *attribute information item* allows Web services applications to optimize the handling of the binary data defined by a *binary element information item* and should be considered as meta-data. The presence of the `contentType` attribute does not changes the value of the element content.

Applications that need to specify expected media types SHOULD use the schema annotation to declare the range of expected values. `expectedMediaType` annotation attribute MAY be used in conjunction with the declaration of *binary element information*

items or with complex type definitions that are derived from *xs:base64Binary* or *xs:hexBinary* in XML Schema. If the `expectedMediaType` annotation attribute is used in both the *binary element information* item declaration as well as definition of the complex type which the *binary element information* item belongs to, then the expected range of values defined for the *binary element information* item MUST be a subset of the expected range of values defined for the complex type.

9 A Logger System based on Web Services

"A Logger System based on Web Services" [1] defines a Logger System which is a general purpose set of component interfaces for logging data in a distributed, heterogeneous computing environment. Fundamentally, it serves as an intermediary between log artifact producers and log artifact consumers. It is designed to virtualize existing logging systems including the z/OS System Logger, Microsoft Windows event logging, and the UNIX syslog facility. The Logger System is outlined using the conventions defined in the Web Services Resource Framework (WSRF) and provides an example of how this framework might be used to define a meta-OS (operating system) for grid computing.

The Logger System described in [1] serves as a manageable repository of log records. The term *log record* refers to the atomic unit read or written through the Logger System's interfaces. Components that write log records to the Logger System are referred to as *log artifact producers* or simply *producers*. Components that read from the Logger System are referred to as *log artifact consumers* or simply *consumers.* The Logger System serves as an intermediary, decoupling log artifact producers and log artifact consumers. Log artifacts written by log producers may or may not be read at a later time by log consumers. A component that manages a Logger System is referred to as a *log manager*. Log managers maintain the Logger System, performing a variety of administrative functions such as creating new logs, purging obsolete records, and managing retention policies.

9.1 Logger System interfaces

The Logger System design using Web services interfaces is expressed using the conventions and specifications established by WSRF. WSRF defines the Web Services Resource (WS-Resource) approach to modeling and managing state in a Web services context. WSRF consists of:

1. A set of six new Web services specifications.
2. A set of conventions for existing technologies.

To describe some of the semantics of the Logger System, the Logger System use interfaces defined in three of the new Web services specifications. These are Web Services Resource Properties (WS-ResourceProperties), WS-Notification, and Web Services Resource Lifetime (WS-ResourceLifetime),

As part of the conventions for existing technologies, WSRF introduces *implied resource pattern* to describe how Web Services Addressing (WS-Addressing) is used to associate a stateful resource with a Web services interface using WSDL portType element. Let wsa represent the WS-Addressing namespace. The implied resource pattern requires that the wsa:EndpointReference element must include a wsa:ReferenceProperties child element to identify the resource associated with the address specified by the wsa:Address child element of this EndpointReference (EPR). Essentially, the EPR is a pointer containing, among other elements, an address (wsa:Address) of the service and an opaque expression of resource identity (wsa:ReferenceProperty).

The Logger System is composed of four interfaces. They are:

1. The log:LogManager interface
2. The log:Log interface
3. The log:LogConnection interface
4. The log:LogBrowseSession interface

The portTypes supporting these interfaces and their relationships are shown in UML diagram below and are described in the following subsections.

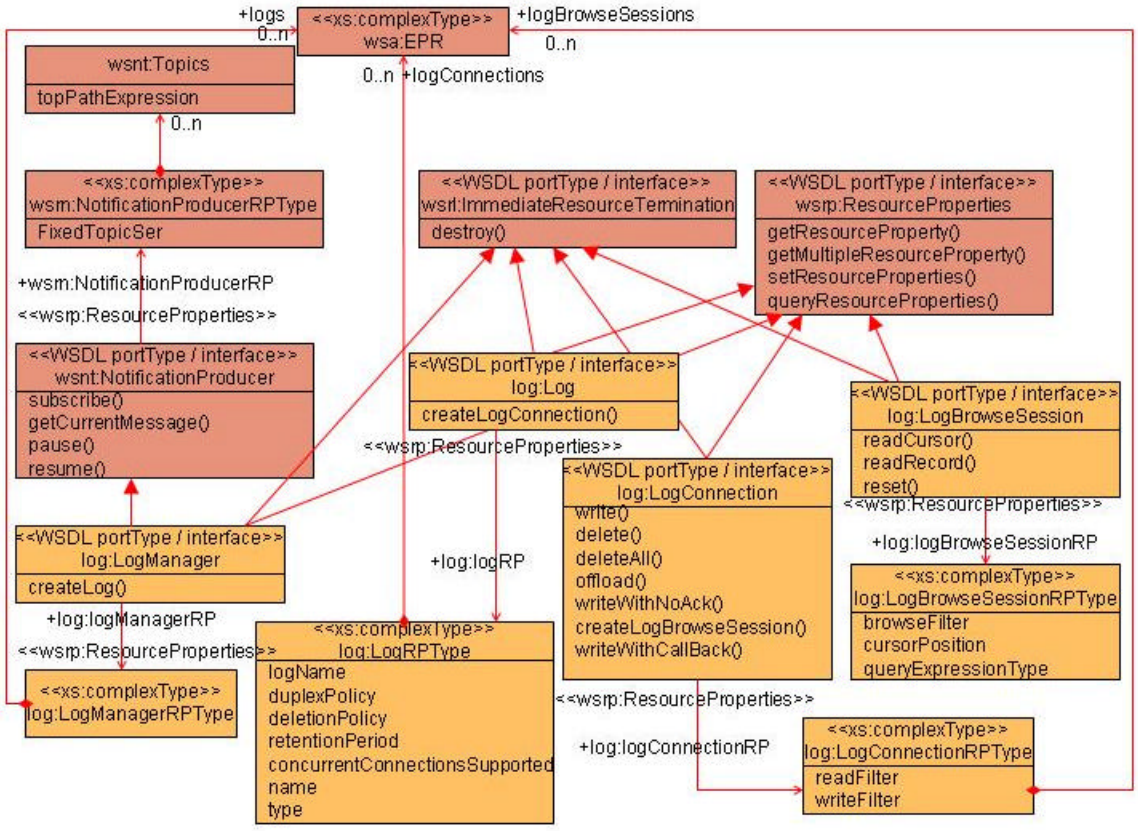

Figure 10 Logger System interfaces

9.1.1 The `log:LogManager` interface

The `log:LogManager` interface is responsible for managing an instance of the Logger System. The `log:LogManager portType` inherits from the `wsrp:ResourceProperties`, `wsnt:NotificationProducer`, and `wsrl:ImmediateResourceTermination portTypes`. The `createLog` operation of the `log:LogManager portType` acts as a factory for `log:Log` resources. This operation may be able to instantiate logs with different QoS. For example, if the underlying system supports a hardware-assisted merge facility and the `log:LogManager` implementation exposes this capability, then a log with merge capability may be instantiated. The home collection of all `log:Log` instances created by an instance of the `log:LogManager`

interface is stored as wsa:EPR child elements in its log:LogManager resource property document. If there are no active connections to any logs managed by a log:LogManager resource, then that resource may be explicitly destroyed with the wsrl:ImmediateResourceTermination portType destroy operation. An explicit destroy of a log:LogManager resource implicitly destroys all the underlying log:Log resources.

The Logger System is an important system and must be managed accordingly. It is necessary not only to monitor its performance but also to deal with storage space thresholds, low-space or insufficient-space conditions, periodic purging, access control, and many other management facets. Management events related to an instance of log:LogManager may be monitored using the inherited wsnt:NotificationProducer portType. All management-related events related to an instance of the Logger System are surfaced through this mechanism.

9.1.2 The log:Log interface

The log:Log portType inherits from the wsrp:ResourceProperties portType and the wsrl:ImmediateResourceTermination portType. A relatively long list of parameters defines the behavior of a particular log service instance. For example, the log resource state could be characterized by the following: duplex policy, deletion policy, retention period, and support for concurrent connections. Although not specified in this paper, it is envisioned that, when appropriate, the setResourceProperties operation of the wsrp:ResourceProperties portType would be used to change the policy of an existing log. Also, technologies related to the specification and deployment of policy could assist in the management of logs in a distributed environment composed of a large number of systems. The createLogConnection operation of the log:Log portType acts as a factory for log:LogConnection resources. The home collection of all log:LogConnection instances created by an instance of log:Log is stored as wsa:EPR child elements in its resource property document. User-specified read and write filters may be set for log:LogConnection instances. If there are no active connections to a log:Log resource, then the log may be explicitly destroyed with the wsrl:ImmediateResourceTermination portType destroy operation. If a log:Log resource is destroyed, then all log records associated with the log are lost.

9.1.3 The log:LogConnection interface

All accesses (read or write) to the records in a log are made through the log:LogConnection interface. The log:LogConnection portType inherits from the wsrp:ResourceProperties portType and the wsrl:ImmediateResourceTermination portType. The log:LogConnection portType contains the write operation, which is the basic mechanism for writing a log record. The writeWithCallback and the writeWithNoAck operations support the requirements outlined in our previous discussion of synchronous and asynchronous write semantics requirements. The delete operation deletes a range of log records from a log. All records older than the specified record are deleted. The deleteAll operation is used to delete all the log records from a log. The offload operation forces any cached log records to be written to a permanent store. The

createLogBrowseSession operation of the `log:LogConnection portType` acts as a factory for `log:LogBrowseSession` resources. A user-specified read filter may be set for a `LogBrowseSession` instance. The home collection of all `log:LogBrowseSession` instances created by an instance of `log:LogConnection` is stored as `wsa:EPR` child elements in its resource property document. A `log:LogConnection` resource may be explicitly destroyed with the `wsrl:ImmediateResourceTermination portType destroy` operation; any `log:LogBrowseSession` resources in the home collection will also be destroyed.

9.1.4 The `log:LogBrowseSession` interface

The `log:LogBrowseSession` interface is used to read log records from a log. The `log:LogBrowseSession portType` inherits from the `wsrp:ResourceProperties portType` and the `wsrl:ImmediateResourceTermination portType`. Each `LogBrowseSession` instance maintains a cursor in its resource properties for navigating its corresponding log. On creation, a custom read filter may be set for an instance of a `LogBrowseSession` interface. Reads are filtered by both the `LogBrowseSession` filter and the `LogConnection` read filter. The `LogBrowseSession portType readRecord` operation accesses records either by record identifier or by record time stamp. The `readCursor` operation reads a sequence of one or more records from the current cursor position. The `reset` operation resets the cursor position to either the oldest or youngest record in the log. As discussed previously, a browse session may support additional query mechanisms. A `log:LogBrowseSession` resource may be explicitly destroyed with the `wsrl:ImmediateResourceTermination portType destroy` operation.

10 Proposed Enhancement to Logger System the based on Web Services

The Enhanced Logger System developed during this work is based on the Logger System which was described, in the paper titled “A Logger System based on Web Services”. The proposed enhancement is to provide a late binding localization to the LogRecords written by the log:LogConnection interface. Currently the localization is restricted to the locale of the environment in which the implementation for the log:LogConnection is executing. With the proposed enhancement, the LogRecords, which are available in binary format, can be transferred to the requesting machine and localized according to the locale of the requestor. This enhancement will also improve the performance of the current Logger System implementation, as extra processing done by the LogRecord Formatters, to localize a LogRecord before hardening, can now be avoided.

The following figure describes the current logger system. The implementation of the current logger system uses the Formatter class, to localize and format the LogRecords in the current locale before it is hardened to the file system.

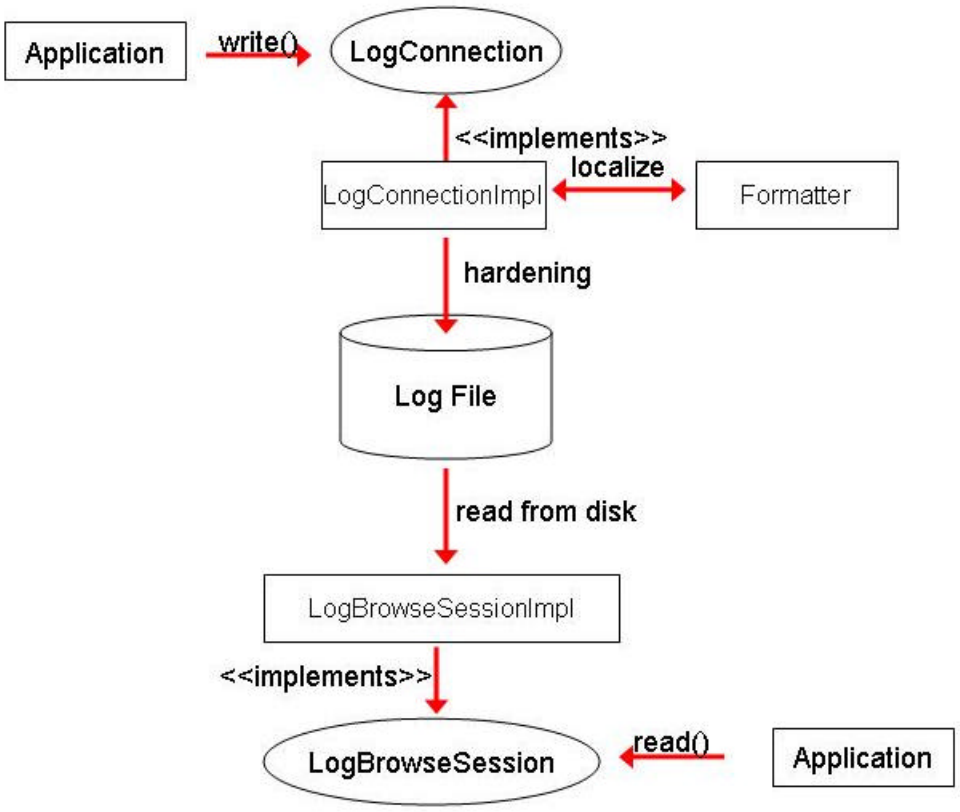

Figure 11 Current Logger System Overview

The enhanced logger system uses late binding for localization of the LogRecords by pushing the localization to the reading application. This improves the performance of the

logger system writes, as the additional processing for localization and formatting is now avoided. When the reading application issues a call to the read() method of the LogBrowseSession interface over Web services, the LogBrowseSession will read the corresponding binary record from the disk, and will send it to the caller application as binary data over SOAP packets using the SOAP Message Transmission Optimization Mechanism (MTOM). The reading application receives the binary data and will localize and format the LogRecords from the binary data, according to its locale requirements. This enhancement also allows the LogRecords to be translated to multiple locales, as compared to a single, origin locale, translation which is available in the Logger System. The figure below shows this enhancement.

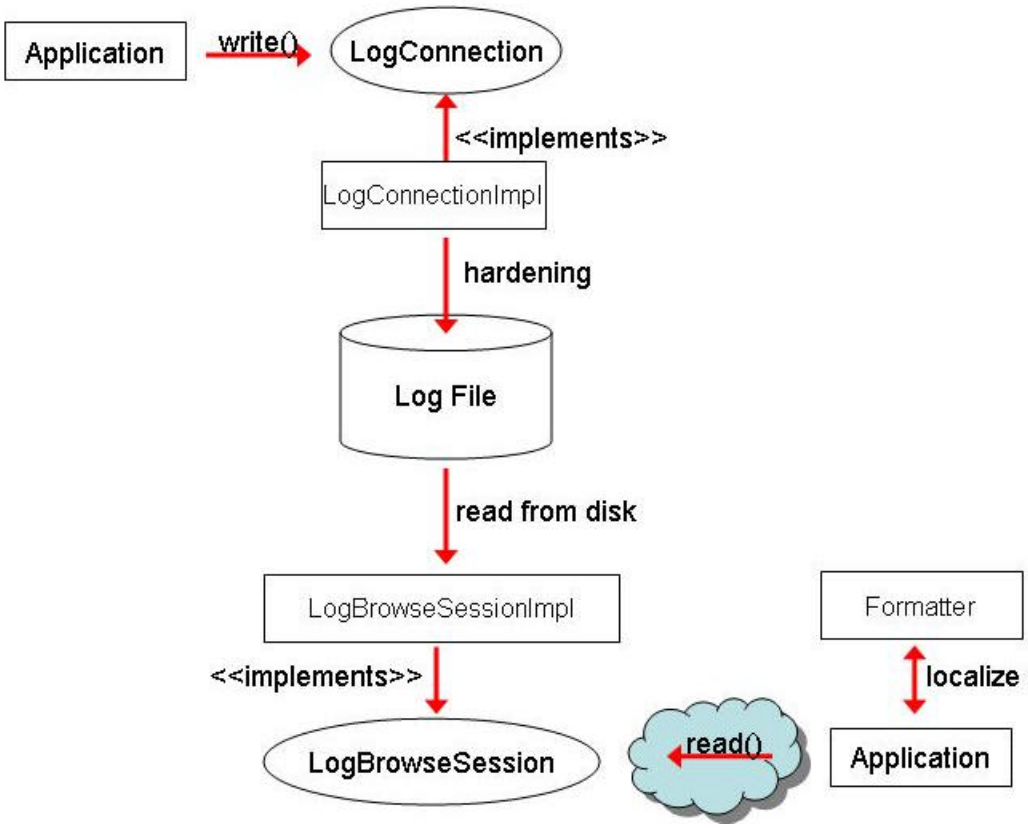

Figure 12 Enhanced Logger System Overview

10.1 Enhanced Logger System Architecture

The Enhanced Logger System is built using an MTOM implementation. The existing logger system will persist log data to the disk with out any localization in pure binary format. The log:LogBrowseSession web service will be enhanced by adding a MTOM layer. With the enhancement the LogBrowseSession web service on receiving a read request for the log data, will filter the binary log data based on the input parameters and send the binary data to the requesting application as an XOP serialized SOAP MTOM message. The receiving application extracts the binary log data from the XOP serialized

MimeMessage and reads the log data from the binary stream. While reading the client application will apply the resource bundles to localize the log messages to its default locale.

The prototype implementation is based on the assumption the base Logger System exists and the log data is persisted to the disk. The java.util.logging framework is used for simulating a logger system in this prototype. The BinaryLogHandler is configured to the Logging framework, which will convert the LogRecords to binary format using the LogRecordBinaryFormatter class. This binary formatted log records are persisted to the disk. When a client request comes for log data, this prototype will introspect the SOAP request message, and based on the request, will read the log data from the disk and return the data to the application. The application will deserialize the binary data, and apply the required resource bundles to extract the locale specific log messages.

The following UML diagram shows all the classes created for implementing the Enhanced Logger System prototype and the relationship between those classes.

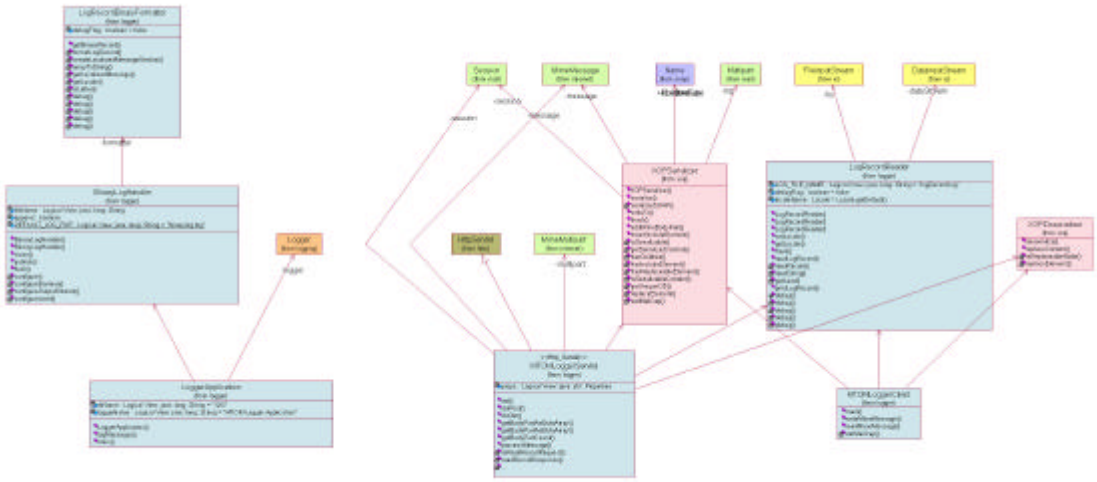

Figure 13 Enhanced Logger Class Diagram

11 Implementation

11.1 MTOM Implementation

The MTOM implementation is provided using an XOP implementation, and a HTTP transport mechanism. The implementation relies heavily on the Java Mail Specification for the Multipart Messages and Java Activation Framework for handling the data content in the messages. In the following sections, a description is provided on the Java Mail specification and the Java beans Activation Framework (JAF) specification. Following this introduction, it is described how the XOP specification is implemented using JAF and Java Mail. Finally the implementation of MTOM based on the XOP implementation is described.

11.1.1 Java Mail Specification

Java Mail API provides a common, uniform API for managing electronic mail. It allows service-providers to provide a standard interface to their standards-based or proprietary messaging systems using the Java programming language. Using this API, applications access message stores, and compose and send messages.

The Java Mail API is composed of a set of abstract classes that model the various pieces of a typical mail system. These classes include,

- `Message` – Abstract class that represents an email message. Java Mail implements the RFC822 and MIME Internet messaging standards. The `MimeMessage` class extends Message class to represent a MIME-style email message. The XOP implementation relies heavily on the Message class.
- `Store` – Abstract class that represents a database of messages maintained by a mail server and grouped by owner. A `Store` uses a particular access protocol.
- `Folder` – Abstract class that provides a way of hierarchically organizing messages. `Folders` can contain messages and other folders. A mail server provides each user with a default folder, and users can typically create and fill subfolders.
- `Transport` – Abstract class that represents a specific transport protocol. A `Transport` object uses a particular transport protocol to send a message.

The Message class defines a set of attributes and content for an email message. The attributes, which are name-value pairs, specify addressing information and define the structure of the message's content including the Content-Type. Messages can contain a single content object or, indirectly, multiple content objects. In either case the content is held by a DataHandler object. The Message class is an abstract class that implements the Part interface.

The UML diagram below describes the Java Mail framework focusing on the Message class and its related classes.

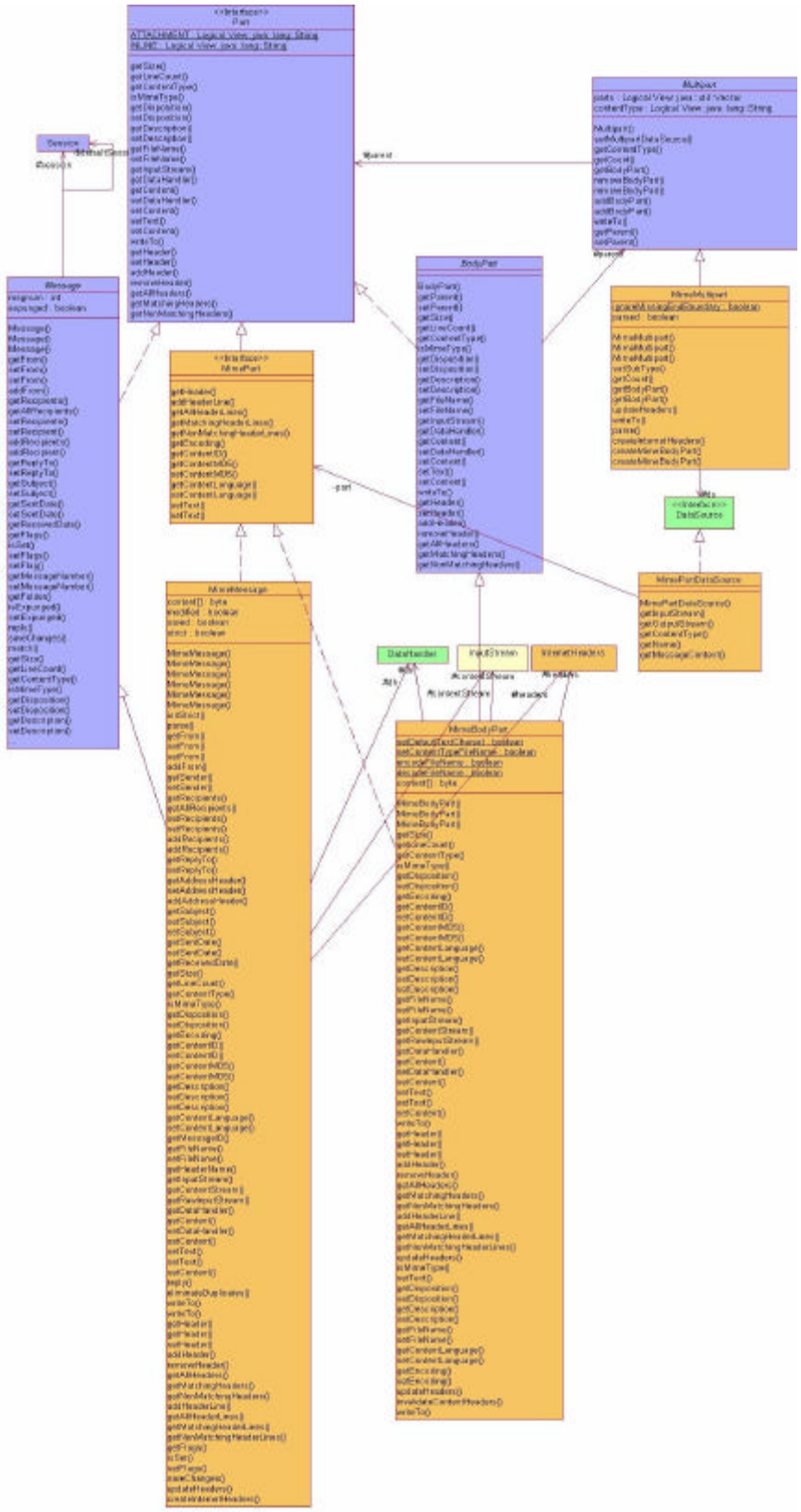

Figure 14 Java Mail Specification

A Message object can contain multiple parts, where each part contains its won set of attributes and content. The content of a Multipart Message is a Multipart object that contains BodyPart objects representing each individual part. The Part interface defines the structural and semantic similarity between the Message class and the BodyPart class. BodyPart is an abstract class that implements the Part interface in order to define the attribute and content body definitions that Part declares.

The Part interface defines a set of standards headers common to most mail systems, specifies the data-type assigned to data comprising a contend block, and defines set and get methods for each of these members. It is the basic data component in the Java Mail API and provides a common interface for both the Message and BodyPart classes. The setDataHandler(DataHandler) method specifies content for a new Part object. Part provides a writeTo() method that writes its byte stream in mail-safe form suitable for transmission. The byte stream is typically an aggregation of the Part attributes and the byte stream for its content.

The Message class adds its own set of attributes to those it inherits from the Part interface. These attributes include the sender and recipient address, the subject, flags and sent and received dates. The Message class also supports non-standard attributes in the form of Headers.

The ContentType attribute specifies the data type of the content, following the MIME typing specification. A MIME type s composed of a primary type that declares the general type of the content subtype that specifies a specific format for the content. A MIME type also includes an optional set of specific parameters.

The Multipart class implements Multipart Messages. A Multipart Message is a Message object where the content-type specifier has been set to *multipart.* Multipart is a container class that contains objects of the BodyPart. A Bodypart object is an instantiation of the Part interface. It contains either a new Multipart container object, or a DataHandler object. A Multipart object can contain other Multipart objects. The content type for each BodyPart element inside a Multipart container should be specified. When a client receives a Multipart object, it first checks the content type of the Multipart container and depending on the type, will call the content type of each of the BodyParts contained it and extracts the data using the appropriate DataHandler object.

11.1.2 JavaBeans Activation Framework

Java Mail relies heavily on the JavaBeans Activation Framework (JAF) to determine the MIME data type, to determine the commands that are available on the data, and to provide a software component corresponding to a particular behavior.

The JAF architecture uses four basic classes for design. They are

- `DataHandler` – The `DataHandler` class provides a consistent interface between the JAF-aware client and other sub systems.

- `DataSource` – The `DataSource` interface encapsulates an object that contains data, and that can return both a stream providing data access, and a string defining the MIME type describing the data. Classes can be implemented for common data sources. The DataSource interface can also be extended to allow per data source user customizations. Once the DataSource is set in the DataHandler, the client can determine the operations available on that data. JAF provides two data source implementations for convenience. These are the FileDataSource to access the content of a file and URLDataSource to access the data held at a URL.
- `CommandMap` – The `CommandMap` interface provides a service that allows customers of its interfaces to determine the commands available on a particular MIME Type as well as an interface to retrieve an object that can operate on an object of a particular MIME Type. The Command Map can generate and maintain a list of available capabilities on a particular data type by a mechanism defined by the implementation of the particular instance of the CommandMap. The CommandMap interface allows developers to develop their own solutions for discovering which commands are available on the system.
- `CommandObject` – Beans extend the `CommandObject` interface in order to interact with JAF services. JAF aware JavaBeans components can directly access their DataSource and DataHandler objects inorder to retrieve the data type and to act on the data.

The UML diagram below shows the architecture of the JavaBeans Activation Framework.

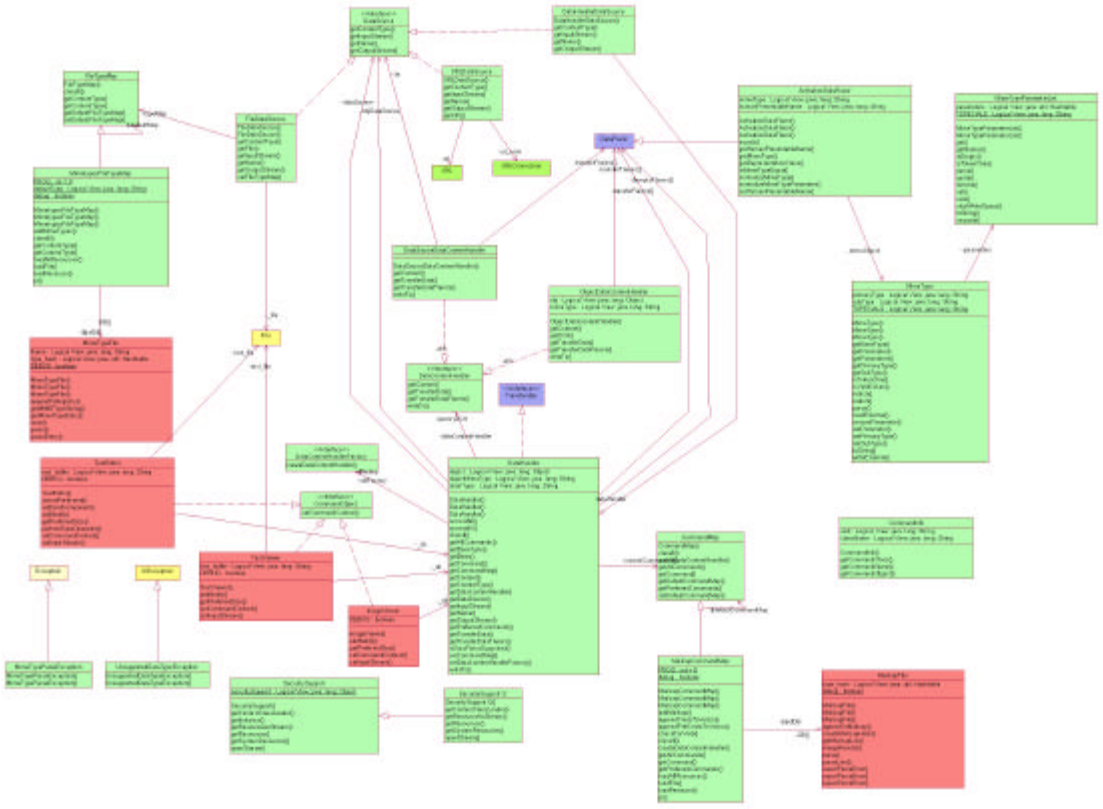

Figure 15 JavaBeans Activation Framework Architecture

11.1.3 XOP Implementation

XOP Implementation is depended on the java mail classes and the java activation classes. The xml binary optimized package is created as a javax.mail.internet.MimeMessage. The content of this mime package is maintained as javax.activation.DataSource objects. A customized DataSource class is created by the name edu.pace.soap.xop.XOPDataSource which functions as the data container for the binary data. If an XML package which is already XOP serialized is again passed to be XOP serialized then an XOPException is thrown. The XOPSerializer class provides the methods for creating and manipulating an XML binary optimized package. The XOPDeserializer class provides the reverse functionality by extracting the XML data and the attached binary data from an XOP serialized MimeMessage. The following UML diagram represents the classes of the XOP package and shows the relationship between the classes and the classes of the associated packages.

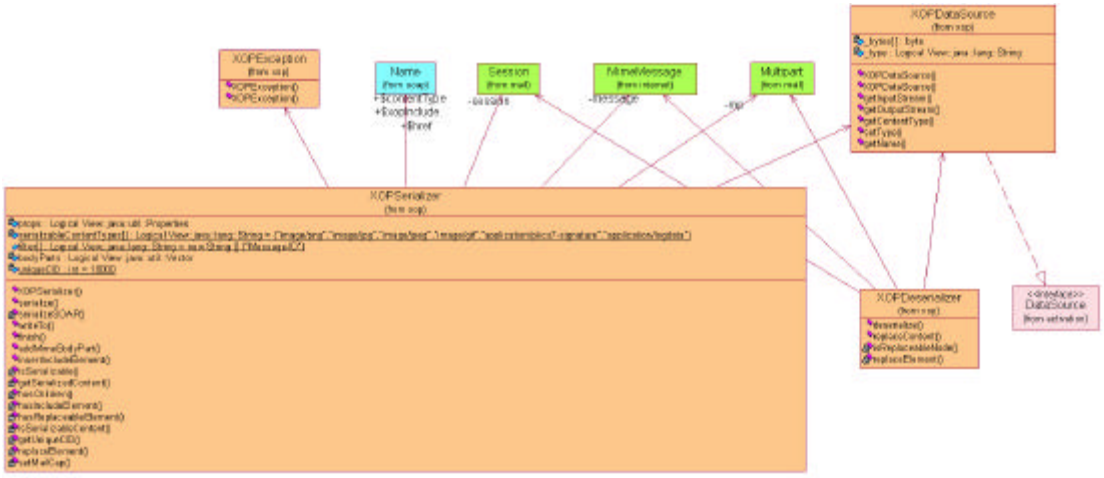

Figure 16 XOP Implementation Class Diagram

11.1.4 MTOM Implementation

MTOM implementation provides an HTTP transport mechanism to the XOP implementation described above. The HTTPServlet class is used to provide the MTOM implementation. The MTOM servlet runs in a HTTPServer and listens for any HTTPRequests coming in. Once it receives a request, it checks the requests input stream to find any XOP serialized MimeMessage. On identifying an XOP serialized MimeMessage, the servlet extracts the root XML part of the MimeMessage. If any binary data is available along with the serialized package, then they are also extracted and the application logic is applied on the data, and the result is sent back to the Client. The MTOM Client waits for the return message from the MTOM Servlet. On receiving a response, the client extracts the input stream of the response and the MimeMessage from the input stream. The MimeMessage is then processed to extract the root xml and other binary data parts as described by the XOP deserialization. The UML diagram describing the classes of the mtom package and the relationship between the classes is shown below.

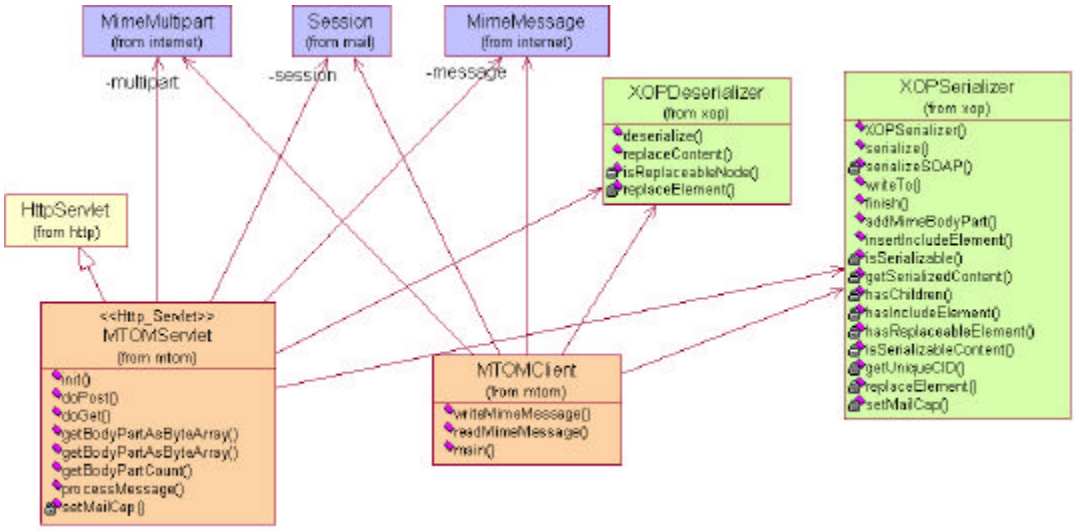

Figure 17 MTOM Implementation Class Diagram

12 Conclusion and Future Work

The XOP and MTOM implementation developed in this project is based on the DOM serialized version of the XML Infoset. A straight forward work is to extend this work to be based on an XML Infoset model similar to AXIOM developed by AXIS 2.0. AXIS 2.0 team is working on developing the MTOM implementation for AXIOM model, which will soon be integrated to the AXIS 2.0 source tree. Another developing area of work will be to integrate DFDL (Data Format Description Language) which is currently being developed in the dfdl-wg gridforum (https://forge.gridforum.org/projects/dfdl-wg/) working group in to the core logger system. The DFDL standards will provide a more efficient mechanism to search through binary logs, and extract the requested data efficiently with requiring parse through binary streams.

References

1. A Logger System based on Web services; B. Horn, H. Balakrishnan, B. T. Maniampadavathu, J. Warnes, D. A. Elko; IBM Systems Journal, Vol 43, No. 4, 2004; http://www.research.ibm.com/journal/sj/434/horn.html
2. SOAP Version 1.2 Specification; http://www.w3c.org/TR/SOAP
    a. Part 0: Primer; June 2003; http://www.w3.org/TR/soap12-part0/
    b. Part 1: Messaging Framework; June 2003; http://www.w3.org/TR/soap12-part1/
    c. Part 2: Adjuncts; June 2003; http://www.w3.org/TR/soap12-part2/
3. XOP W3C Recommendation "XML-binary Optimized Packaging"; M. Gudgin, N. Mendelsohn, M. Nottingham, H. Ruellan; January 2005; http://www.w3.org/TR/xop10/
4. MTOM W3C Recommendation "SOAP Message Transmission Optimization Mechanism"; M. Gudgin, N. Mendelsohn, M. Nottingham, H. Ruellan; January 2005; http://www.w3.org/TR/soap12-mtom/
5. XML 1.0 W3C Recommendation "Extensible Markup Language (Third Edition)"; T.Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau; February 2004; http://www.w3.org/TR/REC-xml
6. XML Schema; http://www.w3.org/XML/Schema; October 2004;
    a. Part 0: Primer; October 2004; http://www.w3.org/TR/xmlschema-0/ ,
    b. Part 1: Structures; October 2004; http://www.w3.org/TR/xmlschema-1/ ,
    c. Part 2: Datatypes; October 2004; http://www.w3.org/TR/xmlschema-2/
7. XML Namespaces W3C Recommendation "Namespaces in XML"; January 1999; http://www.w3.org/TR/REC-xml-names/
8. XML Infoset W3C Recommendation "XML Information Set"; February 2004; http://www.w3.org/TR/xml-infoset/
9. The WS-Resource Framework; March 2004; http://www.globus.org/wsrf/specs/ws-wsrf.pdf
10. WS-Resource Framework Specifications; March 2004; http://www-106.ibm.com/developerworks/library/ws-resource/index.html
11. XML Web Services Basics; R. Wolter; MSDN Library; December 2001; http://msdn.microsoft.com/library/en-us/dnwebsrv/html/webservbasics.asp
12. Web Services Conceptual Architecture (WSCA – 1.0); H. Kreger; May 2001; http://www-306.ibm.com/software/solutions/webservices/pdf/WSCA.pdf
13. WSDL 1.1 W3C Note "Web Services Description Language Version 1.1"; March 2001; http://www.w3.org/TR/wsdl
14. WSDL 2.0 W3C Working Draft "Web Services Description Language Version 2.0"; August 2004
    a. Part 1: Core Language; August 2004; http://www.w3.org/TR/wsdl20/
    b. Part 2: Predefined Extensions; August 2004; http://www.w3.org/TR/wsdl20-extensions
    c. Part 3: Bindings; August 2004; http://www.w3.org/TR/wsdl20-bindings
15. XML SOAP and Binary Data Version 1.0; A. Bosworth, D. Box, M. Gudgin, M. Nottingham, D. Orchard, J Schlimmer; MSDN Library; February 2003; http://msdn.microsoft.com/library/en-us/dnwebsrv/html/infoset_whitepaper.asp

16. The Base 16, Base 32, and Base 64 Data Encodings; RFC 3548; S. Josefsson; July 2003; http://www.faqs.org/rfcs/rfc3548.html
17. Evolution of Web Services Attachments Technologies; C. K. Liu, S. Patil; SAP Developer Network; December 2004; https://www.sdn.sap.com/irj/servlet/prt/portal/prtroot/com.sap.km.cm.docs/library/webas/webservices/Evolution%20of%20Web%20Services%20Attachments%20Technologies.article
18. SwA W3C Note "SOAP Messages with Attachments"; December 2004; http://www.w3.org/TR/SOAP-attachments
19. JSR-000067 SOAP with Attachments API for Java (SAAJ) Specification v1.2 Final Release; October 2003; http://java.sun.com/xml/downloads/saaj.html
20. DIME – Direct Internet Message Encapsulation; H. F. Nielsen, H. Sanders, R. Butek, S. Nash; June 2002; http://msdn.microsoft.com/library/en-us/dnglobspec/html/draft-nielsen-dime-02.txt
21. WS-Attachments; H. F. Nielsen, E. Christensen, J. Farrell; June 2002; http://msdn.microsoft.com/library/en-us/dnglobspec/html/draft-nielsen-dime-soap-01.txt
22. WS-I Attachment Profile 1.0; C. Ferris, A. Karmarkar, C. K. Liu; August 2004; http://www.ws-i.org/Profiles/AttachmentsProfile-1.0.html
23. Proposed Infoset Addendum to SOAP Messages with Attachments; A. Bosworth, D. Box; April 2003; http://www.gotdotnet.com/team/jeffsch/paswa/paswa61.html
24. Building Web Services with Java, Second Edition; S. Graham, D. Davis, S. Simeonov, G. Daniels, P. Brittenham, Y. Nakamura, P. Fremantle, D. Konig and C. Zentner; Sams Publishing; July 2004
25. Web Services, Opaque Data and the Attachments Problem; M. Powell; MSDN Library; June 2004; http://msdn.microsoft.com/library/en-us/dnwebsrv/html/opaquedata.asp
26. Understanding DIME and WS-Attachments; M. Powell; MSDN Library; October 2002; http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/dnarxml/html/dimewsattch.asp
27. SOAP Optimized Serialization Use Cases and Requirements W3C Working Draft; T. Graham, M. Jones, and A. Karmarkar; June 2004; http://www.w3.org/TR/soap12-os-ucr/
28. Resource Representation SOAP Header Block W3C Recommendation; M. Gudgin, Y. Lafon, and A. Karmarkar, January 2005; http://www.w3.org/TR/2005/REC-soap12-rep-20050125/
29. Assigning Media Types to Binary Data in XML W3C Working Draft; Ü. Yalçinalp and A. Karmarkar; November 2004; http://www.w3.org/TR/xml-media-types
30. RFC 2387, The MIME Multipart/Related Content-type; E. Levinson, IETF, August 1998; http://www.ietf.org/rfc/rfc2387.txt
31. RFC 2392, Content-ID and Message-ID Uniform Resource Locators; E. Levinson; IETF, August 1998; http://www.ietf.org/rfc/rfc2392.txt.
32. Java Mail API Specification; September 2000; http://java.sun.com/products/javamail/JavaMail-1.2.pdf

33. JavaBeans Activation Framework Specification; May 1999; http://java.sun.com/products/javabeans/glasgow/JAF-1.0.pdf
34. JSR 047, Java Logging API Specification; http://java.sun.com/j2se/1.4.2/docs/api/java/util/logging/package-summary.html
35. Apache AXIS; http://ws.apache.org/axis/
    a. AXIS Java 1.2; http://ws.apache.org/axis/java/index.html
    b. AXIS 2.0; http://ws.apache.org/axis2/
    c. Implementation of MTOM for AXIOM; http://wiki.apache.org/ws/FrontPage/Axis2/MTOM

Appendix - I

JSR 047 – Java Logging

Java Specification Request (JSR) 047 provides a specification for logging APIs within the Java™ platform. These APIs will be suitable for logging events from within the Java platform and from within Java applications. With this API it is envisaged that:

- It will be possible to enable or disable logging at run-time.
- It will be possible to control logging at a fairly fine granularity, so that logging can be enabled or disabled for specific functionality.
- The logging APIs will allow registration of logging services at run time, so third parties can add new log services.
- It will be possible to provide bridging services that connect the Java logging APIs to existing logging services (e.g. operating system logs).
- Where appropriate, the logging APIs will also support displaying high-priority messages to end users.

The Logging API is available as part of J2SE1.4. The `java.util.logging` package provides the classes and interfaces of the Java ™ 2 platform's core logging facilities. The central goal of the logging APIs is to support maintaining and servicing software at customer sites.

There are four main target uses of the logs:

1. *Problem diagnosis by end users and system administrators*. This consists of simple logging of common problems that can be fixed or tracked locally, such as running out of resources, security failures, and simple configuration errors.
2. *Problem diagnosis by field service engineers.* The logging information used by field service engineers may be considerably more complex and verbose than that required by system administrators. Typically such information will require extra logging within particular subsystems.
3. *Problem diagnosis by the development organization*. When a problem occurs in the field, it may be necessary to return the captured logging information to the original development team for diagnosis. This logging information may be extremely detailed and fairly inscrutable. Such information might include detailed tracing on the internal execution of particular subsystems.
4. *Problem diagnosis by developers*. The Logging APIs may also be used to help debug an application under development. This may include logging information generated by the target application as well as logging information generated by lower-level libraries. Note however that while this use is perfectly reasonable, the logging APIs are not intended to replace the normal debugging and profiling tools that may already exist in the development environment.

The key elements of this package include:

- *Logger*: The main entity on which applications make logging calls. A Logger object is used to log messages for a specific system or application component.

- *LogRecord*: Used to pass logging requests between the logging framework and individual log handlers.
- *Handler*: Exports LogRecord objects to a variety of destinations including memory, output streams, consoles, files, and sockets. A variety of Handler subclasses exist for this purpose. Additional Handlers may be developed by third parties and delivered on top of the core platform.
- *Level*: Defines a set of standard logging levels that can be used to control logging output. Programs can be configured to output logging for some levels while ignoring output for others.
- *Filter*: Provides fine-grained control over what gets logged, beyond the control provided by log levels. The logging APIs support a general-purpose filter mechanism that allows application code to attach arbitrary filters to control logging output.
- *Formatter*: Provides support for formatting LogRecord objects. This package includes two formatters, SimpleFormatter and XMLFormatter, for formatting log records in plain text or XML respectively. As with Handlers, additional Formatters may be developed by third parties.

The Logging APIs offer both static and dynamic configuration control. Static control enables developers to set up a particular configuration and then re-launch the application with the new logging settings. Dynamic control allows for updates to the logging configuration within a currently running program. The APIs also allow for logging to be enabled or disabled for different functional areas of the system.

Applications make logging calls on *Logger* objects. Loggers are organized in a hierarchical namespace and child Loggers may inherit some logging properties from their parents in the namespace. These Logger objects create *LogRecord* objects which are passed to *Handler* objects for publication. Both Loggers and Handlers may use logging *Levels* and (optionally) *Filters* to decide if they are interested in a particular *LogRecord*. When it is necessary to publish a LogRecord externally, a Handler can (optionally) use a *Formatter* to localize and format the message before publishing it to an I/O stream.

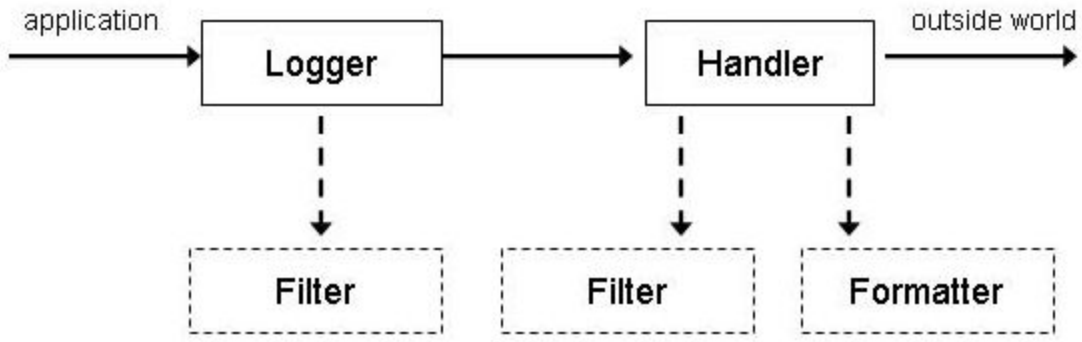

Figure 1 Java Logging Control Flow

Each Logger keeps track of a set of output Handlers. By default all Loggers are configured such that they also send their output to their parent Logger. But Loggers may also be configured to ignore Handlers in their hierarchy.

The Logging API provides isEnabled() method which helps check whether logging is enabled for a particular Logger for any given level. Each logger keeps track of a log level that it is interested in, and discards log requests that are below this level. This check helps avoid the expensive logging operation in situations where it is not required. The Logger pass the generated LogRecord to a Handler which will process the contents, and if Localization and Formatting is required, then only Handler will invoke its attached Formatter to perform these tasks.

Each log message has an associated log *Level*. The Level gives a rough guide to the importance and urgency of a log message. Log level objects encapsulate an integer value, with higher values indicating higher priorities. The `Level` class defines seven standard log levels, ranging from FINEST (the lowest priority, with the lowest value) FINER, FINE, CONFIG, INFO, WARNING and SEVERE (the highest priority, with the highest value).

Loggers are normally named entities, using dot-separated names such as "java.awt". The namespace is hierarchical and is managed by the LogManager. It is recommended that the namespace should typically be aligned with the Java packaging namespace. Loggers keep track of their parent loggers in the logging namespace. A logger's parent is its nearest extant ancestor in the logging namespace. The root Logger (named "") has no parent. Anonymous loggers are all given the root logger as their parent. Loggers may inherit various attributes from their parents in the logger namespace. In particular, a logger may inherit:

- Logging level. If a Logger's level is set to be null then the Logger will use an effective Level that will be obtained by walking up the parent tree and using the first non-null Level.
- Handlers. By default a Logger will log any output messages to its parent's handlers, and so on recursively up the tree.
- Resource bundle names. If a logger has a null resource bundle name, then it will inherit any resource bundle name defined for its parent, and so on recursively up the tree.

The Logger class provides a large set of convenience methods for generating log messages. For convenience, there are methods for each logging level, named after the logging level name. Thus rather than calling "logger.log(Constants.WARNING,..." a developer can simply call the convenience method "logger.warning(..." Java Logging supports two styles of logging methods, the explicit method and the inferred method.

In explicit logging, there are methods that take an explicit source class name and source method name. These methods are intended for developers who want to be able to quickly locate the source of any given logging message. An example of this style is:

```
void warning(String sourceClass, String sourceMethod, String msg);
```

In inferred logging, there are a set of methods that do not take explicit source class or source method names. These are intended for developers who want easy-to-use logging and do not require detailed source information.

```
void warning(String msg);
```

For this second set of methods, the Logging framework will make a "best effort" to determine which class and method called into the logging framework and will add this information into the LogRecord. However, it is important to realize that this automatically inferred information may only be approximate. The latest generation of virtual machines perform extensive optimizations when JITing and may entirely remove stack frames, making it impossible to reliably locate the calling class and method.

J2SE provides the following Handlers:

- *StreamHandler:* A simple handler for writing formatted records to an OutputStream.
- *ConsoleHandler:* A simple handler for writing formatted records to System.err
- *FileHandler:* A handler that writes formatted log records either to a single file, or to a set of rotating log files.
- *SocketHandler:* A handler that writes formatted log records to remote TCP ports.
- *MemoryHandler:* A handler that buffers log records in memory.

J2SE also includes two standard Formatters:

- *SimpleFormatter*: Writes brief "human-readable" summaries of log records.
- *XMLFormatter*: Writes detailed XML-structured information.

There is a global LogManager object that keeps track of global logging information. This includes:

- A hierarchical namespace of named Loggers.
- A set of logging control properties read from the configuration file.

There is a single LogManager object that can be retrieved using the static LogManager.getLogManager() method. This is created during LogManager initialization, based on a system property. This property allows container applications (such as EJB containers) to substitute their own subclass of LogManager in place of the default class.

The logging configuration can be initialized using a logging configuration file that will be read at startup. This logging configuration file is in standard java.util.Properties format. There is a small set of global configuration information. This is specified in the description of the LogManager class and includes a list of root-level Handlers to install during startup.
The initial configuration may specify levels for particular loggers. These levels are applied to the named logger and any loggers below it in the naming hierarchy. The levels are applied in the order they are defined in the configuration file.

The initial configuration may contain arbitrary properties for use by Handlers or by subsystems doing logging. By convention these properties should use names starting with the name of the handler class or the name of the main Logger for the subsystem.

Log messages may need to be localized. Each Logger may have a Resource Bundle name associated with it. The corresponding Resource Bundle can be used to map between raw message strings and localized message strings. Normally localization will be performed by Formatters. As a convenience, the formatter class provides a formatMessage() method that provides some basic localization and formatting support.

Appendix - II

Axis (Apache eXtensible Interaction System)

SOAP processing at the client end and at the server end is done by SOAP processing engines. There are plenty of SOAP processing engines available today. The most popular implementation is the open source implementation from Apache called AXIS. AXIS is the short form for Apache eXtensible Interaction System. AXIS development followed as an evolution for an earlier DOM based SOAP processing engine from Apache called Apace SOAP. AXIS follows the SAX parsing standards compared to the slower in memory DOM implementation followed by Apache SOAP. The main architectural feature of AXIS is the handlers, which is a chain of message processing components which can each processing portions of the message and can be developed separately and can be configured together at deployment time. The handlers are simple java interfaces, which have just one method

```
void invoke(MessageContext context) throws AxisFault
```

When invoked, the handlers execute the code in invoke() method, which could involve reading or writing pieces of a SOAP message, logging access information, checking for the authentication credentials etc. The MessageContext object contains the input SOAP Message, the output SOAP message, and environment properties. During a web service invocation, the MessageContext object is passed from handler to handler. Handlers can be combined together to form chains. There are two types of chains, simple chains and targeted chains. A simple chain is a set of handlers that can be invoked in a preconfigured order. A targeted chain is a collection of three specific handlers, which are: a request handler, a pivot handler and a response handler. The pivot handler is the component which provides the service published by this web service. The request handler works on the preprocessing of the message, like checking for the authentication, and the response handler works on the post-processing of the SOAP message, like logging the number of clients serviced etc.

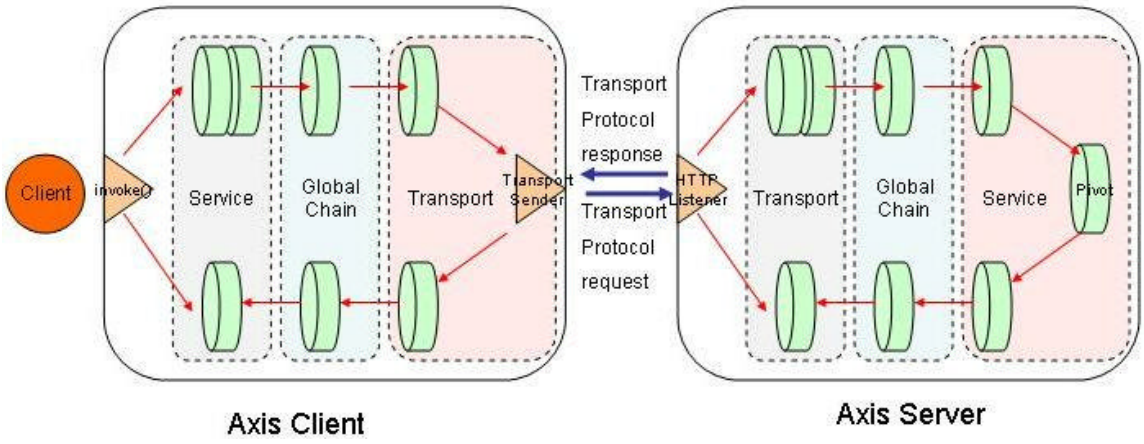

Figure 1 Axis Architecture

On the AXIS server side, a transport listener receives the transport specific messages, and converts them to SOAP messages and sends to the transport specific handlers. The transport specific handlers process the message context and pass the message context to the global chain handlers. The global handlers is the set of handlers which every message processed by this AXIS server should pass through. After the global handlers the message is passed to the targeted handlers where the pivot will provide the specific service. The AXIS client side processing is equivalent to the server side processing, except for the order of the handlers. In the client side, the message will first pass through the service handlers, then through the global chain and finally through the transport chain. The transport chain will add the transport specific properties to the message context and passes the message to the transport sender. The transport sender will send the message to the AXIS server, where the Transport listener will receive these messages.

AXIS 2.0

Axis2 is an effort to redesign and re-implement AXIS on a new Architecture. From the handler model which was the core feature of AXIS 1.0, the new design will move to a pipeline model. The AXIS 1.0 model was based on a SAX parser model, but the new design will be based on a XML pull parser. Another key feature introduced by AXIS2 is a new representation for the SOAP messages called the AXIOM (AXIS Object Model). AXIOM consists of two parts: a complete XML Infoset representation and a SOAP Infoset representation on top of that. The XML Infoset representation provides a JDOM-like simple API but is built on a deferred model via a StAX-based (Streaming API for XML) pull parsing API. A key feature of AXIOM is that it allows one to stop building the XML tree and just access the pull stream directly; thus enabling both maximum flexibility and maximum performance. This approach allows us to support multiple levels of abstraction for consuming and offering Web services: using plain AXIOM, using generated code and statically data-bound data types and so on.

Implementation of MTOM for AXIOM

Axis2 uses the AXIOM implementation for MTOM. The following section introduces the details of this implementation.

De-Serializer

The Transport module identifies the MTOM messages by looking at the “Content-Type” and “type” header fields in the message and chooses between MTOMStAXBuilder and StAXSOAPModelBuilder.

```
Content-Type: Multipart/Related;
boundary=MIME_boundary;
Type=”application/xop+xml”;
```

MTOMStAXSOAPBuilder assumes that the Input Stream provided to it contains an MTOM encoded MIME message. It passes that Input stream to MTOMBuilder. MTOMbuilder extracts the root mime part which contains the XOP soap message and returns a XMLStreamReader(parser) built using that SOAP message to the

MTOMStAXSOAPModelBuilder. This builder deferred builds the AXIOM using that parser.

The following figure shows the flow of how an AXIOM Object is constructed from the MTOM message coming in a Transport.

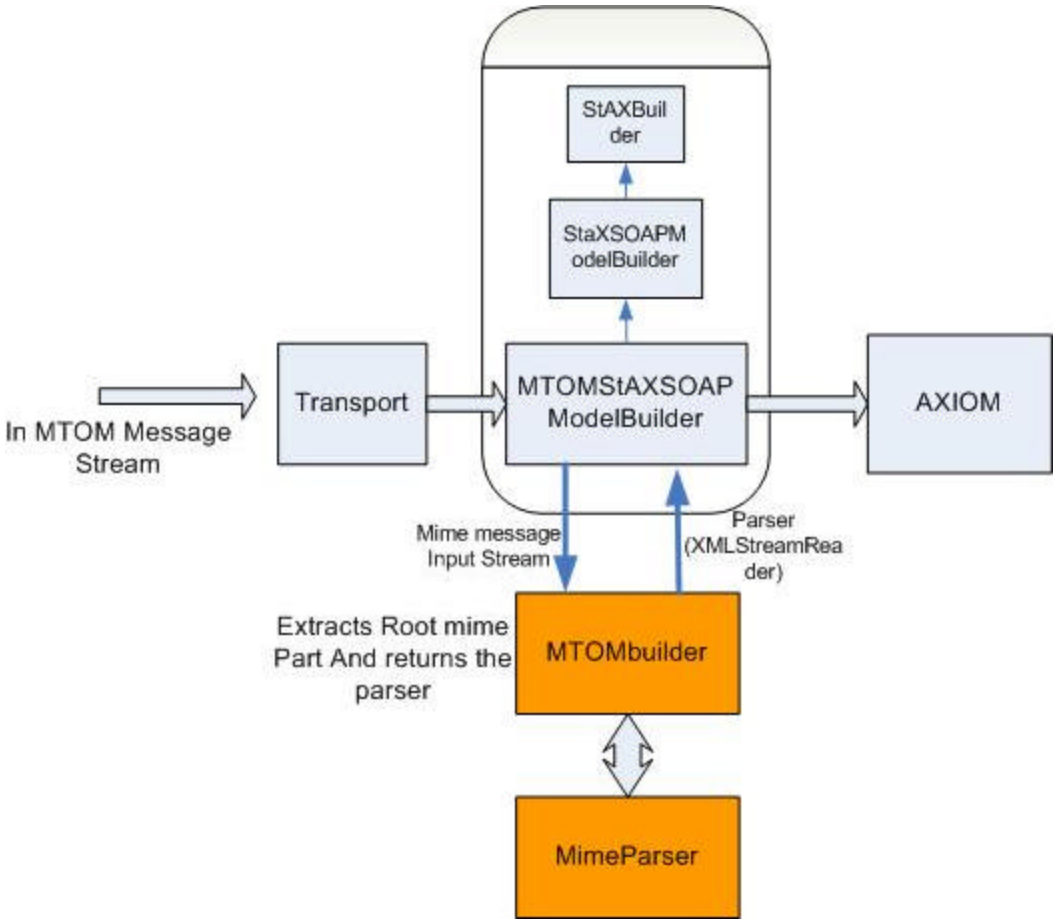

Figure 2 AXIOM implementation for MTOM deserializer

When the builder comes across an xop:include element with the correct namespace, it creates an OMBlob node in the AXIOM. This OMBlob node keeps the "CID" value read from xop:include element and a reference to the MTOMbuilder in it.

```
<xop:include xmlns:xop='http://www.w3.org/2004/08/xop/include'
               href='cid:http://example.org/me.png'/>
```

OMBlob achieves deferred building by reading the MIME parts referenced by it only when a user requests the actual binary data value stored in it. When a request comes for the binary data, OMBlob use its MTOMBuilder reference and 'CID' to obtain the correct Data Handler.

The following picture shows how the AXIOM Tree is constructed from an MTOM message.

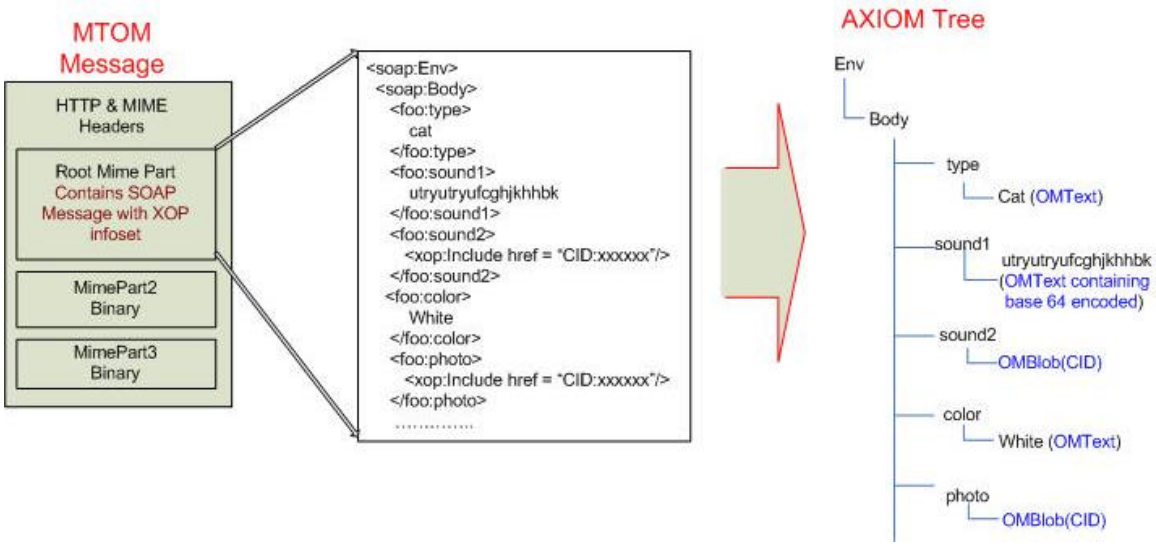

Figure 3 Building an AXIOM Tree from MTOM message

Serializer

MTOMXMLStreamReader has to be used when the message needs to be optimized using MTOM. This will ensure that OMBlobs in the AXIOM will be attached to the message by MTOM optimizing. This MTOMXMLStreamReader buffers the output stream taken from the internally initialized XMLStreamReader. When this serializes an OMBlob it generates an 'CID' value and writes the <xop:include> element using it. At the same time it stores that 'CID' & a reference to the OMBlob in a hash map.

The 'complete()' method of the MTOMXMLStreamReader has to be called After serializing the whole document. At that point the MTOMXMLStreamReader generates the root mime part using the buffered outputStream data from the XMLStreamReader. Also it generates additional mime parts for all the OMBlob & CID pairs in the HashMap and creates a Mime message containing all.

Content in the OMBlob will be serialized as base64encoded strings when plain XMLStreamReader is used to serialize an AXIOM containing OMBlobs. This can be used when it is decided not to use MTOM optimization. Other than that the binary attachments which are not qualified to be optimized will remain as base 64 strings in OMText nodes.

Appendix III - SOAP MTOM Implementation API

Packages	
edu.pace.soap.logger	This package contains the classes which implement the prototype for Message optimized Logger System.
edu.pace.soap.mtom	This package contains the classes which implements the **SOAP Message Optimization specification** provided by w3c
edu.pace.soap.xop	This package contains the classes which implements the **XML binary optimized packaging specification** provided by w3c.
edu.pace.util	This package contains the utility classes used for the Message Optimzed Logger System.

edu.pace.soap.xop

This package contains the classes which implements the **XML binary optimized packaging specification** provided by w3c.

See:

Description

Class Summary	
XOPDataSource	Creates a XOPDataSource.
XOPDeserializer	Creates a Reconstituted XML Infoset from a XOP Package.
XOPSerializer	Creates XOP Package from an Original XML Infoset.

Exception Summary	
XOPException	Creates a XOPException.

Package edu.pace.soap.xop Description

This package contains the classes which implements the **XML binary optimized packaging specification** provided by w3c.
The XOPSerializer class provides the methods to create XOP Package from an Original XML Infoset.
The XOPDeserializer class provides the methods to create a reconstituted XML Infoset from a XOP Package.
The XOPException class is thrown while attempting to serialize an already serialized

class (ie an XML Infoset which contains "xop:Include" [element information item].
The XOPDataSource contains the binary data of the attachments and provides an efficient way for the implemented DataHandlers to access the contained binary data.
Note: This impelementation is based on the DOM serialized form of XML Infoset

edu.pace.soap.xop

Class XOPSerializer

`java.lang.Object`
  └ **`edu.pace.soap.xop.XOPSerializer`**

---

public class **XOPSerializer**
extends Object

Creates XOP Package from an Original XML Infoset. **Creating XOP Packages**

1. Ensure that the original XML Inforset contains no *element information item* with a [namespace name] of "http://www.w3.org/2004/08/xop/include" and a [local name] of `include`. As discussed in **2 XOP Infoset Constructs** . XML Infosets with such *element information items* cannot be represented using XOP.
2. Create an empty package.
3. Identify within the original XML Infoset the *element information items* to be optimized. To be optimized, the characters comprising the [children] of the *element information item* MUST be in the canonical form of `xs:base64Binary` (see **[XML Schema Part 2]3.2.16 base64Binary**) and MUST NOT contain any whitespace characters, preceding, inline with or following the non-whitespace content.
4. Create a XOP Infoset which is a copy of the Original XOP Infoset, but with the [children] of each *element information item* identified in the previous step replaced by a `xop:Include` *element information item* (see **2.1 xop:Include element information item**) construcuted as follows:
    1. Transform the replaced characters into binary data by processing them as base64-encoded data.
    2. Serialize the binary data into a new part of the package, with appropriate metadata corresponding to the [normalized value] of the `href` *attribute information item* of the `xop:Include` *element information item* (see **2.2 href attribute information item**).
    3. If the *element information item* being optimized (i.e. the [parent] of the newly inserted `xop:Include` element information item) has a `xmlmime:contentType` *attribute information item* , its value SHOULD be reflected appropriately in the metadata for the part.
5. Serialize the resulting XOP Infoset into the package using any W3C reccomendation-level version of XML (e.g[XML 1.0], [XML 1.1]) and identify it as the root part according to the packaging mechanism's convention, labelling it with the application/xop+xml media type, as described in **5 Identifying XOP Dcouments**.

This class provides a convinience method called serialize which takes in a SOAPEnvelope and an OutputStream and XOPSerializes the base64 encoded data in the SOAPEnvelope and write it to the OutputStream parameter.

This class also provides the methods which will help create the XOP Serialized object as individual steps. The methods for the individual creation of the XOP Serialized message should be called in the following order

1. insertIncludeElement()
2. addBodyPart()
3. finish()
4. writeTo()

Author:
Biju T Maniampadavathu

Field Summary

`static javax.xml.soap.Name`	**`contentType`** contentType contains the QName for include "http://www.w3.org/2004/06/xmlmime"
`static String[]`	**`filter`**
`static javax.xml.soap.Name`	**`href`** contentType contains the QName for include "href"
`static javax.xml.soap.Name`	**`xopInclude`** xopInclude contains the QName for include "http://www.w3.org/2004/08/xop/include"

Constructor Summary

`XOPSerializer``()` Default Constructor.

Method Summary

`void`	**`addMimeBodyPart`**`(byte[] data, String cType, String cid)` The addMimeBodyPart method creates a MimeBodyPart with the binary data ContentType and Content-ID and adds it to the MimeMessage.
`void`	**`finish`**`(javax.xml.soap.SOAPEnvelope env)`
`String`	**`insertIncludeElement`**`(javax.xml.soap.SOAPElement element)`

	The insertIncludeElement method creates a new Include element and adds it to the parameter SOAPElement.
static MimeMessage	**serialize**(javax.xml.soap.SOAPEnvelope env, OutputStream out) The serialize() method is a convinience method to xop serialize base64 encoded data in a SOAPEnvelope.
void	**writeTo**(OutputStream out) This method should be called if the application is handling the logic of creating a XOPSerialized outputstream.

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

xopInclude

```
public static final javax.xml.soap.Name xopInclude
```

xopInclude contains the QName for include "http://www.w3.org/2004/08/xop/include"

---

contentType

```
public static final javax.xml.soap.Name contentType
```

contentType contains the QName for include "http://www. w3.org/2004/06/xmlmime"

---

href

```
public static final javax.xml.soap.Name href
```

contentType contains the QName for include "href"

---

filter

```
public static String[] filter
```

Constructor Detail

XOPSerializer

```
public XOPSerializer()
```

Default Constructor.

Method Detail

serialize

```
public static MimeMessage serialize(javax.xml.soap.SOAPEnvelope env,
                                     OutputStream out)
                              throws XOPException
```

The serialize() method is a convinience method to xop serialize base64 encoded data in a SOAPEnvelope. This method takes in a SOAPEnvelope and OutputStream as parameters, and serialize the base64 encoded data in the SOAPEnvelope to binary stream and write that stream as a MimeBodypart in a MimeMultipart object of a MimeMessage into the provided output stream. The base64 element which is serialized is replaced with an include element which contains a ContentID URI which refers to the binary data in the output stream.

Throws:

XOPException

---

writeTo

```
public void writeTo(OutputStream out)
             throws IOException,
                    MessagingException
```

This method should be called if the application is handling the logic of creating a XOPSerialized outputstream. This method should be called as the last step of the individual processing of the XOP Serialization. This method will write the MIME message to the provided output stream.

Throws:

IOException
MessagingException

---

finish

```
public void finish(javax.xml.soap.SOAPEnvelope env)
            throws MessagingException
```

Throws:

MessagingException

---

addMimeBodyPart

```
public void addMimeBodyPart(byte[] data,
                            String cType,
                            String cid)
                     throws MessagingException
```

The addMimeBodyPart method creates a MimeBodyPart with the binary data ContentType and Content-ID and adds it to the MimeMessage.

Throws:

MessagingException

---

insertIncludeElement

```
public String insertIncludeElement(javax.xml.soap.SOAPElement element)
                            throws javax.xml.soap.SOAPException
```

The insertIncludeElement method creates a new Include element and adds it to the parameter SOAPElement. It creates a new Content-ID and adds it as a child text element for the Include element and finally returns the created Content-ID.

Throws:

`javax.xml.soap.SOAPException`

---

edu.pace.soap.xop

Class XOPDeserializer

`java.lang.Object`
  └ **`edu.pace.soap.xop.XOPDeserializer`**

---

public class **XOPDeserializer**
extends Object

Creates a Reconstituted XML Infoset from a XOP Package.

To create a Reconstituted XML Infoset from a XOP Package:

1. Construct an XML Infoset by parsing the root part of the package as an XML document. The document must be parsed according to the level of the XML Recommendation identified by the XML declaration of that document. If no XML declaration is present, then the document MUST be parsed per [XML 1.0].
2. Using that XML Infoset, for each *element information item* , E, which has, as the sole member of its [children] property, a `xop:Include` *element information item* (as defined in **2.1 xop:Include element information item**):
    1. Locate the part of the package corresponding to the URI in the `href` *attribute information item* of the `xop:Include` *element information item* (i.e, corresponding to the URI encoded in the *attribute information item's* [normalized value]).
    2. Replace the `xop:Include` *element information item* that appears in the [children] property of E with *character information items* representing the canonical base64 encoding of the entity body of the identified package part (i.e, effectively replace the `xop:Include` *elementinformation item* with the data reconstructed from the package part.

Author:

Biju T Maniampadavathu

Constructor Summary

`XOPDeserializer``()`

Method Summary

static Document	**deserialize**(MimeMessage message)
void	**replaceContent**(Node dom, MimeMultipart mpart)

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

XOPDeserializer

```
public XOPDeserializer()
```

Method Detail

deserialize

```
public static Document deserialize(MimeMessage message)
```

---

replaceContent

```
public void replaceContent(Node dom,
                           MimeMultipart mpart)
                    throws MessagingException,
                           IOException
```

Throws:
MessagingException
IOException

---

edu.pace.soap.xop

Class XOPDataSource

java.lang.Object
  └ **edu.pace.soap.xop.XOPDataSource**

All Implemented Interfaces:
DataSource

---

public class **XOPDataSource**
extends Object
implements DataSource

Creates a XOPDataSource.

XOPDataSource is a DataSource that contains body parts of an XOP package. This allows "xop aware" DataContentHandlers to be implemented more efficiently by being aware of such DataSources and using the appropriate methods to access BodyParts.

Author:
Biju T Maniampadavathu

Constructor Summary

XOPDataSource(byte[] bytes) Constructor, that constructs a DataSource from a byte[]
XOPDataSource(byte[] bytes, String type) Constructor, that constructs a DataSource from a byte[] and contentType

Method Summary

String	**getContentType**() Returns the content-type of this DataSource.
InputStream	**getInputStream**() Returns an input stream from this part.
String	**getName**() DataSource method to return a name.
OutputStream	**getOutputStream**() The getOutputStream method is not supported for this data source.
void	**setType**(String type) Sets the contentType of the data source.

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

XOPDataSource

```
public XOPDataSource(byte[] bytes,
                     String type)
```

Constructor, that constructs a DataSource from a byte[] and contentType

---

XOPDataSource

```
public XOPDataSource(byte[] bytes)
```

Constructor, that constructs a DataSource from a byte[]

Method Detail

getInputStream

```
public InputStream getInputStream()
                            throws IOException
```

Returns an input stream from this part.
The returned input stream is a decoded stream of bytes.

Specified by:

getInputStream in interface DataSource

Returns:

InputStream

Throws:

IOException

---

getOutputStream

```
public OutputStream getOutputStream()
                              throws IOException
```

The getOutputStream method is not supported for this data source.
Any attempt to invoke this method will throw an IOException.

Specified by:

getOutputStream in interface DataSource

Throws:

IOException

---

getContentType

```
public String getContentType()
```

Returns the content-type of this DataSource.

Specified by:

getContentType in interface DataSource

Returns:

String

---

setType

```
public void setType(String type)
```

Sets the contentType of the data source.
This method provides a convinience mechanism to set the contentType for this data source object.

---

getName

```
public String getName()
```

DataSource method to return a name.
This data source will return "XOPDataSource".

Specified by:
`getName` in interface `DataSource`
Returns:
String

---

edu.pace.soap.xop
Class XOPException

`java.lang.Object`
  └ `java.lang.Throwable`
    └ `java.lang.Exception`
      └ **`edu.pace.soap.xop.XOPException`**

All Implemented Interfaces:
Serializable

---

public class **XOPException**
extends Exception

Creates a XOPException.

XOPException is thrown while trying to construct a serialized XOP package from an XML infoset which already contains a `xop:Include` *element information item* (as defined in **2.1 xop:Include element information item**

Author:
Biju T Maniampadavathu
See Also:
Serialized Form

Constructor Summary

`XOPException``()`
`XOPException``(String reason)`

Methods inherited from class java.lang.Throwable
`fillInStackTrace`, `getCause`, `getLocalizedMessage`, `getMessage`, `getStackTrace`, `initCause`, `printStackTrace`, `printStackTrace`, `printStackTrace`, `setStackTrace`, `toString`

Methods inherited from class java.lang.Object
`equals`, `getClass`, `hashCode`, `notify`, `notifyAll`, `wait`, `wait`, `wait`

Constructor Detail

XOPException

```
public XOPException()
```

XOPException

```
public XOPException(String reason)
```

Package edu.pace.soap.mtom

Class Summary	
MTOMClient	
MTOMServlet	The MTOMServlet class provides a HTTP Transport mechanism for the XOP Serialized packets.

Package edu.pace.soap.xop Description

This package contains the classes which implements the **SOAP Message Optimization specification** provided by w3c

---

edu.pace.soap.mtom

Class MTOMServlet

```
java.lang.Object
  └ javax.servlet.GenericServlet
      └ javax.servlet.http.HttpServlet
          └ edu.pace.soap.mtom.MTOMServlet
```

All Implemented Interfaces:

Serializable, Servlet, ServletConfig

---

public class **MTOMServlet**
extends HttpServlet

The MTOMServlet class provides a HTTP Transport mechanism for the XOP Serialized packets.

Author:

Biju T Maniampadavathu

See Also:

Serialized Form

Constructor Summary

MTOMServlet()	

Method Summary

void	**doGet**(HttpServletRequest request, HttpServletResponse response)
void	**doPost**(HttpServletRequest request, HttpServletResponse response) Handles a POST request from a client.
byte[]	**getBodyPartAsByteArray**(int index)
byte[]	**getBodyPartAsByteArray**(String cid)
int	**getBodyPartCount**()
void	**init**(ServletConfig config) Initialization
void	**processMessage**(HttpServletRequest request, HttpServletResponse response)

Methods inherited from class javax.servlet.http.HttpServlet

service

Methods inherited from class javax.servlet.GenericServlet

destroy, getInitParameter, getInitParameterNames, getServletConfig, getServletContext, getServletInfo, getServletName, init, log, log

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

MTOMServlet

```
public MTOMServlet()
```

Method Detail

init

```
public void init(ServletConfig config)
          throws ServletException
```

Initialization

Throws:

ServletException

---

doPost

```
public void doPost(HttpServletRequest request,
                   HttpServletResponse response)
            throws ServletException,
                   IOException
```

Handles a POST request from a client. The request is assumed to contain a SOAP message with the HTTP binding.

Throws:

ServletException
IOException

---

doGet

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
           throws ServletException,
                  IOException
```

Throws:

ServletException
IOException

---

getBodyPartAsByteArray

```
public byte[] getBodyPartAsByteArray(String cid)
                              throws MessagingException,
                                     IOException
```

Throws:

MessagingException
IOException

---

getBodyPartAsByteArray

```
public byte[] getBodyPartAsByteArray(int index)
                              throws MessagingException,
                                     IOException
```

Throws:

MessagingException
IOException

---

getBodyPartCount

```
public int getBodyPartCount()
                    throws MessagingException
```

Throws:
MessagingException

---

processMessage

```
public void processMessage(HttpServletRequest request,
                           HttpServletResponse response)
                    throws ServletException,
                           IOException
```

Throws:
ServletException
IOException

---

edu.pace.soap.mtom

Class MTOMClient

java.lang.Object
  └ **edu.pace.soap.mtom.MTOMClient**

---

public class **MTOMClient**
extends Object

Constructor Summary

MTOMClient()

Method Summary

static void	**main**(String[] args)
static void	**readMimeMessage**(InputStream in)
static void	**writeMimeMessage**(OutputStream out)

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

MTOMClient

```
public MTOMClient()
```

Method Detail

main

```
public static void main(String[] args)
```

---

writeMimeMessage

```
public static void writeMimeMessage(OutputStream out)
                             throws Exception
```

Throws:
Exception

---

readMimeMessage

```
public static void readMimeMessage(InputStream in)
                            throws Exception
```

Throws:
Exception

Package edu.pace.soap.logger

Class Summary

BinaryLogHandler	The `BinaryLogHandler` writes to the specified log file, which can be configured through the properties file, or defaults to the binarylog.log file The `LogRecordBinaryFormatter` class is used for formatting the LogRecords to the binary format before persisting to the file.
LoggerApplication	The LoggerApplication class is a sample class which creates an instance of java.util.logging.Logger and logs messages to that logger
LogRecordBinaryFormatter	The LogRecordBinaryFormatter class, converts the LogRecord into a byte array.
LogRecordReader	The LogRecordReader class is responsible for reading the binary LogRecord and reconstituting the LogRecord object.
MTOMLoggerClient	
MTOMLoggerServlet	The MTOMServlet class provides a HTTP Transport mechanism for the XOP Serialized packets.

edu.pace.soap.logger

Class BinaryLogHandler

java.lang.Object
  └ java.util.logging.Handler
    └ **edu.pace.soap.logger.BinaryLogHandler**

---

public class **BinaryLogHandler**
extends Handler

The `BinaryLogHandler` writes to the specified log file, which can be configured through the properties file, or defaults to the binarylog.log file

The `LogRecordBinaryFormatter` class is used for formatting the LogRecords to the binary format before persisting to the file.

Configuration: By default each `BinaryLogHandler` is initialized using the following `LogManager` configuration properties. If properties are not defined (or have invalid values) then the specified default values are used.

- edu.pace.soap.logger.BinaryLogHandler.level specifies the default level for the `Handler` (defaults to `java.util.logging.Level.ALL`).
- edu.pace.soap.logger.BinaryLogHandler.file specifies a name for the output file. (defaults to "binarylog.log").
- com.ibm.logging.icl.ODIMessageLogHandler.append specifies whether the FileHandler should append onto any existing files (defaults to false).

Author:
Biju T Maniampadavathu

Constructor Summary

`BinaryLogHandler``()` Create a `BinaryLogHandler`, using only `LogManager` properties (or their defaults).
`BinaryLogHandler``(String fileName, boolean append)` Construct a `BinaryLogHandler` using a specified fileName and append property.

Method Summary

`void`	**`close`**`()` Close this output stream.
`void`	**`flush`**`()` Flush any buffered messages.

void	**publish**(LogRecord record) Format and publish a LogRecord.

Methods inherited from class java.util.logging.Handler
getEncoding, getErrorManager, getFilter, getFormatter, getLevel, isLoggable, setEncoding, setErrorManager, setFilter, setFormatter, setLevel

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

BinaryLogHandler

```
public BinaryLogHandler()
                  throws IOException
```

Create a BinaryLogHandler, using only LogManager properties (or their defaults).

Throws:

IOException - if we are unable to connect to the FileOutputStream.

BinaryLogHandler

```
public BinaryLogHandler(String fileName,
                        boolean append)
                  throws IOException
```

Construct a BinaryLogHandler using a specified fileName and append property. The BinaryLogHandler is configured based on LogManager properties (or their default values) except that the given log fileName and boolean append arguments are used.

Parameters:

fileName - target log file.

append - append to logfile or not.

Throws:

IllegalArgumentException - if fileName is invalid.

IOException - if we are unable to connect to the FileOutputStream.

Method Detail

close

```
public void close()
```

Close this output stream.

---

publish

`public void publish(LogRecord record)`

Format and publish a `LogRecord`.

Parameters:

`record` - description of the log event

---

flush

`public void flush()`

Flush any buffered messages.

---

edu.pace.soap.logger

Class LogRecordBinaryFormatter

`java.lang.Object`
  └ **`edu.pace.soap.logger.LogRecordBinaryFormatter`**

---

public class **LogRecordBinaryFormatter**
extends Object

The LogRecordBinaryFormatter class, converts the LogRecord into a byte array. This byte array is persisted to the log repository. The binary logs are read back and returned as byte array type to the requesting appliction

Author:

Biju T Maniampadavathu

Constructor Summary

`LogRecordBinaryFormatter()`

Method Summary

`ByteVector`	**`getBinaryRecord`**`(LogRecord record)` Format the given LogRecord to the binary XMFFormat

Methods inherited from class java.lang.Object
`equals`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`

Constructor Detail

LogRecordBinaryFormatter

`public LogRecordBinaryFormatter()`

Method Detail

getBinaryRecord

`public ByteVector getBinaryRecord(LogRecord record)`

Format the given LogRecord to the binary XMFFormat

Parameters:

`record` - LogRecord the log record to be formatted

Returns:

String a formatted log record

---

edu.pace.soap.logger

Class LogRecordReader

`java.lang.Object`

└ **`edu.pace.soap.logger.LogRecordReader`**

---

public class **LogRecordReader**
extends Object

The LogRecordReader class is responsible for reading the binary LogRecord and reconstituting the LogRecord object.

Author:

Biju T Maniampadavathu

Constructor Summary

`LogRecordReader``()` default constructor
`LogRecordReader``(byte[] data, String locale)` Constructor which takes a byte array parameter and creates the binary data stream from it.
`LogRecordReader``(String fileName)` Constructor which takes in the fileName as a String and creates the binary data stream from the file

Method Summary

`Locale`	**`getLocale`**`()` Returns the locale object for this reader.
`static void`	**`main`**`(String[] args)` Main method.

void	**printLogRecord**(String msg, LogRecord logRecord) Print LogRecord method
Vector	**readLogRecord**() This method reads the InputStream and converts the byte array to a LogRecord.
void	**setLocale**(String locale) Sets the locale for this reader.

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

LogRecordReader

```
public LogRecordReader()
```

default constructor

LogRecordReader

```
public LogRecordReader(String fileName)
```

Constructor which takes in the fileName as a String and creates the binary data stream from the file

---

LogRecordReader

```
public LogRecordReader(byte[] data,
                       String locale)
```

Constructor which takes a byte array parameter and creates the binary data stream from it.

Method Detail

setLocale

```
public void setLocale(String locale)
```

Sets the locale for this reader. The parameter string should be of the format xx_YY, where xx is the two letter code for the language and YY is the two letter code for the Country.

---

getLocale

```
public Locale getLocale()
```

Returns the locale object for this reader.

---

main

```
public static void main(String[] args)
```

Main method.

Parameters:

`args` -

---

readLogRecord

```
public Vector readLogRecord()
```

This method reads the InputStream and converts the byte array to a LogRecord. The converted LogRecords are added to a Vector and the vector is returned.

---

printLogRecord

```
public void printLogRecord(String msg,
                           LogRecord logRecord)
```

Print LogRecord method

Parameters:

`logRecord` -

`msg` -

---

edu.pace.soap.logger

Class LoggerApplication

`java.lang.Object`
  └ **`edu.pace.soap.logger.LoggerApplication`**

---

public class **LoggerApplication**
extends Object

The LoggerApplication class is a sample class which creates an instance of java.util.logging.Logger and logs messages to that logger

Author:

Biju T Maniampadavathu

Constructor Summary

`LoggerApplication`()	

Method Summary

void	**`logMessages`**() Log messages to the logger
static void	**`main`**(String[] args)

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

LoggerApplication

```
public LoggerApplication()
```

Method Detail

logMessages

```
public void logMessages()
```

Log messages to the logger

---

main

```
public static void main(String[] args)
```

---

edu.pace.soap.logger

Class MTOMLoggerClient

```
java.lang.Object
  └ edu.pace.soap.logger.MTOMLoggerClient
```

---

public class **MTOMLoggerClient**
extends Object

Constructor Summary

MTOMLoggerClient()

Method Summary

static void	**main**(String[] args)
static void	**readMimeMessage**(InputStream in, String locale)
static void	**writeMimeMessage**(OutputStream out)

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

MTOMLoggerClient

```
public MTOMLoggerClient()
```

Method Detail

main

```
public static void main(String[] args)
```

---

writeMimeMessage

```
public static void writeMimeMessage(OutputStream out)
                             throws Exception
```

Throws:
Exception

---

readMimeMessage

```
public static void readMimeMessage(InputStream in,
                                   String locale)
                            throws Exception
```

Throws:
Exception

---

edu.pace.soap.logger

Class MTOMLoggerServlet

java.lang.Object
  └ javax.servlet.GenericServlet
    └ javax.servlet.http.HttpServlet
      └ **edu.pace.soap.logger.MTOMLoggerServlet**

All Implemented Interfaces:
Serializable, Servlet, ServletConfig

---

public class **MTOMLoggerServlet**
extends HttpServlet

The MTOMServlet class provides a HTTP Transport mechanism for the XOP Serialized packets.

Author:

Biju T Maniampadavathu

See Also:

Serialized Form

Constructor Summary

MTOMLoggerServlet()

Method Summary

void	**doGet**(HttpServletRequest request, HttpServletResponse response)
void	**doPost**(HttpServletRequest request, HttpServletResponse response) Handles a POST request from a client.
byte[]	**getBodyPartAsByteArray**(int index)
byte[]	**getBodyPartAsByteArray**(String cid)
int	**getBodyPartCount**()
void	**init**(ServletConfig config) Initialization
void	**processMessage**(HttpServletRequest request, HttpServletResponse response)

Methods inherited from class javax.servlet.http.HttpServlet

service

Methods inherited from class javax.servlet.GenericServlet

destroy, getInitParameter, getInitParameterNames, getServletConfig, getServletContext, getServletInfo, getServletName, init, log, log

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

MTOMLoggerServlet

```
public MTOMLoggerServlet()
```

Method Detail

init

```
public void init(ServletConfig config)
          throws ServletException
```

Initialization

Throws:

ServletException

---

doPost

```
public void doPost(HttpServletRequest request,
                   HttpServletResponse response)
            throws ServletException,
                   IOException
```

Handles a POST request from a client. The request is assumed to contain a SOAP message with the HTTP binding.

Throws:

ServletException
IOException

---

doGet

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
           throws ServletException,
                  IOException
```

Throws:

ServletException
IOException

---

getBodyPartAsByteArray

```
public byte[] getBodyPartAsByteArray(String cid)
                              throws MessagingException,
                                     IOException
```

Throws:

MessagingException
IOException

---

getBodyPartAsByteArray

```
public byte[] getBodyPartAsByteArray(int index)
                              throws MessagingException,
                                     IOException
```

Throws:

MessagingException

IOException

---

getBodyPartCount

```
public int getBodyPartCount()
                    throws MessagingException
```

Throws:
MessagingException

---

processMessage

```
public void processMessage(HttpServletRequest request,
                           HttpServletResponse response)
                    throws ServletException,
                           IOException
```

Throws:
ServletException
IOException

Package edu.pace.util

Class Summary	
ByteVector	A dynamically extensible array of bytes.
SOAPUtils	Utility methods for SOAP processing.
XMLUtils	Utility methods for XML processing.

---

edu.pace.util

Class ByteVector

java.lang.Object
  └ **edu.pace.util.ByteVector**

---

public final class **ByteVector**
extends Object

A dynamically extensible array of bytes. This class is an efficient implementation of a DataOutputStream over a ByteArrayOutputStream, with more granular access to data content.

Field Summary	
byte[]	**data** The content of this array.

int	**length** Actual number of bytes in this array.

Constructor Summary

ByteVector() Constructs a new ByteVector with a default initial size.
ByteVector(int initialSize) Constructs a new ByteVector with the given initial size.

Method Summary

ByteVector	**copyByteArray**(byte[] b, int destOff, int len)
byte[]	**getValidData**() This method gets the effective data from the ByteVector and return it.
ByteVector	**putByte**(int b) Puts a byte into this byte array.
ByteVector	**putByteArray**(byte[] b, int srcOff, int destOff, int len) Puts an array of bytes into this byte array.
ByteVector	**putChar**(char c) Puts a char into this byte vector.
ByteVector	**putChars**(String s) Puts a String as a UTF String into this byte array.
ByteVector	**putDouble**(double d) Puts a double into this byte array.
ByteVector	**putFloat**(float f) Puts a float into this byte array.
ByteVector	**putInt**(int i) Puts an int into this byte array.
ByteVector	**putLong**(long l) Puts a long into this byte array.
ByteVector	**putShort**(int s) Puts a short into this byte array.
ByteVector	**putUTF**(String s) Puts a String in UTF format into this byte vector.

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

data

`public byte[] data`

The content of this array.

---

length

`public int length`

Actual number of bytes in this array.

Constructor Detail

ByteVector

`public ByteVector()`

Constructs a new `ByteVector` with a default initial size.

ByteVector

`public ByteVector(int initialSize)`

Constructs a new `ByteVector` with the given initial size.

Parameters:

`initialSize` - the initial size of the byte array to be constructed.

Method Detail

putByte

`public ByteVector putByte(int b)`

Puts a byte into this byte array. The byte array is automatically enlarged if necessary.

Parameters:

`b` - a byte.

Returns:

this byte array.

---

putChar

`public ByteVector putChar(char c)`

Puts a char into this byte vector. The byte vector is automatically enlarged if necessary.

Parameters:

`c` - char.

Returns:

ByteVector.

---

putShort

`public ByteVector putShort(int s)`

Puts a short into this byte array. The byte array is automatically enlarged if necessary.

Parameters:

`s` - a short.

Returns:

this byte array.

---

putInt

`public ByteVector putInt(int i)`

Puts an int into this byte array. The byte array is automatically enlarged if necessary.

Parameters:

`i` - an int.

Returns:

this byte array.

---

putLong

`public ByteVector putLong(long l)`

Puts a long into this byte array. The byte array is automatically enlarged if necessary.

Parameters:

`l` - a long.

Returns:

this byte array.

---

putFloat

`public ByteVector putFloat(float f)`

Puts a float into this byte array. The byte array is automatically enlarged if necessary.

Parameters:

`f` -

Returns:

ByteVector

---

putDouble

`public ByteVector putDouble(double d)`

Puts a double into this byte array. The byte array is automatically enlarged if necessary.

Parameters:

`d` -

Returns:
ByteVector

---

putUTF

```
public ByteVector putUTF(String s)
```

Puts a String in UTF format into this byte vector. The byte vector is automatically enlarged if necessary.

Parameters:
`s` - a String.

Returns:
this byte vector.

---

putChars

```
public ByteVector putChars(String s)
```

Puts a String as a UTF String into this byte array. The byte array is automatically enlarged if necessary.

Parameters:
`s` - a String

Returns:
ByteVector

---

putByteArray

```
public ByteVector putByteArray(byte[] b,
                               int srcOff,
                               int destOff,
                               int len)
```

Puts an array of bytes into this byte array. The byte array is automatically enlarged if necessary.

Parameters:
`b` - an array of bytes. May be null to put len null bytes into this byte array.
`srcOff` - index of the fist byte of b that must be copied.
`len` - number of bytes of b that must be copied.

Returns:
this byte array.

---

copyByteArray

```
public ByteVector copyByteArray(byte[] b,
                                int destOff,
                                int len)
```

Parameters:
`b` -
`destOff` -
`len` -

Returns:
ByteVector

---

getValidData

`public byte[] getValidData()`

This method gets the effective data from the ByteVector and return it. This will avoid the null characters in the extended part of the byte[] data

Returns:
ByteVector

---

edu.pace.util

Class SOAPUtils

java.lang.Object
└ **edu.pace.util.SOAPUtils**

---

public class **SOAPUtils**
extends Object

Constructor Summary

SOAPUtils()

Method Summary

static javax.xml.soap.SOAPElement	**convertDOMToSOAPElement**(javax.xml.soap.SOAPEnvelo Node DOMNode)
static String	**getAsString**(javax.xml.soap.SOAPEnvelope env)
static javax.xml.soap.SOAPEnvelope	**getSOAPEnvelope**()
static javax.xml.soap.SOAPEnvelope	**getSOAPEnvelope**(Document dom)

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

SOAPUtils

```
public SOAPUtils()
```

Method Detail

getSOAPEnvelope

```
public static javax.xml.soap.SOAPEnvelope getSOAPEnvelope()
```

---

getAsString

```
public static String getAsString(javax.xml.soap.SOAPEnvelope env)
```

---

convertDOMToSOAPElement

```
public static javax.xml.soap.SOAPElement
convertDOMToSOAPElement(javax.xml.soap.SOAPEnvelope env,

Node DOMNode)
                                                        throws
javax.xml.soap.SOAPException
```

Throws:

javax.xml.soap.SOAPException

---

getSOAPEnvelope

```
public static javax.xml.soap.SOAPEnvelope getSOAPEnvelope(Document dom)
```

---

edu.pace.util

Class XMLUtils

java.lang.Object
  └ **edu.pace.util.XMLUtils**

---

public class **XMLUtils**
extends Object

Constructor Summary

XMLUtils()	

Method Summary

static Document	**createEmptyDocument**()
static Document	**getDocumentFromString**(String xml)

static Element	**getNewElement**(String tagName)
static String	**getStringFromDocument**(Node dom)

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

XMLUtils

public **XMLUtils**()

Method Detail

getDocumentFromString

public static Document **getDocumentFromString**(String xml)

---

getStringFromDocument

public static String **getStringFromDocument**(Node dom)

---

createEmptyDocument

public static Document **createEmptyDocument**()

---

getNewElement

public static Element **getNewElement**(String tagName)

Appendix IV– Source Code

Package edu.pace.soap.xop

```
/*
 * File name : XOPSerializer.java
 * Created on : April 23, 2005
 * Author : Biju T Maniampadavathu
 */
package edu.pace.soap.xop;

import java.util.Properties;
import java.util.Iterator;
import java.util.Vector;
import java.util.Enumeration;
import java.io.ByteArrayInputStream;
import javax.mail.Session;
import javax.mail.Multipart;
import javax.mail.MessagingException;
import javax.mail.internet.MimeMessage;
import javax.mail.internet.MimeMultipart;
import javax.mail.internet.MimeBodyPart;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.Name;
import javax.xml.soap.SOAPException;
import javax.xml.soap.SOAPElement;
import javax.activation.MailcapCommandMap;
import javax.activation.CommandMap;
import org.apache.axis.message.PrefixedQName;
import org.apache.axis.encoding.Base64;
import javax.activation.DataHandler;
import java.io.OutputStream;
import java.io.IOException;

import edu.pace.util.SOAPUtils;

/**
 * Creates XOP Package from an Original XML Infoset.
 * <B>Creating XOP Packages</B><br>
 * <ol>
 * <li>Ensure that the original XML Inforset contains no
 * <i>element information item</i> with a [namespace name] of
 * "http://www.w3.org/2004/08/xop/include" and a [local name] of
 * <code>include</code>. As discussed in <b><a
 * href='http://www.w3.org/TR/2004/CR-xop10-20040826'> 2 XOP
 * Infoset Constructs</a> </b>. XML Infosets with such <i> element
 * information items </i> cannot be represented using XOP. </li>
 *
 * <li>Create an empty package.</li>
 *
 * <li>Identify within the original XML Infoset the <i>
 * element information items </i> to be optimized. To be
 * optimized, the characters comprising the [children] of
 * the <i>element information item </i> MUST be in the
 * canonical form of <code>xs:base64Binary</code> (see <b>
 * <a href='http://www.w3.org/TR/2001/REC-xmlschema-0-20010502'>
 * [XML Schema Part 2]3.2.16 base64Binary</a> </b>) and MUST NOT
 * contain any whitespace characters, preceding, inline with
 * or following the non-whitespace content.</li>
 *
```

```
 * <li>Create a XOP Infoset which is a copy of the Original XOP
 * Infoset, but with the [children] of each <i> element information
 * item </i> identified in the previous step replaced by a <code>
 * xop:Include </code> <i>element information item </i> (see <b><a
 * href='http://www.w3.org/TR/2004/CR-xop10-20040826'> 2.1 xop:Include
 * element information item</a></b>) construcuted as follows:
 *<ol>
 * <li>Transform the replaced characters into binary data by processing
 * them as base64-encoded data.  </li>
 * <li>Serialize the binary data into a new part of the package,
 * with appropriate metadata corresponding to the [normalized value]
 * of the <code>href</code> <i> attribute information item </i> of the
 * <code>xop:Include</code> <i> element information item </i> (see <b><a
 * href='http://www.w3.org/TR/2004/CR-xop10-20040826'> 2.2 href
 * attribute information item</a></b>).</li>
 * <li>If the <i>element information item</i> being optimized (i.e. the
 * [parent] of the newly inserted <code>xop:Include</code> element
 * information item) has a <code>xmlmime:contentType</code> <i> attribute
 * information item </i>, its value SHOULD be reflected appropriately in the
 * metadata for the part.</li></ol>
 *</li>
 * <li>Serialize the resulting XOP Infoset into the package using any
 * W3C reccomendation-level version of XML (e.g[XML 1.0], [XML 1.1])
 * and identify it as the root part according to the packaging
 * mechanism's convention, labelling it with the application/xop+xml
 * media type, as described in <b>
 *<a href='http://www.w3.org/TR/2004/CR-xop10-20040826'> 5 Identifying
 * XOP Dcouments</a></b>.
 *</li></ol>
 *<P>
 *<P>
 * This class provides a convinience method called serialize which takes
 * in a SOAPEnvelope and an OutputStream and XOPSerializes the base64
 * encoded data in the SOAPEnvelope and write it to the OutputStream
 * parameter.
 *<p>
 * This class also provides the methods which will help create the XOP
 * Serialized object as individual steps. The methods for the individual
 * creation of the XOP Serialized message should be called in the following
 * order
 * <ol>
 * <li>insertIncludeElement()</li>
 * <li>addBodyPart()</li>
 * <li>finish()</li>
 * <li>writeTo()</li>
 * </ol>
 *@author Biju T Maniampadavathu
 */

public class XOPSerializer{
    private Properties props;
    private Session session;
    private MimeMessage message;
    /**
     *xopInclude contains the QName for include
     *"http://www.w3.org/2004/08/xop/include"
     */
    public static final Name xopInclude = new
PrefixedQName("http://www.w3.org/2004/08/xop/include","Include","xop");
    /**
     *contentType contains the QName for include
     *"http://www.w3.org/2004/06/xmlmime"
     */
```

```
    public static final Name contentType = new
PrefixedQName("http://www.w3.org/2004/06/xmlmime","contentType","xmlmime");
    /**
     *contentType contains the QName for include
     *"href"
     */
    public static final Name href = new PrefixedQName("","href","");
    /**
     * The serializableContentTypes contains the currently identified
     * binary types which xop serializable. As new contentTypes are
     * identified they should be added to this String[].
     * Note: In future modification this array should be replaced by
     * a logic/method which will check the encoded string and
     * identify whether it is base64 encoded or not.
     */
    private static final String[] serializableContentTypes = {
       "image/png",
       "image/jpg",
       "image/jpeg",
       "image/gif",
       "application/pkcs7-signature",
       "application/logdata"
    };
    // to filter the message ID header
    public static String[] filter = new String[] { "Message-ID" };
    // create the container Multipart to which the bodyparts will be added
    private Multipart mp;
    private Vector bodyParts;

    private static int uniqueCID = 10000;

    /**
     * Default Constructor.
     */
    public XOPSerializer(){
       // create some properties and get the default Session
       props = new Properties();
       session = Session.getInstance(props,null);
       message = new MimeMessage(session);
       bodyParts = new Vector();
       bodyParts.add(0,"place holder for serialized envelope");
       mp = new MimeMultipart("Related");
    }
    /**
     * The serialize() method is a convinience method to xop serialize
     * base64 encoded data in a SOAPEnvelope. This method takes in a
     * SOAPEnvelope and OutputStream as parameters, and serialize the
     * base64 encoded data in the SOAPEnvelope to binary stream and
     * write that stream as a MimeBodypart in a MimeMultipart object
     * of a MimeMessage into the provided output stream. The base64
     * element which is serialized is replaced with an include element
     * which contains a ContentID URI which refers to the binary data
     * in the output stream.
     */
    public static MimeMessage serialize(SOAPEnvelope env, OutputStream out)
       throws XOPException{
       setMailCap();
       XOPSerializer serializer = new XOPSerializer();
       return serializer.serializeSOAP(env,out);
    }

    private  MimeMessage serializeSOAP(SOAPEnvelope env,OutputStream out)
       throws XOPException{
```

```
        try{
                SOAPBody body = env.getBody();
                if(isSerializable(body)){
                        getSerializedContent(body);
                        //finish packaging
                        finish(env);
                        //write the message to the output stream
                        writeTo(out);
                        //debug
                        //message.writeTo(System.out,filter);
                    }
                else{
                        throw new XOPException();
                    }
            }
        catch(SOAPException se){
                System.out.println("SOAPException in serialize: " +
se.getMessage());
            }
        catch(MessagingException me){
                System.out.println("MessagingException in
XOPSerializer.isSerializable: " + me.getMessage());
            }
        catch(IOException ioe){
                System.out.println("IOException in XOPSerializer.isSerializable: "
+ ioe.getMessage());
            }
        return message;
    }

    /**
     * This method should be called if the application is handling the
     * logic of creating a XOPSerialized outputstream. This method should
     * be called as the last step of the individual processing of the
     * XOP Serialization. This method will write the MIME message to the
     * provided output stream.
     */

    public void writeTo(OutputStream out)
       throws IOException, MessagingException{
       message.writeTo(out,filter);
    }

    /*
     * The finish method adds the necessary headers for the SOAP Envelope
     * MimeBodyPart and inserts the XOP Serialized SOAPEnvelope as the
     * starting bodyPart of the MimeMessage. Then it attaches the rest
     * of the body parts.
     */
    public void finish(SOAPEnvelope env) throws MessagingException{
       MimeBodyPart xop = new MimeBodyPart();
       xop.setContent(SOAPUtils.getAsString(env),"text/xml");
       xop.addHeader("Content-Type", "application/xop+xml");
       xop.addHeader("Content-Transfer-Encoding", "8bit");
       xop.addHeader("Content-ID",getUniqueCID());
       bodyParts.setElementAt(xop,0);

       for(Enumeration e = bodyParts.elements(); e.hasMoreElements();){
                mp.addBodyPart((MimeBodyPart)e.nextElement());
            }
       // add the Multipart to the message
       message.setContent(mp);
    }
```

```
/**
 * The addMimeBodyPart method creates a MimeBodyPart with the binary data
 * ContentType and Content-ID and adds it to the MimeMessage.
 */
public void addMimeBodyPart(byte[] data, String cType, String cid)
   throws MessagingException{
   MimeBodyPart mbp = new MimeBodyPart();
   mbp.setDataHandler(new DataHandler(new XOPDataSource(data,cType)));
   mbp.addHeader("Content-Transfer-Encoding","binary");
   mbp.addHeader("Content-ID", cid);
   bodyParts.add(mbp);
}

/**
 * The insertIncludeElement method creates a new Include element and
 * adds it to the parameter SOAPElement. It creates a new Content-ID
 * and adds it as a child text element for the Include element and
 * finally returns the created Content-ID.
 */
public String insertIncludeElement(SOAPElement element)
   throws SOAPException
{
   String cType = element.getAttributeValue(contentType);
   String cid = getUniqueCID();
   SOAPElement includeElement = element.addChildElement(xopInclude);
   includeElement.addAttribute(href,cid);
   return cid;
}

private boolean isSerializable(SOAPElement element)
{
   Iterator iter = hasChildren(element);

   if(iter != null)
       {
          while(iter.hasNext()){
              SOAPElement next = (SOAPElement)iter.next();
              if(hasIncludeElement(next))
                 {
                     return false;
                 }
              else
                 {
                     isSerializable(next);
                 }
          }
       }
   return true;
}

private void getSerializedContent(SOAPElement element)
   throws SOAPException, MessagingException
{

   Iterator iter = hasChildren(element);

   if(iter != null)
       {
          while(iter.hasNext()){
              SOAPElement next = (SOAPElement)iter.next();
              if(hasReplaceableElement(next))
                 {
```

```
                        replaceElement(next);
                    }
                else
                    {
                        getSerializedContent(next);
                    }
            }
        }
    }

    private Iterator hasChildren(SOAPElement element)
    {
       Iterator iter = element.getChildElements();
       if(iter.hasNext())
           return iter;
       else
           return null;
    }

    private boolean hasIncludeElement(SOAPElement element)
    {
       Iterator iter = element.getChildElements(xopInclude);
       return(iter.hasNext());
    }

    private boolean hasReplaceableElement(SOAPElement element)
    {
       String cType = element.getAttributeValue(contentType);
       return(isSerializableContent(cType));
    }

    private boolean isSerializableContent(String cType)
    {
       for(int i = 0; i < serializableContentTypes.length; i++)
           {
              if(serializableContentTypes[i].equalsIgnoreCase(cType))
                  return true;
           }
       return false;
    }

    private String getUniqueCID()
    {
       return "cid:http://www.xopserializer.com/cid" + (++uniqueCID);
    }

    private void replaceElement(SOAPElement element) throws MessagingException,
SOAPException
    {
       String encodedString = element.getValue();
       Iterator iter = element.getChildElements();
       //There should be only one child element which is a Text node
       if(iter.hasNext()){
           SOAPElement childTextNode = (SOAPElement)iter.next();
           childTextNode.detachNode();
       }
       String cType = element.getAttributeValue(contentType);
       String cid = insertIncludeElement(element);
       byte[] data = Base64.decode(encodedString);
       addMimeBodyPart(data,cType,cid);
    }

    private static void setMailCap(){
```

```
        // add handlers for main MIME types
        MailcapCommandMap mc =
(MailcapCommandMap)CommandMap.getDefaultCommandMap();
        mc.addMailcap("text/html;; x-java-content-
handler=com.sun.mail.handlers.text_html");
        mc.addMailcap("text/xml;; x-java-content-
handler=com.sun.mail.handlers.text_xml");
        mc.addMailcap("text/plain;; x-java-content-
handler=com.sun.mail.handlers.text_plain");
        mc.addMailcap("multipart/*;; x-java-content-
handler=com.sun.mail.handlers.multipart_mixed");
        mc.addMailcap("message/rfc822;; x-java-content-
handler=com.sun.mail.handlers.message_rfc822");
        mc.addMailcap("image/gif;; x-java-content-
handler=com.sun.mail.handlers.image_gif");
        mc.addMailcap("image/jpeg;; x-java-content-
handler=com.sun.mail.handlers.image_jpeg");
        mc.addMailcap("application/xop+xml;; x-java-content-
handler=com.sun.mail.handlers.text_xml");
        CommandMap.setDefaultCommandMap(mc);
    }
}

/*
 * File name : XOPDeserializer.java
 * Created on : April 24, 2005
 * Author : Biju T Maniampadavathu
 */
package edu.pace.soap.xop;

import javax.mail.internet.MimeMessage;
import javax.mail.internet.MimeMultipart;
import javax.mail.MessagingException;
import org.w3c.dom.Node;
import org.w3c.dom.Document;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.NodeList;
import org.w3c.dom.Element;
import org.w3c.dom.Text;
import org.w3c.dom.DOMException;
import org.apache.axis.encoding.Base64;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import javax.mail.internet.MimeBodyPart;
import javax.activation.DataSource;
import edu.pace.util.XMLUtils;

/**
 * Creates a Reconstituted XML Infoset from a XOP Package.
 *
 * <p>To create a Reconstituted XML Infoset from a XOP Package:
 * <ol>
 * <li>Construct an XML Infoset by parsing the root part of the package as
 * an XML document. The document must be parsed according to the level of
 * the XML Recommendation identified by the XML declaration of that document.
 * If no XML declaration is present, then the document MUST be parsed per
 * <a href='http://www.w3.org/TR/2004/REC-xml-20040204'> [XML 1.0]</a>.</li>
 *
 * <li>Using that XML Infoset, for each <i> element information item </i>, E,
 * which has, as the sole member of its [children] property, a <code>
 * xop:Include</code><i> element information item </i> (as defined in <b> <a
```

```
 * href='http://www.w3.org/TR/2004/CR-xop10-20040826'>2.1 xop:Include element
 * information item</a></b>):<ol>
 *      <li>Locate the part of the package corresponding to the URI in the
<code>
 *      href </code> <i> attribute information item </i> of the <code>
 *      xop:Include</code> <i>element information item</i> (i.e, corresponding
 *      to the URI encoded in the<i> attribute information item's </i>
 *      [normalized value]).</li>
 *      <li>Replace the <code>xop:Include</code> <i>element information
item</i>
 *      that appears in the [children] property of E with <i>character
 *      information items </i> representing the canonical base64 encoding of
 *      the entity body of the identified package part (i.e, effectively
 *      replace the <code>xop:Include</code> <i> elementinformation item </i>
 *      with the data reconstructed from the package part.</li></ol>
 * </ol>
 *
 *@author Biju T Maniampadavathu
 */

public class XOPDeserializer {

    public static Document deserialize(MimeMessage message){
       Document dom = null;
       try
           {
              MimeMultipart mpart = (MimeMultipart)message.getContent();
              String xml = (String)mpart.getBodyPart(0).getContent();
              dom = XMLUtils.getDocumentFromString(xml);
              XOPDeserializer deserializer = new XOPDeserializer();
              deserializer.replaceContent(dom,mpart);
           }
       catch(MessagingException me)
           {
              System.out.println("MessagingException deserializing XOPPacket :"
                            + me.getMessage());
           }
       catch(IOException ioe)
           {
              System.out.println("IOException deserializing XOPPacket :"
                            + ioe.getMessage());
           }
       catch(Exception e){
           e.printStackTrace();
       }
       return dom;
    }

    public void replaceContent(Node dom, MimeMultipart mpart)
       throws MessagingException,IOException
    {
       NodeList nodes = dom.getChildNodes();
       int nodeLength = nodes.getLength();
       if(nodeLength != 0){
           for(int i = 0; i < nodeLength; i++)
              {
                  Node next = nodes.item(i);
                  if(isReplaceableNode(next))
                     {
                         replaceElement(next,mpart);
                     }
                  else
                     {
```

```
                        replaceContent(next,mpart);
                    }
                }
            }
        }

        private boolean isReplaceableNode(Node node)
        {
            if(node.getNodeType() != Node.ELEMENT_NODE)
                {
                    return false;
                }
            else
                {
                    if(!(node.getNodeName().equalsIgnoreCase("xop:include")))
                        {
                            return false;
                        }
                }
            return true;
        }

        private void replaceElement(Node node, MimeMultipart mpart)
            throws MessagingException, IOException
        {
            Node parent = node.getParentNode();
            NamedNodeMap attrbs = node.getAttributes();
            Node href = attrbs.getNamedItem("href");
            String cid = href.getNodeValue();

            DataSource dataSource =
mpart.getBodyPart(cid).getDataHandler().getDataSource();
            InputStream bais = dataSource.getInputStream();
            byte[] buffer = new byte[1024];
            ByteArrayOutputStream os = new ByteArrayOutputStream();
            int count = 0;
            while ((count = bais.read(buffer, 0, buffer.length)) > 0)
                {
                    os.write(buffer, 0, count);
                }
            String encodedString = Base64.encode(os.toByteArray());
            Text newNode = parent.getOwnerDocument().createTextNode(encodedString);
            parent.replaceChild(newNode,node);
        }
}

/*
 * File name : XOPDataSource.java
 * Created on : April 24, 2005
 * Author : Biju T Maniampadavathu
 */
package edu.pace.soap.xop;

import javax.activation.DataSource;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

/**
 * Creates a XOPDataSource.
 * <p>
```

```
 * XOPDataSource is a DataSource that contains body parts of an XOP package.
 * This allows "xop aware" DataContentHandlers to be implemented more
 * efficiently by being aware of such DataSources and using the appropriate
 * methods to access BodyParts.
 *
 *@author Biju T Maniampadavathu
 */

public class XOPDataSource implements DataSource
{
    // content
    private byte[] _bytes;
    // content-type
    private String _type;

    /**
     * Constructor, that constructs a DataSource from a byte[] and contentType
     */
    public XOPDataSource(byte[] bytes,String type)
    {
       super();
       _bytes = bytes;
       _type = type;
    }

    /**
     * Constructor, that constructs a DataSource from a byte[]
     */
    public XOPDataSource(byte[] bytes)
    {
       super();
       _bytes = bytes;
    }

    /**
     * Returns an input stream from this part.
     * <br>
     * The returned input stream is a decoded stream of bytes.
     * @return InputStream
     */
    public InputStream getInputStream() throws IOException
    {
       return new ByteArrayInputStream(_bytes);
    }

    /**
     * The getOutputStream method is not supported for this data source.
     * <br>
     * Any attempt to invoke this method will throw an IOException.
     *
     */
    public OutputStream getOutputStream() throws IOException
    {
       throw new IOException("Not Supported");
    }

    /**
     * Returns the content-type of this DataSource.
     * @return String
     */
    public String getContentType()
    {
       if(_type == null)
```

```
            return "application/octet-stream";
        else
            return _type;
    }
    /**
     * Sets the contentType of the data source.
     * <br>
     * This method provides a convinience mechanism to set the
     * contentType for this data source object.
     */


    public void setType(String type)
    {
        _type = type;
    }

    /**
     * DataSource method to return a name.
     * <br>
     * This data source will return "XOPDataSource".
     * @return String
     */
    public String getName()
    {
        return "XOPDataSource";
    }
}


/*
 * File name : XOPException.java
 * Created on : April 24, 2005
 * Author : Biju T Maniampadavathu
 */

package edu.pace.soap.xop;

/**
 * Creates a XOPException.
 * <p>
 * XOPException is thrown while trying to construct a serialized
 * XOP package from an XML infoset which already contains a
 * <code>xop:Include</code><i> element information item </i>
 * (as defined in <b><a href='http://www.w3.org/TR/2004/CR-xop10-20040826'>
 * 2.1 xop:Include element information item</a></b>
 *
 *@author Biju T Maniampadavathu
 */
public class XOPException extends Exception
{
    public XOPException()
    {
        super("This XML infoset could not be XOP serialized");
    }

    public XOPException(String reason)
    {
        super(reason);
    }
}
```

Package edu.pace.soap.mtom

```
/*
 * File name : MTOMServlet.java
 * Created on : April 25, 2005
 * Author : Biju T Maniampadavathu
 */
package edu.pace.soap.mtom;

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.mail.internet.MimeMessage;
import javax.mail.internet.MimeMultipart;
import javax.mail.internet.MimeBodyPart;
import javax.mail.Session;
import javax.mail.MessagingException;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPException;
import java.util.Properties;
import java.io.IOException;
import javax.activation.DataSource;
import java.io.PrintWriter;
import java.io.InputStream;
import java.io.ByteArrayOutputStream;
import org.w3c.dom.Document;
import javax.activation.MailcapCommandMap;
import javax.activation.CommandMap;
import edu.pace.util.SOAPUtils;
import edu.pace.util.XMLUtils;
import edu.pace.soap.xop.XOPDeserializer;

/**
 * The MTOMServlet class provides a HTTP Transport mechanism for the
 * XOP Serialized packets.
 *@author Biju T Maniampadavathu
 */

public class MTOMServlet extends HttpServlet
{
    // create some properties and get the default Session
    private Properties props;
    private Session session;
    private MimeMessage message;
    private MimeMultipart multipart;
    /**
     * Initialization
     */
    public void init(ServletConfig config) throws ServletException {
        super.init(config);
       // create some properties and get the default Session
       props = new Properties();
       session = Session.getInstance(props,null);
       setMailCap();
    }

    /**
     * Handles a POST request from a client. The request is assumed
     * to contain a SOAP message with the HTTP binding.
     */
```

```
    public void doPost(HttpServletRequest request, HttpServletResponse response)
                            throws ServletException, IOException {
       processMessage(request,response);
    }

    public void doGet(HttpServletRequest request, HttpServletResponse response)
       throws ServletException, IOException {
       processMessage(request,response);
    }

    public byte[] getBodyPartAsByteArray(String cid) throws MessagingException,
IOException
    {
       DataSource dataSource =
multipart.getBodyPart(cid).getDataHandler().getDataSource();
       InputStream bais = dataSource.getInputStream();
       byte[] buffer = new byte[1024];
       ByteArrayOutputStream os = new ByteArrayOutputStream();
       int count = 0;
       while ((count = bais.read(buffer, 0, buffer.length)) > 0)
           {
              os.write(buffer, 0, count);
           }

       return os.toByteArray();
    }

    public byte[] getBodyPartAsByteArray(int index) throws MessagingException,
IOException
    {
       DataSource dataSource =
multipart.getBodyPart(index).getDataHandler().getDataSource();
       InputStream bais = dataSource.getInputStream();
       byte[] buffer = new byte[1024];
       ByteArrayOutputStream os = new ByteArrayOutputStream();
       int count = 0;
       while ((count = bais.read(buffer, 0, buffer.length)) > 0)
           {
              os.write(buffer, 0, count);
           }

       return os.toByteArray();
    }

    public int getBodyPartCount() throws MessagingException{
       return multipart.getCount();
    }

    public void processMessage(HttpServletRequest request, HttpServletResponse
response)
       throws ServletException, IOException{
       try{
              message = new MimeMessage(session, request.getInputStream());
              message.writeTo(System.out);
              MimeMultipart mpart = (MimeMultipart)message.getContent();
              String xml = (String)mpart.getBodyPart(0).getContent();
              //Document dom = XOPDeserializer.deserialize(message);
              System.out.println(xml);
              //System.out.println(XMLUtils.getStringFromDocument(dom));
              Document dom = XMLUtils.getDocumentFromString(xml);
              SOAPEnvelope env = SOAPUtils.getSOAPEnvelope(dom);

              //debug
```

```
            PrintWriter out = response.getWriter();
            out.println();
        }
        catch(MessagingException me)
            {
            System.out.println("MessagingException in processMessage: " +
me.getMessage());
            }
    }

    private static void setMailCap(){
        // add handlers for main MIME types
        MailcapCommandMap mc =
(MailcapCommandMap)CommandMap.getDefaultCommandMap();
        mc.addMailcap("text/html;; x-java-content-
handler=com.sun.mail.handlers.text_html");
        mc.addMailcap("text/xml;; x-java-content-
handler=com.sun.mail.handlers.text_xml");
        mc.addMailcap("text/plain;; x-java-content-
handler=com.sun.mail.handlers.text_plain");
        mc.addMailcap("multipart/*;; x-java-content-
handler=com.sun.mail.handlers.multipart_mixed");
        mc.addMailcap("message/rfc822;; x-java-content-
handler=com.sun.mail.handlers.message_rfc822");
        mc.addMailcap("image/gif;; x-java-content-
handler=com.sun.mail.handlers.image_gif");
        mc.addMailcap("image/jpeg;; x-java-content-
handler=com.sun.mail.handlers.image_jpeg");
        mc.addMailcap("application/xop+xml;; x-java-content-
handler=com.sun.mail.handlers.text_xml");
        CommandMap.setDefaultCommandMap(mc);
    }
}

/*
 * File name : MTOMClient.java
 * Created on : April 24, 2005
 * Author : Biju T Maniampadavathu
 */
package edu.pace.soap.mtom;

import java.net.URL;
import java.net.URLConnection;
import java.net.MalformedURLException;
import java.io.OutputStream;
import java.io.IOException;
import javax.mail.internet.MimeMessage;
import javax.mail.internet.MimeMultipart;
import javax.mail.Session;
import edu.pace.soap.xop.XOPSerializer;
import edu.pace.util.SOAPUtils;
import edu.pace.util.XMLUtils;
import java.io.InputStream;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPBodyElement;
import javax.xml.soap.SOAPElement;
import java.util.Properties;
import org.w3c.dom.Document;

import java.io.BufferedReader;
import java.io.InputStreamReader;
```

```
public class MTOMClient{
  public static void main(String[] args) {
    try {
      URL url = new URL("http://127.0.0.1:8080/mtom/servlet/MTOMServlet");
      URLConnection conn = url.openConnection();
      conn.setDoOutput(true);

      OutputStream out = conn.getOutputStream();
      writeMimeMessage(out);
      out.flush();
      out.close();

      InputStream in = conn.getInputStream();
      //readMimeMessage(in);

    } catch (MalformedURLException e) {
       System.out.println("MalformedURLException in main: "+ e.getMessage());
    } // end of try-catch
    catch (IOException e) {
       System.out.println("IOException in main: "+ e.getMessage());
       e.printStackTrace();
    } // end of try-catch
    catch (Exception e) {
       System.out.println("Exception in main: "+ e.getMessage());
    } // end of try-catch

  } // end of main()


    public static void writeMimeMessage(OutputStream out) throws Exception
    {
       SOAPEnvelope env = SOAPUtils.getSOAPEnvelope();
       SOAPBody body = env.getBody();
       SOAPBodyElement data =
body.addBodyElement(env.createName("data","m","http://www.w3.org/biju"));
       SOAPElement photo = data.addChildElement("photo","m");
       photo.addAttribute(env.createName("contentType","xmlmime","http://www.w3.
org/2004/06/xmlmime"),"image/png");
       photo.addTextNode("/aWKKapGGyQ=");
       SOAPElement sig = data.addChildElement("sig","m");
       sig.addAttribute(env.createName("contentType","xmlmime","http://www.w3.or
g/2004/06/xmlmime"),"application/pkcs7-signature");
       sig.addTextNode("Faa7vROi2VQ=");

       XOPSerializer.serialize(env,out);
    }

    public static void readMimeMessage(InputStream in) throws Exception{
       Properties props = new Properties();
       Session session = Session.getInstance(props,null);
       MimeMessage message = new MimeMessage(session,in);
       message.writeTo(System.out);
       MimeMultipart multipart = (MimeMultipart)message.getContent();
       String xml = (String)multipart.getBodyPart(0).getContent();
       Document dom = XMLUtils.getDocumentFromString(xml);
       SOAPEnvelope env = SOAPUtils.getSOAPEnvelope(dom);
    }

}
```

Package edu.pace.soap.logger

```
/*
 * FileName: BinaryLogHandler.java
 * Created on Apr 30, 2005
 * Author : Biju T Maniampadavathu
 */
package edu.pace.soap.logger;

import java.io.FileOutputStream;
import java.io.IOException;
import java.util.logging.LogManager;
import java.util.logging.Handler;
import java.util.logging.Level;
import edu.pace.util.ByteVector;

/**
 * The <tt>BinaryLogHandler</tt> writes to the specified log file,
 * which can be configured through the properties file, or defaults to
 * the binarylog.log file
 * <p>
 * The <tt>LogRecordBinaryFormatter</tt> class is used for formatting the
 * LogRecords to the binary format before persisting to the file.
 * <p>
 * <b>Configuration:</b>
 * By default each <tt>BinaryLogHandler</tt> is initialized using
 * the following <tt>LogManager</tt> configuration properties.  If
 * properties are not defined (or have invalid values) then the specified
 * default values are used.
 * <ul>
 * <li>   edu.pace.soap.logger.BinaryLogHandler.level
 *       specifies the default level for the <tt>Handler</tt>
 *       (defaults to <tt>java.util.logging.Level.ALL</tt>).
 * <li>   edu.pace.soap.logger.BinaryLogHandler.file
 *       specifies a name for the output file.
 *       (defaults to "binarylog.log").
 * <li>   com.ibm.logging.icl.ODIMessageLogHandler.append
 *       specifies whether the FileHandler should append onto
 *        any existing files (defaults to false).
 * </ul>
 * <p>
 * <p>
 * @author Biju T Maniampadavathu
 */
public class BinaryLogHandler extends Handler{
    private String fileName;
    private FileOutputStream fos;
    private boolean append;
    private static final String DEFAULT_LOG_FILE = "binarylog.log";
    private LogRecordBinaryFormatter formatter = new LogRecordBinaryFormatter();

    /**
     * Create a <tt>BinaryLogHandler</tt>, using only <tt>LogManager</tt>
     * properties (or their defaults).
     *
     * @throws IOException if we are unable to connect to the FileOutputStream.
     */
    public BinaryLogHandler() throws IOException{
       configure();
       if (this.fileName == null || this.fileName.length() == 0) {
           this.fileName = DEFAULT_LOG_FILE;
       }
```

```
        try{
            configureOutputStream(this.fileName,this.append);
        } catch (IOException ioe) {
            System.err.println("BinaryLogHandler: failed to open FileOutputStream
with" + this.fileName + "," + this.append);
            throw ioe;
        }
    }

    /**
     * Construct a <tt>BinaryLogHandler</tt> using a specified
     * fileName and append property.
     *
     * The <tt>BinaryLogHandler</tt> is configured based on
     * <tt>LogManager</tt> properties (or their default values) except that
     * the given log fileName and boolean append arguments are used.
     *
     * @param fileName target log file.
     * @param append append to logfile or not.
     *
     * @throws IllegalArgumentException if fileName is invalid.
     * @throws IOException if we are unable to connect to the FileOutputStream.
     */
    public BinaryLogHandler(String fileName, boolean append) throws
IOException{
        configure();
        this.fileName = fileName;
        this.append = append;
        configureOutputStream(this.fileName, this.append);
    }

    /**
     * Close this output stream.
     *
     */
    public synchronized void close(){
        flush();
        if (fos != null) {
            try {
                fos.close();
            } catch (IOException ix) {
                // drop through.
            }
        }
        fos = null;
    }

    /**
     * Format and publish a <tt>LogRecord</tt>.
     *
     * @param  record  description of the log event
     */
    public synchronized void publish(java.util.logging.LogRecord record) {
        if (!isLoggable(record)) {
            return;
        }
        try{
            //If the ByteVector object received is not null, then write the
            //valid bytes in it to the outputstream
            ByteVector bytes = formatter.getBinaryRecord(record);
            if(bytes != null)
                fos.write(bytes.getValidData());
        }catch(Exception e){
```

```
            //drop through
        }
        flush();
    }

    /**
     * Flush any buffered messages.
     */
    public synchronized void flush() {
        if (fos != null) {
            try {
                fos.flush();
            } catch (IOException ex) {
                //drop through
            }
        }
    }

    private void configure(){
        LogManager manager = LogManager.getLogManager();
        String cname = BinaryLogHandler.class.getName();
        setLevel(configureLevel(manager.getProperty(cname +".level")));
        this.fileName = manager.getProperty(cname +".fileName");
        System.out.println("FileName: " + fileName);
        this.append = configureBoolean(manager.getProperty(cname +".append"));
    }

    private boolean configureBoolean(String boolValue){
        if((boolValue == null) || (boolValue.length() == 0) ||
"false".equalsIgnoreCase(boolValue)){
            return false;
        }
        else{
            return true;
        }
    }

    private void configureOutputStream(String fileName, boolean append) throws
IOException{
        if((fileName == null)|| fileName.length() == 0){
            throw new IllegalArgumentException("Null file name: " + fileName);
        }
        this.fos = new FileOutputStream(fileName,append);
    }

    private Level configureLevel(String level)
    {
        Level defaultLevel = Level.ALL;
        if(level != null)
        {
            try
            {
                defaultLevel = Level.parse(level);
            }
            catch(Exception e)
            {
                //if an exception is thrown fall back to Level.ALL
            }
        }
        return defaultLevel;
    }
}
```

```java
/**
 * FileName: LogRecordBinaryFormatter.java
 * Created on April 27, 2005
 * Author: Biju T Maniampadavathu
 */
package edu.pace.soap.logger;

import java.io.PrintWriter;
import java.io.StringWriter;
import java.util.Locale;
import java.util.MissingResourceException;
import java.util.ResourceBundle;
import edu.pace.util.ByteVector;

/**
 * The LogRecordBinaryFormatter class, converts the LogRecord into a
 * byte array. This byte array is persisted to the log repository. The
 * binary logs are read back and returned as byte array type to the
 * requesting appliction
 *
 *@author Biju T Maniampadavathu
 */

public class LogRecordBinaryFormatter {

    //debugFlag is used for testing purpose now. SHOULD be REMOVED
    //before production release. Along with the debug methods.
    private boolean debugFlag = false;

    /**
     * Format the given LogRecord to the binary XMFFormat
     * @param record LogRecord the log record to be formatted
     * @return String a formatted log record
     */
    public synchronized ByteVector getBinaryRecord(
                              java.util.logging.LogRecord record) {
       ByteVector byteVector = new ByteVector();
       try {
           formatLogRecord(byteVector, record);
       } catch (Exception e) {
           return null;
           //Do Nothing
       }
       debug("Byte Array data length before return", byteVector.data.length);
       debug("Byte Array valid characters length", byteVector.length);
       return byteVector;
    }

    /**
     *
     * @param byteVector
     * @param record
     * @throws
     */
    private void formatLogRecord(
                            ByteVector byteVector,
                            java.util.logging.LogRecord record)
       throws Exception{
       byte[] filler = new byte[4];
       byteVector.putByteArray(filler,0,0,filler.length);
       //jdk log record
       byteVector.data[0] = 0; //to be filled
       byteVector.data[1] = 0; //to be filled
```

```
byteVector.data[2] = 0; //to be filled
byteVector.data[3] = 0; //to be filled
      //--------------IDENTIFICATION SECTION BEGIN -------
//write the level
byteVector.putInt(record.getLevel().intValue());
//write the time
byteVector.putLong(record.getMillis());
//write source class name
byteVector.putChars(record.getSourceClassName());
//write source method name
byteVector.putChars(record.getSourceMethodName());
//write logger name
byteVector.putChars(record.getLoggerName());
//write thread id
byteVector.putInt(record.getThreadID());
//write sequence number
byteVector.putLong(record.getSequenceNumber());
      //--------------IDENTIFICATION SECTION END-------

//--------------MESSAGE SECTION BEGIN-------
//createMessageSection(byteVector, record);
//write the raw message String
debug("Message", record.getMessage());
debug("Message length", record.getMessage().length());
byteVector.putChars(record.getMessage());

//write the resource bundle name
String resourceBundleName = "";
resourceBundleName = record.getResourceBundleName();

debug("ResourceBundle Name", resourceBundleName);
if (resourceBundleName != null) {
    debug("ResourceBundle name length", resourceBundleName.length());
}

byteVector.putChars(resourceBundleName);
//--------------MESSAGE SECTION END--------------

//--------------LOCALIZED MESSAGE SECTION BEGIN-------

//createLocalizedMessageSection(byteVector, record);

//--------------LOCALIZED MESSAGE SECTION END-------

//--------------PARAMETER SECTION BEGIN----------
Object[] params = record.getParameters();
Throwable thrown = record.getThrown();
int paramType = 10; //XMF10PTY = 10
byteVector.putShort(0);
//A JDK LogRecord will not have both Object parameters and
//Throwable type. So it is either one of these
if (params != null) {
    //write Object type, XMF10PTY = 10
    byteVector.putShort(10);
    //convert the Object to a String
    StringBuffer objectString = new StringBuffer();
    arrayToString(record.getParameters(), objectString);
    //write the Object String
    byteVector.putChars(objectString.toString());
}
else if (thrown != null) {
    //write Thrown type, XMF10PTY = 9
    byteVector.putShort(9);
```

```
        //convert the throwable type to a String
        StringWriter stringwriter = new StringWriter();
        PrintWriter printwriter = new PrintWriter(stringwriter);
        Throwable throwable = record.getThrown();
        throwable.printStackTrace(printwriter);
        //write the Throwable String
        byteVector.putChars(stringwriter.toString());
    }
    //------------------PARAMETER SECTION END------------

    //Finally now fill the RDW as an int
    //XMF10LEN = length of byteVector

    int XMF10LEN = byteVector.length;

    byteVector.data[0] = (byte) ((XMF10LEN >>> 24) & 0xFF);
    byteVector.data[1] = (byte) ((XMF10LEN >>> 16) & 0xFF);
    byteVector.data[2] = (byte) ((XMF10LEN >>> 8) & 0xFF);
    byteVector.data[3] = (byte) ((XMF10LEN >>> 0) & 0xFF);
    debug("XMF10LEN", Integer.toBinaryString(XMF10LEN));
    debug("record length ", XMF10LEN);
//return
}


/**
 *
 * @param record
 * @return
 */
private void createLocalizedMessageSection(
                                    ByteVector byteVector,
                                    java.util.logging.LogRecord record) {
    //write the current default locale string
    debug("Localename", Locale.getDefault().toString());
    debug("Localename length", Locale.getDefault().toString().length());
    byteVector.putChars(Locale.getDefault().toString());

    //write the localized and formatted message
    debug("Localized Message", getLocalizedMessage(record));
    debug("Localized Message length", getLocalizedMessage(record).length());

    byteVector.putChars(getLocalizedMessage(record));
    //return
}

/**
 *
 * @param params
 * @param buff
 */
private void arrayToString(Object[] params, StringBuffer buff){
    if(params != null){
        int paramsLen = params.length;
        for(int i = 0; i < paramsLen; i++){
            buff.append("<OBJECT>");
            buff.append(params[i].toString());
            buff.append("</OBJECT>");
        }
    }
}

/**
```

```
     *
     * @param record
     * @return String
     */

    private String getLocalizedMessage(java.util.logging.LogRecord record) {

        String localizedMsg = record.getMessage();
        ResourceBundle rBundle = null;
        /*
            try {
               rBundle =
                   ResourceBundle.getBundle(
                                       iclRecord.getMessageResourceBundleName(),
                                       getLocale(iclRecord.getMessageLocale()));
            } catch (MissingResourceException e) {
               //e.printStackTrace();
            }
        */

        rBundle = record.getResourceBundle();

        if (rBundle != null) {
            try {
               localizedMsg = rBundle.getString(localizedMsg);
            } catch (MissingResourceException mre) {
               //localizedMsg;
            }
        }

        try {
            Object[] params = record.getParameters();
            if ((params == null) || params.length == 0) {
               //No Parameters return the localizedMsg
               return localizedMsg;
            }

            if (localizedMsg.indexOf("{0") >= 0) {
               return java.text.MessageFormat.format(localizedMsg, params);
            }
            return localizedMsg;
        } catch (Exception e) {
            return localizedMsg;
        }
    }

    /**
     * Private method to create Locale object from the Locale String
     * Examples: "en", "en_US", "_US", "en_US_WIN", "de__POSIX", "fr__MAC"
     * @param String
     * @return Locale
     */
    private Locale getLocale(String locString) {
        String language = "";
        String country = "";
        String variant = "";
        int i = 0;
        int beginIndex = 0;
        int length = locString.length();

        while ((i < length) && (isLetter(locString.charAt(i))))
            i++;
        language = locString.substring(0, i);
```

```
    if ((i < length) && locString.charAt(i) == '_')
        i++;

    beginIndex = i;
    while ((i < length) && (isLetter(locString.charAt(i))))
        i++;
    country = locString.substring(beginIndex, i);

    if ((i < length) && locString.charAt(i) == '_')
        i++;

    beginIndex = i;
    while ((i < length) && (isLetter(locString.charAt(i))))
        i++;
    variant = locString.substring(beginIndex, i);

    return new Locale(language, country, variant);
}

/**
 * Private utility method to help the getLocale method.
 * This method returns true if the argument is an alphabet
 *
 * @param char
 * @return boolean
 */
private boolean isLetter(char c) {
    return ('a' <= c && c <= 'z') || ('A' <= c && c <= 'Z');
}

/**
 * Private debug method
 * @param msg
 * @param str
 */
private void debug(String msg, String str) {
    if (debugFlag)
        System.out.println(msg + " = " + str);
}

/**
 * Private debug method
 * @param msg
 * @param i
 */
private void debug(String msg, int i) {
    if (debugFlag)
        System.out.println(msg + " " + i + " = " + Integer.toHexString(i));
}

/**
 * Private debug method
 * @param msg
 * @param l
 */
private void debug(String msg, long l) {
    if (debugFlag)
        System.out.println(msg + " " + l + " = " + Long.toHexString(l));
}

/**
 * Private debug method
```

```
     * @param msg
     * @param s
     */
    private void debug(String msg, short s) {
       if (debugFlag)
           System.out.println(msg + " " + s + " = " + Integer.toHexString(s));
    }

    /**
     * Private debug method
     * @param msg
     * @param b
     */
    private void debug(String msg, byte b) {
       if (debugFlag)
           System.out.println(msg + " " + b + " = " + Integer.toHexString(b));
    }

}

/*
 * FileName: LogRecordReader.java
 * Created on April 28, 2005
 * Author : Biju T Maniampadavathu
 */
package edu.pace.soap.logger;
/**
 * @author Biju T Maniampadavathu
 */

import java.io.ByteArrayInputStream;
import java.io.DataInputStream;
import java.io.EOFException;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Calendar;
import java.util.logging.LogRecord;
import java.util.logging.Level;
import java.util.ResourceBundle;
import java.util.Locale;
import java.util.Vector;
import java.util.Enumeration;
import java.util.MissingResourceException;

/**
 * The LogRecordReader class is responsible for reading the
 * binary LogRecord and reconstituting the LogRecord object.
 *
 * @author Biju T Maniampadavathu
 */
public class LogRecordReader {
    //fis is used to read the binary input stream
    private FileInputStream fis;
    //dataStream is plugged to the fis, to make use to the utility read methods
    private DataInputStream dataStream;

    private String LOG_FILE_NAME = "logRecord.log";
    //debugFlag is used for testing purpose now. SHOULD be REMOVED
    //before production release. Along with the debug methods.
    private boolean debugFlag = false;
```

```
    private Locale localeName = Locale.getDefault();

    /**
     * default constructor
     *
     */
    public LogRecordReader(){
       try{
           File logFile = new File(LOG_FILE_NAME);
           debug("File length = ", logFile.length());
           fis = new FileInputStream(logFile);
           dataStream = new DataInputStream(fis);
       }catch(IOException ioe){
           System.out.println("Error initilaizing the files " +
ioe.getMessage());
           ioe.printStackTrace();
       }
    }

    /**
     * Constructor which takes in the fileName as a String and creates
     * the binary data stream from the file
     */
    public LogRecordReader(String fileName){
       try{
           File logFile = new File(fileName);
           debug("File length = ", logFile.length());
           fis = new FileInputStream(logFile);
           dataStream = new DataInputStream(fis);
       }catch(IOException ioe){
           System.out.println("Error initilaizing the files " +
ioe.getMessage());
           ioe.printStackTrace();
       }
    }

    /**
     * Constructor which takes a byte array parameter and creates
     * the binary data stream from it.
     */
    public LogRecordReader(byte[] data, String locale){
       setLocale(locale);
       ByteArrayInputStream bais = new ByteArrayInputStream(data);
       dataStream = new DataInputStream(bais);
    }

    /**
     * Sets the locale for this reader. The parameter string should be
     * of the format xx_YY, where xx is the two letter code for the
     * language and YY is the two letter code for the Country.
     */
    public void setLocale(String locale){
       localeName = new Locale(locale);
    }

    /**
     * Returns the locale object for this reader.
     */
    public Locale getLocale(){
       return localeName;
    }

    /**
```

```
 * Main method.
 * @param args
 */
public static void main(String [] args){
   LogRecordReader reader = new LogRecordReader();
   if(args.length != 0)
       reader.setLocale(args[0].toUpperCase());

   Vector v = reader.readLogRecord();
   Enumeration enum = v.elements();
   for(;enum.hasMoreElements();)
       reader.printLogRecord("",(LogRecord)enum.nextElement());
}

/**
 * This method reads the InputStream and converts the byte array
 * to a LogRecord. The converted LogRecords are added to a Vector
 * and the vector is returned.
 *
 */
public Vector readLogRecord(){
   int recordLength = 0;
   Vector records = new Vector();

   try{
       while(dataStream.available() != 0 ){
          //read the record length
          recordLength = dataStream.readInt();
          debug("recordLength = ",recordLength);
          //create a buffer to hold the record.
          //the buffer length is (recordLength - 4)
          //where the 4 bytes (bytes to hold an int = 4)
          //subtracted from the
          //recordLength is the value
          //of recordLength which we read above.
          byte[] buffer = new byte[recordLength - 4];
          //copy the binary record content to the buffer
          dataStream.read(buffer);
          //create a new in memory dataStream containing the
          //content of the buffer
          //TODO Can i reuse the dataStream
          debug("byte[] buffer.length " , buffer.length);
          ByteArrayInputStream bais = new ByteArrayInputStream(buffer);
          DataInputStream dataStream = new DataInputStream(bais);

          //recreate the LogRecord
          LogRecord record = readRecord(dataStream);

          records.addElement(record);

       }
   }catch(EOFException eof){
       System.out.println("EOF Exception " + eof.getMessage());
       eof.printStackTrace();
   }catch(IOException ioe){
       System.out.println("IO Exception " + ioe.getMessage());
       ioe.printStackTrace();
   }catch(Exception e){
       System.out.println("Exception " + e.getMessage());
       e.printStackTrace();
   }
   return records;
}
```

```
    /**
     * Private helper method which does the work of reading and converting
     * byte array to a LogRecord.
     * @param dis
     * @param sb
     * @return
     * @throws IOException
     */
    private LogRecord readRecord(DataInputStream dis) throws IOException{
       if(dis == null){
           debug("dis is null", 0);
       }
       debug("inside readRecord ", 1);
       int levelValue = dis.readInt();
       debug("levelValue = ", levelValue);
       Level level = getLevel(levelValue);
       LogRecord logRecord = new LogRecord(level,"");
       debug("inside readRecord ", 1);
       logRecord.setMillis(dis.readLong());
       logRecord.setSourceClassName(readString(dis));
       logRecord.setSourceMethodName(readString(dis));
       logRecord.setLoggerName(readString(dis));
       logRecord.setThreadID(dis.readInt());
       logRecord.setSequenceNumber(dis.readLong());
       logRecord.setMessage(readString(dis));
       String rbName = readString(dis);
       logRecord.setResourceBundleName(rbName);
       debug("ResourceBundle name: ", rbName);
       try{

logRecord.setResourceBundle(ResourceBundle.getBundle(rbName,localeName));
       }catch(MissingResourceException mre){
           System.out.println("Missing Resource Exception");
       }
       //doesn't matter whether parameter or throwable, both are strings
       short paramType = dis.readShort();
       if(paramType == 9 || paramType == 10){
           Object[] params = { readString(dis)};
           logRecord.setParameters(params);
           printLogRecord("The LogRecord read: ",logRecord);
       }
       return logRecord;
    }

    /**
     *
     * @param dis
     * @return
     * @throws IOException
     */
    private String readString(DataInputStream dis) throws IOException{
       int length = dis.readUnsignedShort();
       debug("Strlen", length);
       String readString = dis.readUTF();
       debug("String read", readString);
       return readString;
    }

    private Level getLevel(int value){
       switch(value){
       case Integer.MAX_VALUE:
           return Level.OFF;
```

```
    case 1000:
        return Level.SEVERE;
    case 900:
        return Level.WARNING;
    case 800:
        return Level.INFO;
    case 700:
        return Level.CONFIG;
    case 500:
        return Level.FINE;
    case 400:
        return Level.FINER;
    case 300:
        return Level.FINEST;
    case Integer.MIN_VALUE:
        return Level.ALL;
    }
    return null;
}

/**
 * Print LogRecord method
 * @param logRecord
 * @param msg
 */
public void printLogRecord(String msg,LogRecord logRecord){
    System.out.println("================= LogRecord ================");
    System.out.println(msg);
    System.out.println(logRecord.getMillis());
    System.out.println(logRecord.getLevel().toString());
    System.out.println(logRecord.getSourceClassName());
    System.out.println(logRecord.getSourceMethodName());
    System.out.println(logRecord.getLoggerName());
    System.out.println(logRecord.getThreadID());
    System.out.println(logRecord.getSequenceNumber());
    String msgKey = logRecord.getMessage();
    System.out.println(msgKey);
    String rbName = logRecord.getResourceBundleName();
    System.out.println(rbName);
    ResourceBundle rBundle = logRecord.getResourceBundle();
    System.out.println(rBundle.getString(msgKey));
    Object[] params = logRecord.getParameters();
    if(params != null)
        System.out.println(params[0]);
    System.out.println("================= LogRecord ================");
}

/**
 * Private debug method
 * @param msg
 * @param str
 */
void debug(String msg, String str){
    if(debugFlag)
        System.out.println(msg + " = " + str);
}

/**
 * Private debug method
 * @param msg
 * @param i
 */
private void debug(String msg, int i){
```

```
        if(debugFlag)
            System.out.println(msg + " " + i + " = " + Integer.toHexString(i));
    }

    /**
     * Private debug method
     * @param msg
     * @param l
     */
    private void debug(String msg, long l){
        if(debugFlag)
            System.out.println(msg + " " + l + " = " + Long.toHexString(l));
    }

    /**
     * Private debug method
     * @param msg
     * @param s
     */
    private void debug(String msg, short s){
        if(debugFlag)
            System.out.println(msg + " " + s + " = " + Integer.toHexString(s));
    }

    /**
     * Private debug method
     * @param msg
     * @param b
     */
    private void debug(String msg, byte b){
        if(debugFlag)
            System.out.println(msg + " " + b + " = " + Integer.toHexString(b));
    }
}

/*
 * File name : MTOMLoggerServlet.java
 * Created on : May 1, 2005
 * Author : Biju T Maniampadavathu
 */
package edu.pace.soap.logger;

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.mail.internet.MimeMessage;
import javax.mail.internet.MimeMultipart;
import javax.mail.internet.MimeBodyPart;
import javax.mail.Session;
import javax.mail.MessagingException;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPException;
import java.util.Properties;
import java.io.IOException;
import javax.activation.DataSource;
import java.io.PrintWriter;
import java.io.InputStream;
import java.io.FileInputStream;
import java.io.ByteArrayOutputStream;
import org.w3c.dom.Document;
```

```
import javax.activation.MailcapCommandMap;
import javax.activation.CommandMap;
import edu.pace.util.SOAPUtils;
import edu.pace.util.XMLUtils;
import edu.pace.soap.xop.XOPDeserializer;
import edu.pace.soap.xop.XOPSerializer;
import java.util.Iterator;
import javax.xml.soap.Name;
import javax.xml.soap.SOAPBody;
import org.apache.axis.message.PrefixedQName;
import javax.activation.CommandMap;
import javax.activation.MailcapCommandMap;
import java.io.OutputStream;
import java.io.File;

/**
 * The MTOMServlet class provides a HTTP Transport mechanism for the
 * XOP Serialized packets.
 *@author Biju T Maniampadavathu
 */

public class MTOMLoggerServlet extends HttpServlet
{
    // create some properties and get the default Session
    private Properties props;
    private Session session;
    private MimeMessage message;
    private MimeMultipart multipart;
    /**
     * Initialization
     */
    public void init(ServletConfig config) throws ServletException {
        super.init(config);
       // create some properties and get the default Session
       props = new Properties();
       session = Session.getInstance(props,null);
    }

    /**
     * Handles a POST request from a client. The request is assumed
     * to contain a SOAP message with the HTTP binding.
     */
    public void doPost(HttpServletRequest request, HttpServletResponse response)
                            throws ServletException, IOException {
       processMessage(request,response);
    }

    public void doGet(HttpServletRequest request, HttpServletResponse response)
       throws ServletException, IOException {
       processMessage(request,response);
    }

    public byte[] getBodyPartAsByteArray(String cid) throws MessagingException,
IOException
    {
       DataSource dataSource =
multipart.getBodyPart(cid).getDataHandler().getDataSource();
       InputStream bais = dataSource.getInputStream();
       byte[] buffer = new byte[1024];
       ByteArrayOutputStream os = new ByteArrayOutputStream();
       int count = 0;
       while ((count = bais.read(buffer, 0, buffer.length)) > 0)
           {
```

```
            os.write(buffer, 0, count);
        }

      return os.toByteArray();
    }

    public byte[] getBodyPartAsByteArray(int index) throws MessagingException,
IOException
    {
       DataSource dataSource =
multipart.getBodyPart(index).getDataHandler().getDataSource();
       InputStream bais = dataSource.getInputStream();
       byte[] buffer = new byte[1024];
       ByteArrayOutputStream os = new ByteArrayOutputStream();
       int count = 0;
       while ((count = bais.read(buffer, 0, buffer.length)) > 0)
           {
              os.write(buffer, 0, count);
           }

       return os.toByteArray();
    }

    public int getBodyPartCount() throws MessagingException{
       return multipart.getCount();
    }

    public void processMessage(HttpServletRequest request, HttpServletResponse
response)
       throws ServletException, IOException{
       try{
           setMailCap();
           message = new MimeMessage(session, request.getInputStream());
           //message.writeTo(System.out);
           MimeMultipart mpart = (MimeMultipart)message.getContent();
           String xml = (String)mpart.getBodyPart(0).getContent();
           Document dom = XMLUtils.getDocumentFromString(xml);
           SOAPEnvelope env = SOAPUtils.getSOAPEnvelope(dom);

           OutputStream out = response.getOutputStream();
           if(isReadRecordRequest(env)){
              readRecordResponse(out);
           }
       }
       catch(MessagingException me)
           {
              System.out.println("MessagingException in processMessage: " +
me.getMessage());
           }
       catch(SOAPException se)
           {
              System.out.println("SOAPException in processMessage: " +
se.getMessage());
           }
       catch(Exception e)
           {
              System.out.println("Exception in processMessage: " +
e.getMessage());
           }
    }

    private boolean isReadRecordRequest(SOAPEnvelope env) throws SOAPException
    {
```

```
        SOAPBody body = env.getBody();
        Name readRecord = new
PrefixedQName("http://logger.soap.pace.edu","readRecord","intf");
        Iterator iter = body.getChildElements(readRecord);
        SOAPElement request = null;
        if(iter.hasNext())
            return true;

        return false;
    }

    private void readRecordResponse(OutputStream out)
        throws SOAPException, MessagingException, IOException{
        SOAPEnvelope env = SOAPUtils.getSOAPEnvelope();
        SOAPBody body = env.getBody();
        Name readRecordResponse = new
PrefixedQName("http://logger.soap.pace.edu","readRecordResponse","intf");
        SOAPElement response = body.addBodyElement(readRecordResponse);
        XOPSerializer xopSerializer = new XOPSerializer();
        String cid = xopSerializer.insertIncludeElement(response);
        byte[] data = getLogStream();
        xopSerializer.addMimeBodyPart(data,"application/logData",cid);
        xopSerializer.finish(env);
        xopSerializer.writeTo(out);
        //xopSerializer.writeTo(System.out);
    }

    private byte[] getLogStream(){
        String fileName = "logRecord.log";
        File file = new File("logRecord.log");
        System.out.println(file.getAbsolutePath());
        ByteArrayOutputStream os = new ByteArrayOutputStream();
        try{
            FileInputStream fis = new FileInputStream(fileName);

            int count;
            byte[] buffer = new byte[1024];

            while ((count = fis.read(buffer, 0, buffer.length)) > 0) {
               os.write(buffer, 0, count);
            }
        }catch(Exception e){
            System.out.println("Exception while reading logStream: " +
e.getMessage());
        }
        return os.toByteArray();
    }

    private static void setMailCap(){
        // add handlers for main MIME types
        MailcapCommandMap mc =
(MailcapCommandMap)CommandMap.getDefaultCommandMap();
        mc.addMailcap("text/html;; x-java-content-
handler=com.sun.mail.handlers.text_html");
        mc.addMailcap("text/xml;; x-java-content-
handler=com.sun.mail.handlers.text_xml");
        mc.addMailcap("text/plain;; x-java-content-
handler=com.sun.mail.handlers.text_plain");
        mc.addMailcap("multipart/*;; x-java-content-
handler=com.sun.mail.handlers.multipart_mixed");
        mc.addMailcap("message/rfc822;; x-java-content-
handler=com.sun.mail.handlers.message_rfc822");
```

```
        mc.addMailcap("image/gif;; x-java-content-
handler=com.sun.mail.handlers.image_gif");
        mc.addMailcap("image/jpeg;; x-java-content-
handler=com.sun.mail.handlers.image_jpeg");
        mc.addMailcap("application/xop+xml;; x-java-content-
handler=com.sun.mail.handlers.text_xml");
        CommandMap.setDefaultCommandMap(mc);
    }
}

/*
 * File name : MTOMLoggerClient.java
 * Created on : May 1, 2005
 * Author : Biju T Maniampadavathu
 */
package edu.pace.soap.logger;

import java.net.URL;
import java.net.URLConnection;
import java.net.MalformedURLException;
import java.io.OutputStream;
import java.io.IOException;
import javax.mail.internet.MimeMessage;
import javax.mail.internet.MimeMultipart;
import javax.mail.Session;
import edu.pace.soap.xop.XOPSerializer;
import edu.pace.util.SOAPUtils;
import edu.pace.util.XMLUtils;
import java.io.InputStream;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPBodyElement;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.Name;
import org.apache.axis.message.PrefixedQName;
import java.util.Properties;
import java.util.Iterator;
import javax.activation.DataSource;
import org.w3c.dom.Document;
import java.io.ByteArrayOutputStream;
import java.util.Vector;
import java.util.Enumeration;
import java.util.logging.LogRecord;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import javax.activation.MailcapCommandMap;
import javax.activation.CommandMap;
import java.util.Locale;

public class MTOMLoggerClient{
  public static void main(String[] args) {
    try {
        String locale = Locale.getDefault().toString();
        if(args.length != 0)
            locale = args[0];
        setMailCap();
        URL url = new
URL("http://127.0.0.1:8080/logger/servlet/MTOMLoggerServlet");
        URLConnection conn = url.openConnection();
        conn.setDoOutput(true);

        OutputStream out = conn.getOutputStream();
        writeMimeMessage(out);
```

```
            out.flush();
            out.close();
            InputStream in = conn.getInputStream();
            readMimeMessage(in,locale);
            in.close();

        } catch (MalformedURLException e) {
            System.out.println("MalformedURLException in main: "+ e.getMessage());
        } // end of try-catch
        catch (IOException e) {
            System.out.println("IOException in main: "+ e.getMessage());
            e.printStackTrace();
        } // end of try-catch
        catch (Exception e) {
            System.out.println("Exception in main: "+ e.getMessage());
            e.printStackTrace();
        } // end of try-catch

      } // end of main()


        public static void writeMimeMessage(OutputStream out)
            throws Exception{
            SOAPEnvelope env = SOAPUtils.getSOAPEnvelope();
            SOAPBody body = env.getBody();
            SOAPBodyElement data =
body.addBodyElement(env.createName("readRecord","intf","http://logger.soap.pace
.edu"));
            XOPSerializer.serialize(env,out);
        }

        public static void readMimeMessage(InputStream in, String locale)
            throws Exception{
            Properties props = new Properties();
            Session session = Session.getInstance(props,null);
            MimeMessage message = new MimeMessage(session,in);
            MimeMultipart multipart = (MimeMultipart)message.getContent();
            String xml = (String)multipart.getBodyPart(0).getContent();
            Document dom = XMLUtils.getDocumentFromString(xml);
            SOAPEnvelope env = SOAPUtils.getSOAPEnvelope(dom);
            SOAPBody body = env.getBody();
            Name readRecordResponse = new
PrefixedQName("http://logger.soap.pace.edu","readRecordResponse","intf");
            Iterator iter = body.getChildElements(readRecordResponse);
            SOAPElement response = null;
            if(iter.hasNext()){
                response = (SOAPElement)iter.next();
            }else{
                System.out.println("NO LOGRECORDS AVAILABLE");
            }

            //Iterator iterInclude =
response.getChildElements(XOPSerializer.xopInclude);
            Iterator iterInclude = response.getChildElements();
            SOAPElement include = null;
            if(iterInclude.hasNext()){
                //System.out.println("HAS INCLUDE ELEMENT");
                include = (SOAPElement)iterInclude.next();
            }else{
                System.out.println("NO INCLUDE ELEMENTS");
            }
            String cid = include.getAttributeValue(XOPSerializer.href);
            //System.out.println("CID: " + cid);
```

```
        DataSource dataSource =
multipart.getBodyPart(cid).getDataHandler().getDataSource();
        InputStream bais = dataSource.getInputStream();
        byte[] buffer = new byte[1024];
        ByteArrayOutputStream os = new ByteArrayOutputStream();
        int count = 0;
        while ((count = bais.read(buffer, 0, buffer.length)) > 0)
            {
               os.write(buffer, 0, count);
            }

        //pass the Locale parameter, which is read from this clients
        // command line
        LogRecordReader reader = new LogRecordReader(os.toByteArray(),locale);

        Vector v = reader.readLogRecord();
        Enumeration enum = v.elements();
        for(;enum.hasMoreElements();)
            reader.printLogRecord("",(LogRecord)enum.nextElement());

    }


    private static void setMailCap(){
        // add handlers for main MIME types
        MailcapCommandMap mc =
(MailcapCommandMap)CommandMap.getDefaultCommandMap();
        mc.addMailcap("text/html;; x-java-content-
handler=com.sun.mail.handlers.text_html");
        mc.addMailcap("text/xml;; x-java-content-
handler=com.sun.mail.handlers.text_xml");
        mc.addMailcap("text/plain;; x-java-content-
handler=com.sun.mail.handlers.text_plain");
        mc.addMailcap("multipart/*;; x-java-content-
handler=com.sun.mail.handlers.multipart_mixed");
        mc.addMailcap("message/rfc822;; x-java-content-
handler=com.sun.mail.handlers.message_rfc822");
        mc.addMailcap("image/gif;; x-java-content-
handler=com.sun.mail.handlers.image_gif");
        mc.addMailcap("image/jpeg;; x-java-content-
handler=com.sun.mail.handlers.image_jpeg");
        mc.addMailcap("application/xop+xml;; x-java-content-
handler=com.sun.mail.handlers.text_xml");
        CommandMap.setDefaultCommandMap(mc);
    }
}

/*
 * File name : LoggerApplication.java
 * Created on : May 1, 2005
 * Author : Biju T Maniampadavathu
 */

package edu.pace.soap.logger;

import java.util.logging.Logger;
import java.util.logging.Level;

/**
 * The LoggerApplication class is a sample class which creates an instance
 * of java.util.logging.Logger and logs messages to that logger
 *
 * @author Biju T Maniampadavathu
```

```
 */

public class LoggerApplication{
    private String rbName = "XXX";
    private String loggerName = "MTOM Logger Application";
    private Logger logger;

    public LoggerApplication(){
       this.logger = Logger.getLogger(loggerName,rbName);
    }

    /**
     * Log messages to the logger
     */
    public void logMessages(){
       logger.log(Level.INFO,"msgid-0");
       logger.log(Level.INFO,"msgid-1");
       logger.log(Level.INFO,"msgid-2");
       logger.log(Level.INFO,"msgid-3");
       logger.log(Level.INFO,"msgid-4");
       logger.log(Level.INFO,"msgid-5");
       logger.log(Level.INFO,"msgid-6");
       logger.log(Level.INFO,"msgid-7");
       logger.log(Level.INFO,"msgid-8");
       logger.log(Level.INFO,"msgid-9");
    }

    public static void main(String [] args){
       LoggerApplication app = new LoggerApplication();
       app.logMessages();
    }
}
```

logging.properties

```
############################################################
#      Logging Configuration File
############################################################

############################################################
#      Global properties
############################################################
handlers= edu.pace.soap.logger.BinaryLogHandler

# Default global logging level.
.level= ALL

############################################################
# Handler specific properties.
# Describes specific configuration info for Handlers.
############################################################

# default file output is in user's home directory.
edu.pace.soap.logger.BinaryLogHandler.fileName = logRecord.log
edu.pace.soap.logger.BinaryLogHandler.append = false
```

Package edu.pace.util

```
/*
 * File name : ByteVector.java
 * Created on : May 1, 2005
 * Author : Biju T Maniampadavathu
 */
package edu.pace.util;

/**
 * A dynamically extensible array of bytes. This class is roughly
 * equivalent to a DataOutputStream on top of a ByteArrayOutputStream,
 * but is more efficient.
 */

public final class ByteVector {

  /**
   * The content of this array.
   */
  public byte[] data;

  /**
   * Actual number of bytes in this array.
   */
  public int length;

  /**
   * Constructs a new {@link ByteVector ByteVector} with a default initial size.
   */
  public ByteVector () {
    data = new byte[64];
  }

  /**
   * Constructs a new {@link ByteVector ByteVector} with the given initial size.
   *
   * @param initialSize the initial size of the byte array to be constructed.
   */
  public ByteVector (final int initialSize) {
    data = new byte[initialSize];
  }

  /**
   * Puts a byte into this byte array. The byte array is automatically
   * enlarged if necessary.
   *
   * @param b a byte.
   * @return this byte array.
   */

  public ByteVector putByte (final int b) {
    int length = this.length;
    if (length + 1 > data.length) {
      enlarge(1);
    }
    data[length++] = (byte)b;
    this.length = length;
    return this;
  }

  /**
```

```
 * Puts a char into this byte vector. The byte vector is automatically
 * enlarged if necessary.
 *
 * @param c char.
 * @return ByteVector.
 */

public ByteVector putChar (final char c) {
  int length = this.length;
  if (length + 2 > data.length) {
    enlarge(2);
  }
  byte[] data = this.data;
  data[length++] = (byte)(c >>> 8);
  data[length++] = (byte)(c >>> 0);
  this.length = length;
  return this;
}

/**
 * Puts a short into this byte array. The byte array is automatically
 * enlarged if necessary.
 *
 * @param s a short.
 * @return this byte array.
 */

public ByteVector putShort (final int s) {
  int length = this.length;
  if (length + 2 > data.length) {
    enlarge(2);
  }
  byte[] data = this.data;
  data[length++] = (byte)(s >>> 8);
  data[length++] = (byte)s;
  this.length = length;
  return this;
}

/**
 * Puts an int into this byte array. The byte array is automatically
 * enlarged if necessary.
 *
 * @param i an int.
 * @return this byte array.
 */

public ByteVector putInt (final int i) {
  int length = this.length;
  if (length + 4 > data.length) {
    enlarge(4);
  }
  byte[] data = this.data;
  data[length++] = (byte)(i >>> 24);
  data[length++] = (byte)(i >>> 16);
  data[length++] = (byte)(i >>> 8);
  data[length++] = (byte)i;
  this.length = length;
  return this;
}

/**
 * Puts a long into this byte array. The byte array is automatically
```

```
 * enlarged if necessary.
 *
 * @param l a long.
 * @return this byte array.
 */

public ByteVector putLong (final long l) {
  int length = this.length;
  if (length + 8 > data.length) {
    enlarge(8);
  }
  byte[] data = this.data;
  int i = (int)(l >>> 32);
  data[length++] = (byte)(i >>> 24);
  data[length++] = (byte)(i >>> 16);
  data[length++] = (byte)(i >>> 8);
  data[length++] = (byte)i;
  i = (int)l;
  data[length++] = (byte)(i >>> 24);
  data[length++] = (byte)(i >>> 16);
  data[length++] = (byte)(i >>> 8);
  data[length++] = (byte)i;
  this.length = length;
  return this;
}

/**
 * Puts a float into this byte array. The byte array is automatically
 * enlarged if necessary.
 * @param f
 * @return ByteVector
 */
public ByteVector putFloat(float f){
     return putInt(Float.floatToIntBits(f));
}

/**
 * Puts a double into this byte array. The byte array is automatically
 * enlarged if necessary.
 * @param d
 * @return ByteVector
 */
public ByteVector putDouble(double d){
     return putLong(Double.doubleToLongBits(d));
}

/**
 * Puts a String in UTF format into this byte vector. The byte vector is
 * automatically enlarged if necessary.
 *
 * @param s a String.
 * @return this byte vector.
 */
public ByteVector putUTF (final String s) {
     int charLength = s.length();
     int byteLength = 0;
     for (int i = 0; i < charLength; ++i) {
       char c = s.charAt(i);
       if (c >= '\001' && c <= '\177') {
            byteLength++;
       } else if (c > '\u07FF') {
            byteLength += 3;
       } else {
```

```
            byteLength += 2;
          }
        }
        if (byteLength > 65535) {
          throw new IllegalArgumentException();
        }
        int length = this.length;
        if (length + 2 + byteLength > data.length) {
          enlarge(2 + byteLength);
        }
        byte[] data = this.data;
        data[length++] = (byte)(byteLength >>> 8);
        data[length++] = (byte)(byteLength);
        for (int i = 0; i < charLength; ++i) {
          char c = s.charAt(i);
          if (c >= '\001' && c <= '\177') {
            data[length++] = (byte)c;
          } else if (c > '\u07FF') {
            data[length++] = (byte)(0xE0 | c >> 12 & 0xF);
            data[length++] = (byte)(0x80 | c >> 6 & 0x3F);
            data[length++] = (byte)(0x80 | c & 0x3F);
          } else {
            data[length++] = (byte)(0xC0 | c >> 6 & 0x1F);
            data[length++] = (byte)(0x80 | c & 0x3F);
          }
        }
        this.length = length;
        return this;
    }

    /**
     * Puts a String as a UTF String into this byte array. The byte array
     * is automatically enlarged if necessary.
     *
     * @param s a String
     * @return  ByteVector
     */
        public ByteVector putChars (String s) {
          int length = this.length;

          //the length of a null string written is set to 0
          //i.e null strings are assigned with empty strings

          if(s == null){
            s = "";
          }
          int strLength = s.length();
          if ((length + (2*strLength) + 2) > data.length) {
            enlarge((2*strLength) + 2);
          }
          byte[] data = this.data;
          data[length++] = (byte)(strLength >>> 8);
          data[length++] = (byte)(strLength >>> 0);
          this.length = length;
          putUTF(s);


          return this;
        }

    /**
     * Puts an array of bytes into this byte array. The byte array is
     * automatically enlarged if necessary.
```

```
 *
 * @param b an array of bytes. May be null to put len null
 *      bytes into this byte array.
 * @param srcOff index of the fist byte of b that must be copied.
 * @param len number of bytes of b that must be copied.
 * @return this byte array.
 */
public ByteVector putByteArray (
  final byte[] b,
  final int srcOff,
  final int destOff,
  final int len)
{
  if (length + len > data.length) {
    enlarge(len);
  }
  if (b != null) {
    System.arraycopy(b, srcOff, data, destOff, len);
  }
  length += len;
  return this;
}

/**
 *
 * @param b
 * @param destOff
 * @param len
 * @return ByteVector
 */
public ByteVector copyByteArray (
     final byte[] b,
     final int destOff,
     final int len){
            //ensure the requested limits of the array to be
            //copied falls inside the boundaries of byte[] data
            if(((destOff + len) < length) && (b != null)){
                  System.arraycopy(b,0,data,destOff,len);
            }

            return this;
     }

/**
 * This method gets the effective data from the ByteVector and
 * return it. This will avoid the null characters in the extended
 * part of the byte[] data
 *
 * @return  ByteVector
 */
public byte[] getValidData(){
     byte[] validData = new byte[length];
     System.arraycopy(data,0,validData,0,length);
     return validData;
}

/**
 * Enlarge this byte array so that it can receive n more bytes.
 *
 * @param size number of additional bytes that this byte array should be
 *      able to receive.
 */
private void enlarge (final int size) {
```

```
    byte[] newData = new byte[Math.max(2*data.length, length + size)];
    System.arraycopy(data, 0, newData, 0, length);
    data = newData;
  }
}

/*
 * File name : SOAPUtils.java
 * Created on : May 1, 2005
 * Author : Biju T Maniampadavathu
 */

package edu.pace.util;

import javax.xml.soap.MessageFactory;
import javax.xml.soap.SOAPMessage;
import javax.xml.soap.SOAPPart;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPException;
import javax.xml.soap.SOAPFactory;
import javax.xml.soap.SOAPElement;
import java.io.StringWriter;
import java.io.IOException;
import javax.xml.transform.dom.DOMSource;
import org.w3c.dom.Node;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.NodeList;
import org.w3c.dom.Document;
import javax.activation.DataSource;

public class SOAPUtils
{
    public static javax.xml.soap.SOAPEnvelope getSOAPEnvelope()
    {
       javax.xml.soap.SOAPEnvelope envelope = null;
       try
           {
              MessageFactory msgFactory= MessageFactory.newInstance();
              SOAPMessage message= msgFactory.createMessage();
              javax.xml.soap.SOAPPart soapPart = message.getSOAPPart();
              envelope = soapPart.getEnvelope();

       envelope.addNamespaceDeclaration("xmlmime","http://www.w3.org/2004/06/xml
mime");
           }
       catch(SOAPException se)
           {
              System.out.println("Exception obtaining SOAPEnvelope: "
                              + se.getMessage());
           }
       return envelope;
    }

    public static String getAsString(javax.xml.soap.SOAPEnvelope env){
       StringWriter writer = new StringWriter();
       org.apache.axis.message.SOAPEnvelope
           axisEnv = (org.apache.axis.message.SOAPEnvelope)env;
       try
           {
              MessageFactory msgFactory= MessageFactory.newInstance();
              SOAPMessage message= msgFactory.createMessage();
              org.apache.axis.SOAPPart
                  soapPart = (org.apache.axis.SOAPPart)message.getSOAPPart();
```

```
            soapPart.setSOAPEnvelope(axisEnv);
            soapPart.writeTo(writer);
        }
      catch(SOAPException se)
        {
            System.out.println("SOAPException in  envelope to String: "
                               + se.getMessage());
        }
      catch(IOException ioe)
        {
            System.out.println("IOException in  envelope to String: "
                               + ioe.getMessage());
        }
      catch (Exception e)
        {
            System.out.println("Exception in  envelope to String: "
                               + e.getMessage());
         }
      return writer.toString();
   }

   public static SOAPElement convertDOMToSOAPElement(SOAPEnvelope env,
org.w3c.dom.Node DOMNode) throws SOAPException
   {
      //Test that DOMNode is of type org.w3c.dom.Node.ELEMENT_NODE.
      if((DOMNode.getNodeType()) != org.w3c.dom.Node.ELEMENT_NODE)
          throw new SOAPException("DOMNode must of type ELEMENT_NODE");
      javax.xml.soap.SOAPFactory factory =
javax.xml.soap.SOAPFactory.newInstance();
       SOAPElement se = factory.createElement(DOMNode.getLocalName(),
                              DOMNode.getPrefix(),
                              DOMNode.getNamespaceURI());
      if (DOMNode.hasAttributes())
        {
            NamedNodeMap DOMAttributes = DOMNode.getAttributes();
            int noOfAttributes = DOMAttributes.getLength();
            for(int i = 0; i < noOfAttributes; i++)
                {
                   org.w3c.dom.Node attr = DOMAttributes.item(i);
                   se.addAttribute(env.createName(attr.getLocalName(),
                                                  attr.getPrefix(),
                                                  attr.getNamespaceURI()),
                                   attr.getNodeValue());
                }
        }
      if(DOMNode.hasChildNodes())
        {
            NodeList children = DOMNode.getChildNodes();
             for(int i = 0; i< children.getLength(); i++)
                {
                   org.w3c.dom.Node child = children.item(i);
                   switch(child.getNodeType())
                       {
                       case org.w3c.dom.Node.PROCESSING_INSTRUCTION_NODE:
                          break;
                       case org.w3c.dom.Node.DOCUMENT_TYPE_NODE:
                          break;
                       case org.w3c.dom.Node.CDATA_SECTION_NODE:
                       case org.w3c.dom.Node.COMMENT_NODE:
                       case org.w3c.dom.Node.TEXT_NODE:
                          {
                              se.addTextNode(child.getNodeValue());
                              break;
```

```
                        }
                     default:
                        se.addChildElement(convertDOMToSOAPElement(env,
child));
                     }
                  }//end of for
            }//end of if
         return se;
      }

      public static javax.xml.soap.SOAPEnvelope getSOAPEnvelope(Document dom)
      {
         SOAPEnvelope envelope = null;
         try
             {
                DOMSource source = new DOMSource(dom);
                MessageFactory msgFactory= MessageFactory.newInstance();
                SOAPMessage message= msgFactory.createMessage();
                javax.xml.soap.SOAPPart
                    soapPart = message.getSOAPPart();
                soapPart.setContent(source);
                envelope = soapPart.getEnvelope();
             }
         catch(SOAPException se)
             {
                System.out.println("SOAPException in  envelope to String: "
                                + se.getMessage());
             }
         return envelope;
      }
}

/*
 * File name : XMLUtils.java
 * Created on : May 1, 2005
 * Author : Biju T Maniampadavathu
 */

package edu.pace.util;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import java.io.StringWriter;
import java.io.StringReader;
import java.io.IOException;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.DOMException;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;


public class XMLUtils
{
    private static Document dom = null;
    private static DocumentBuilder builder = null;
```

```
    public static Document getDocumentFromString(String xml)
    {
       Document document = createEmptyDocument();
       try
           {
              document = builder.parse(new InputSource(new StringReader(xml)));
           }
       catch(SAXException se)
           {
              System.out.println("SAXException parsing DOM from String :"
                              + se.getMessage());
           }
       catch(IOException ioe)
           {
              System.out.println("IOException parsing DOM from String :"
                              + ioe.getMessage());
           }
       return document;
    }

    public static String getStringFromDocument(Node dom)
    {
       StringWriter sw=new StringWriter();
       try
           {
              TransformerFactory tFactory = TransformerFactory.newInstance();
              Transformer transformer = tFactory.newTransformer();
              DOMSource source = new DOMSource(dom);
              StreamResult result = new StreamResult(sw);
              transformer.transform(source, result);
           }
       catch(TransformerConfigurationException tce)
           {
              System.out.println("TransformerConfigurationException "
                              + "parsing DOM to String: "
                              + tce.getMessage());
           }
       catch(TransformerException te)
           {
              System.out.println("TransformerException "
                              + "parsing DOM to String: "
                              + te.getMessage());
           }
       return (sw.toString());
    }

    public static Document createEmptyDocument(){
       if(dom != null)
           {
              return dom;
           }
       else
           {
              try{
                  builder =
DocumentBuilderFactory.newInstance().newDocumentBuilder();
                  dom = builder.newDocument();
              }
              catch(Exception e){
                  e.printStackTrace();
              }
              return dom;
```

```
            }
    }

    public static Element getNewElement(String tagName){
       Element element = null;
       try
            {
               element = createEmptyDocument().createElement(tagName);
            }
       catch(DOMException domEx)
            {
                  domEx.printStackTrace();
            }
       return element;
    }
}
```

MTOMLogger Service WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://logger.soap.pace.edu"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://logger.soap.pace.edu"
xmlns:intf="http://logger.soap.pace.edu"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <wsdl:types>
  <schema elementFormDefault="qualified"
targetNamespace="http://logger.soap.pace.edu"
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://logger.soap.pace.edu"
xmlns:intf="http://logger.soap.pace.edu"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
   <element name="readRecord">
    <complexType>
     <sequence/>
    </complexType>
   </element>
   <element name="readRecordResponse">
    <complexType>
     <sequence>
      <element maxOccurs="1" name="readRecordReturn" type="xsd:anyURI"/>
     </sequence>
    </complexType>
   </element>
   <element name="readCursor">
    <complexType>
     <sequence/>
    </complexType>
   </element>
   <element name="readCursorResponse">
    <complexType>
     <sequence>
      <element maxOccurs="1" name="readCursorReturn" type="xsd:anyURI"/>
     </sequence>
    </complexType>
   </element>
   <element name="reset">
    <complexType>
     <sequence/>
    </complexType>
   </element>
   <element name="resetResponse">
    <complexType>
     <sequence/>
    </complexType>
   </element>
  </schema>
 </wsdl:types>

   <wsdl:message name="readRecordRequest">
      <wsdl:part element="intf:readRecord" name="parameters"/>
   </wsdl:message>

   <wsdl:message name="readCursorResponse">
      <wsdl:part element="intf:readCursorResponse" name="parameters"/>
   </wsdl:message>
```

```
   <wsdl:message name="resetResponse">
      <wsdl:part element="intf:resetResponse" name="parameters"/>
   </wsdl:message>

   <wsdl:message name="readRecordResponse">
      <wsdl:part element="intf:readRecordResponse" name="parameters"/>
   </wsdl:message>

   <wsdl:message name="resetRequest">
      <wsdl:part element="intf:reset" name="parameters"/>
   </wsdl:message>

   <wsdl:message name="readCursorRequest">
      <wsdl:part element="intf:readCursor" name="parameters"/>
   </wsdl:message>

   <wsdl:portType name="MTOMLogger">
      <wsdl:operation name="readRecord">
         <wsdl:input message="intf:readRecordRequest"
name="readRecordRequest"/>
         <wsdl:output message="intf:readRecordResponse"
name="readRecordResponse"/>
      </wsdl:operation>

      <wsdl:operation name="readCursor">
         <wsdl:input message="intf:readCursorRequest"
name="readCursorRequest"/>
         <wsdl:output message="intf:readCursorResponse"
name="readCursorResponse"/>
      </wsdl:operation>

      <wsdl:operation name="reset">
         <wsdl:input message="intf:resetRequest" name="resetRequest"/>
         <wsdl:output message="intf:resetResponse" name="resetResponse"/>
      </wsdl:operation>

   </wsdl:portType>
   <wsdl:binding name="MTOMLoggerSoapBinding" type="intf:MTOMLogger">
      <wsdlsoap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>

      <wsdl:operation name="readRecord">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="readRecordRequest">
            <wsdlsoap:body use="literal"/>
         </wsdl:input>
         <wsdl:output name="readRecordResponse">
            <wsdlsoap:body use="literal"/>
         </wsdl:output>
      </wsdl:operation>

      <wsdl:operation name="readCursor">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="readCursorRequest">
            <wsdlsoap:body use="literal"/>
         </wsdl:input>
         <wsdl:output name="readCursorResponse">
            <wsdlsoap:body use="literal"/>
         </wsdl:output>
      </wsdl:operation>

      <wsdl:operation name="reset">
```

```
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="resetRequest">
            <wsdlsoap:body use="literal"/>
         </wsdl:input>
         <wsdl:output name="resetResponse">
            <wsdlsoap:body use="literal"/>
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>

   <wsdl:service name="MTOMLoggerService">
      <wsdl:port binding="intf:MTOMLoggerSoapBinding" name="MTOMLogger">
         <wsdlsoap:address
location="http://localhost:8080/logger/servlet/MTOMLoggerServlet"/>
      </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```