Open GIS Consortium, Inc.

Date: 2003-01-20

Reference number of this OpenGIS® project document: **OGC 03-029**

Version: 0.0.3

Category: OpenGIS® Discussion Paper

Editor: Stephane Fellah (PCI Geomatics)
Steven Keens (PCI Geomatics)

OWS Messaging Framework (OMF)

Document type:	OpenGIS® Discussion Paper
Document stage:	Publicly Available
Document language:	English

Contents

i. Preface

This document defines a messaging framework to conduct communications between the OGC web services. It is independent of any transport protocol and any messaging encoding. By using the framework, the service designer could focus only on the message definitions and messaging flows for every action supported by the service, without worry on the messaging transport and delivery. The framework should considerably simplify the implementations of the OGC web services and should enable service chaining.

ii. Submitting organizations

The following companies submitted this specification to the OGC as a Request for Comment:

```
PCI Geomatics Inc.
490 rue St. Joseph, Suite 400
Hull, Quebec
Canada  J8Y 3Y7
```

iii. Submission contact points

All questions regarding this submission should be directed to the Editor or to the WWW Mapping SIG chair:

Stephane Fellah
PCI Geomatics Inc.
490 rue St. Joseph, Suite 400
Hull, Quebec J8Y 3Y7 CANADA
1-819-770-0022 ext. 223
fellah@pcigeomatics.com

Steven Keens
PCI Geomatics Inc.
490 rue St. Joseph, Suite 400
Hull, Quebec J8Y 3Y7 CANADA
1-819-770-0022
skeens@pcigeomatics.com

iv. Revision history

Date	Release	Author	Paragraph modified	Description
24 Dec 02	0.0.2	Steven Keens		This update contains the new OMF schema and more examples. Also contains examples of OMF bound to HTTP, SOAP, and multipart related.
20 Jan 03	0.0.3	John Davidson		Put into OGC IPR template form.

v. Changes to the OpenGIS® Abstract Specification

None required for IPRs.

Foreword

Attention is drawn to the possibility that some of the elements of this standard may be the subject of patent rights. Open GIS Consortium Inc. shall not be held responsible for identifying any or all such patent rights. However, to date, no such rights have been claimed or identified.

This version of the specification cancels and replaces all previous versions.

Introduction

The current OpenGIS® Web Services such as WMS, WFS and WCS support very simple synchronous data queries based on HTTP transport protocol. These queries use extensively the HTTP GET method with a set of standardized key-value pairs. It allows a user to refer to a map, feature, or coverage by using a simple URL or hyperlink. This is perfectly valid in the case of simple synchronous queries of information from a end-user client such as a browser or a viewer application. The more advanced service WFS-T (WFS with transactional operations) allows the user to insert, update and delete features encoded in GML by using XML messages transported by using HTTP POST method.

The initial OpenGIS web services (WMS/WFS) are quite simple in comparison with the coming services such as the image archive service (an aggregation of multiple services), web coverage server, sensor collection service, metadata services. These new services have to convey much more complex information than GML such as multiple binary data (video, image…), metadata of any formats, notification messages, complex spatio-temporal filter on coverage … A number of limitations have also been identified with the use of HTTP and a centralized network topology for web services. As a standard body, OGC remains agnostic of the network topology and transport protocol to use for its web services infrastructure.

The use cases developed in OWS1.2 have identified a number of requirements for the communication between web services that are going beyond the current existing communication. Among these requirements:

- Support of asynchronous messaging useful for notification or long transaction.
- Support of multiple payloads with different content types (binary or text data)
- Secure and reliable message delivery
- Support of streamlined data such as video or audio
- Transport protocol independence
- Mechanism to favor service chaining
- Network topology independence (centralized, decentralized, hybrid, etc.)
- Extensible and robust framework that could accommodate technology changes.

This document defines a messaging framework for the transport, routing, and packaging of messages, so that clients and OpenGIS services can reliably send and receive their information. In addition, the guidelines for message exchange are designed to be applicable independently of the physical systems, network topologies and messaging standards, used for sending and receiving the data. This approach acts as insurance against obsolescence as newer interchange technologies come along.

The OWS Messaging Framework (OMF) favors a higher decoupling between the service and the client. This allows the service designer to focus on the message definitions and messaging flows for every action supported by the service, without worry on the messaging transport. The framework considerably simplifies the implementations of the OpenGIS services and enables service chaining.

OWS Messaging Framework (OMF)

1 Scope

The scope of this document is to provide a **normative model** for the transport, routing, and packaging of messages, so that the client and services can reliably send and receive their information. OWS Messaging Framework (OMF) is designed to be applicable independently of the physical systems and of any messaging standards. This approach acts as insurance against obsolescence as newer interchange technologies come along.

The following is out of scope in this document:

- Specification of the transport protocol to use with OMF

  The OMF requires that the messages be capable of being carried over any available communications protocol. Therefore, this document does not mandate use of a specific communications protocol. This version of the specification provides bindings to HTTP and SMTP but other protocols can, and reasonably will, be used.

- Specification of the network topology to use with OMF.

- Specification of the binding to specific standard messaging service protocols such as SOAP, ebXML, DIME.

  The OMF is a normative model that specifies the required information to match the requirements described in the introduction. This information is described by UML, XML schema and RDF schema. The XML schema is normative and could be used directly by any XML messaging protocol, but it is not a requirement. It is possible to translate the model to any other interchange technology such as ebXML message, SOAP or DIME by performing a semantic mapping using XSLT or an inference engine. The RDF schema defines the ontology for the messaging framework, i.e. the semantic of the concepts of the model and their relationships. This ontology could be used to perform semantic mapping between different XML syntax automatically. Note that it is also possible to use RDF for messaging exchange that will use the ontology. In this case the messages are exchanged semantically.

2 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

- **Message**
- **Action**
- **Interface**: a named set of operations that characterize the behavior of an entity.
- **Service**: a distinct part of the functionality that is provided by an entity through messages (different from current OGC definition)
- **Client**: a software component that can invoke an action from a server.
- **Request**: an invocation by a client of an operation.
- **Response**: the result of an operation returned from a server to a client.

3 Conventions

3.1 Normative verbs

In the sections labeled as normative, the key words "must", "must not", "required", "shall", "shall not", "should", "should not", "recommended", "may", and "optional" in this document are to be interpreted as described in Internet RFC 2119 [1].

3.2 Abbreviated terms

CGI	Common Gateway Interface
DCP	Distributed Computing Platform
DTD	Document Type Definition
EPSG	European Petroleum Survey Group
GIS	Geographic Information System
GML	Geography Markup Language
HTTP	Hypertext Transfer Protocol
IA	Image Archive
IETF	Internet Engineering Task Force
MIME	Multipurpose Internet Mail Extensions
OGC	Open GIS Consortium

OMF	OpenGIS® Messaging Framework
OWS	OpenGIS® Web Service
SOAP	Simple Object Access Protocol
URL	Uniform Resource Locator
WCS	Web Coverage Service
WFS	Web Feature Service
WOS	Web Object Service
XML	Extensible Markup Language

3.3 Use of examples

This specification makes extensive use of XML examples. They are meant to illustrate the various aspects of the OMF discussed in this specification. While every effort has been made to ensure that the examples are well formed and valid in many cases this goal was sacrificed for the sake of clarity. For example, many examples are formatted in a specific way to highlight a particular aspect that would render the example invalid from the perspective of an XML validation tool. Further, most examples reference fictitious servers and data.

Thus, this specification does not assert that any XML encoded example, copied from this document, will necessarily execute correctly or validate using a particular XML validation tool.

4 Messaging-style versus RPC-style communication

There are two styles of communication between processes that are possible: **Remote Procedure Call (RPC) style and message-style.** This section attempts to discuss about the pros and cons of both approaches and explains why the messaging style is more appropriate for the OpenGIS web service architecture.

4.1 Overview of RPC style

A remote procedure call (RPC)-style communication can be implemented using many existing protocols such as RMI, RPC SOAP, CORBA, DCOM, etc. These traditional distributed object paradigms are tightly coupled and require many restrictions for client and servers.

In the RPC case, the service appears as a remote object to the client application. The interaction between a client and an RPC-style Web service is centered on a service-specific interface. When clients invoke the Web service, they send parameter values to the Web service, which executes the required methods, and then sends back the return values. Because of this back and forth conversation between the client and the Web service,

The RPC style introduces tight coupling in a number of ways::

- Both the client and the service must be up and running at the same time in order for the service to receive the synchronous remote procedure call from the application. In a service chain, if one element in a service chain is offline, the application breaks.

The client application must be programmed to interact with the interface of the service. The name of the method and its parameters needs to be known. Consequently if you change the signature of the method, the application breaks. All the clients of the former interface will need to be updated. That is one of the major drawbacks of the RPC approach. The client needs to know where the service is located to be able to communicate with it.

- The client is required to have some knowledge of the capabilities of the service to find out if the service is compatible or not with it (for example supported filter encoding by a WCS, supported SLD elements by WMS or CPS, supported SRS or formats in WCS).

- RPC-style Web services are synchronous, meaning that when a client sends a request, it waits for a response before doing anything else.

The RPC style is easier to implement: it takes less work to develop communication between two systems when it could be semi-automated (binding code could be generated from existing code). However the tight-coupling of the RPC is not adapted in a widely distributed environment such as the web where the control of the services is decentralized. Service chaining would require a very high availability of the services. We all know that is not the case.

RPC Style Support for different protocol bindings

HTTP Get	Limited cases
HTTP Post	No (convention needs to be used)
SOAP (RPC)	Yes
RMI	Yes
ebXML (more generally any EDI protocol)	No
Corba	Yes

4.2 Messaging Style

Message-style communication is loosely coupled and document-driven rather than being associated with a service-specific interface. When a client invokes a web service using a message-style communication, the client typically sends an entire document, such as a feature filter, rather than a discrete set of parameters. The Web service accepts the entire document, processes it, and may or may not return a result message. Because no tightly-coupled request-response between the client and Web service occurs, message-style Web services promote a looser coupling between client and server.

Message-style Web services could support asynchronous communication. A client that invokes the Web service does not wait for a response before it can do something else. The response from the Web service, if any, can appear minutes, hours or days later. A client can either send or receive a document to or from a message-style Web service.

In a message-style approach, it is necessary to define a message format, and then compose the message using the format before sending and extracting that data from the message after receiving. The messaging style enables a loose coupling by not being bound to particular methods and arguments. This is a good approach when two different business entities need to communicate by using some industry-standard message schema. Basically the interface of the message style services consists of two operations: send and receive.

Messaging Style Support for different protocol bindings

CORBA	Possible
ebXML (more generally any EDI protocol)	Yes
HTTP Get	Limited cases
HTTP Post	Yes
RMI	Possible
SOAP (document)	Yes

The loosely-coupling in "time and space" conveyed the messaging style has motivated the decision to choose the messaging style instead of the RPC style communication. The messaging is supporting all the requirements described in the introduction.

5 Architectural considerations

5.1 XML based messaging

TO DO: The OGC framework is based on XML because…

5.2 Network topology independence

The OGC messaging framework is designed to remain independent of any network topology used by the OGC service architecture. Web Services are typically described in two ways: the conceptual view and the manifestation view. From the conceptual viewpoint, Web services are an example of Service Oriented Architecture (SOA). As with most SOAs, there are three main entities: service providers, service consumers and service registrars. The three entities work in concert to provide a loosely coupled computing paradigm. The manifestation of this paradigm is through standards, such as XML, SOAP, WSDL and UDDI. Peer-to-peer or space systems also leverage a SOA. But unlike Web services, the determination of who is a provider, a consumer or a registrar is much looser. Typically a peer/space is all three aforementioned roles, whereas in Web Services a node is typically a producer and a consumer but not a registrar. Most peer/space systems tend to have strengths in self healing, resilience through redundancy and very loose coupling through a highly distributed topology. All peer/spaces systems are communicating by using asynchronous messaging. It is up to the system architect to choose which network topology is required for its needs. The OGC messaging framework remains neutral to the choice of the network topology and could accommodate both approaches.

5.3 Transport Protocol independence

The OGC messaging framework is independent of any transport protocol. The current OGC web services are highly based on HTTP, but it is not a requirement. Other protocol could be used such as RMI, SMTP, BEEP, wireless packet, Bluetooth, SMS,…that their own twists and purposes. It is undeniable that HTTP is used by virtually every web page on the internet. There is nothing wrong with the protocol per se, as its ubiquity and high dependability mean that it is the one of the best way to make a reliable connection over the internet. However HTTP has severe limitations that could make it a barrier for using web services.

Among the problems with HTTP:

- HTTP is a RPC (Remote Protocol Call) protocol (or synchronous): One program, such as browser, uses an RPC protocol to request a service from another program located in another computer in a network, such as the server, without having to understand the network details. This works for small transactions such as asking for web pages, but when web services start running transactions that take some time to complete over the protocol, the model fails (sensor planning service, orthorectification, classification services…). Intermediaries such as routers and cables between clients and servers will not allow single transactions that take this long. As explained above, RPC is often not a good model for web services. Notification (asynchronous) message is very hard to implement in HTTP without some serious hacking.

- HTTP protocol is asymmetric: Only one entity can initiate an exchange over HTTP, the other entity is passive and can only respond. For peer-to-peer applications, this is not really suitable.

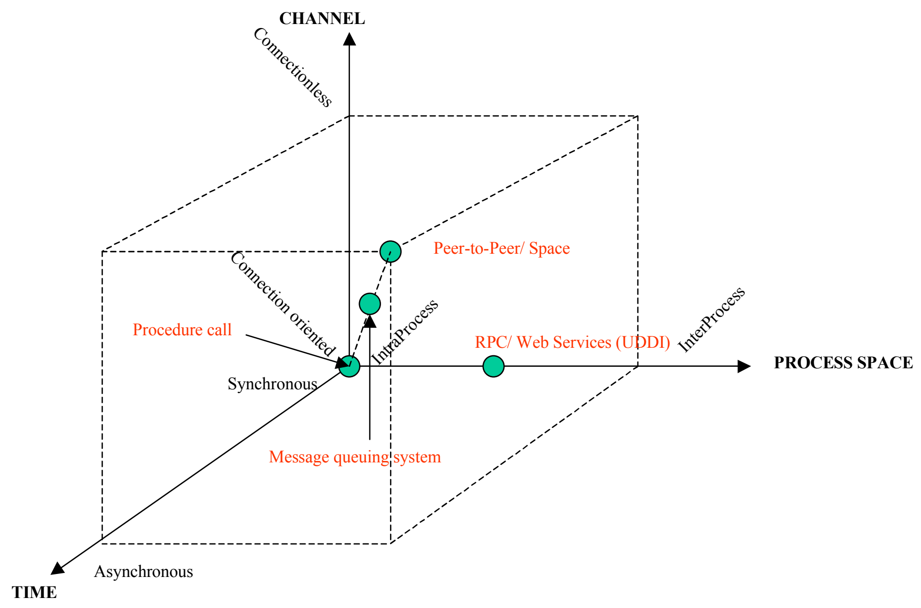
- Inefficient to streamline binary data such as video.

Microsoft, IBM (HTTP-R), W3C and Sun (JXTA) are working on alternative protocols an industry-wide way to do long-running requests that will probably make HTTP less important in the future. The messaging framework is built on top the transport layer and does not have any dependency on any transport protocol.

5.4 Favor loose-coupling in "time and space"

The messaging framework is designed to favor loose-coupling between the client and the services. An application process can broadcast a request message and then terminate. Some time later, a service may come online and download the request message. This is said to be loose coupling in both "time and space" in that the application and service do not need to be connected simultaneously and the application never needs to know the address of the service in order to make that connection. Also, since all communications are message-centric instead of interface or protocol based, applications do not need to be programmed to a particular interface or protocol. As there are no interfaces or protocols to change, the application does not break and the system scales very well.

Using a decentralized broadcast medium, the client does not need to be aware of the service interfaces. Instead, they subscribe to a broadcast medium such as Messaging Oriented Middleware (MOM), space technology, or peer to peer. An application seeking service will publish its message, such as an image processing job, or specific coverage at a specific location, to the medium. When the services see the published message, they, being knowledgeable of their own capabilities, will decide whether to service it or not. For example, a service may service the coverage or performs an image processing task or respond with a bid to perform the job as appropriate. They are many different ways to implement this loose-coupling. Among these technologies are J2EE JMS, ebXML messaging, JXTA, JavaSpace. The figure below summarizes the level of decoupling for each system type.



The focus of this document is the definition of a framework for all the standard messages that will be used across all OGC web services messages for inter-process communication. Numerous messaging framework are available such as ebXML Message (based on SOAP), SOAP, JXTA message, DIME. These technologies are still maturing and subject to change. However, what is common among all these messaging frameworks is the use of XML to encode messages. The intent of this OGC messaging framework is to remain agnostic to the different existing messaging framework. It is up to system architect to use the messaging framework fitting its needs. For example, ebXML messaging is mainly used for B2B where security is critical and collaboration agreement is necessary prior to business transactions. However HTTP POST could be also used for simple access from a Web Browser to some OGC services such as WMS. The message body is the same, but the transport is different. Some implementation guidelines could be provided by OGC for mapping OGC messages to different type messaging frameworks that will emerge in the future. Another crucial aspect is the support of multipart messages and support of binary data attachments (ref. SOAP with attachments). Some protocols are more adequate for transferring binary data than others. Once again, it is up to the system designer to make the choice of the protocol.

5.5 Action-oriented messaging

The design of the framework intends to separate the data from the action to perform on the data. Multiple actions could also be performed on the same data. The framework is said action-oriented.

The advantages of separating the action from the data are the following ones:

- Clean separation of the data from the action on the data. The data-oriented approach is not coupled to any service, so it could be used by a variety of services.
- The same data can be acted upon in many different ways, simply by sending the data to a different service and specifying a different action (assuming that the service supports that action)
- We can capitalize on existing industry DTDs and XML Schemas. For example, the Geographic Markup Language (GML) defines how to structure geographic features. There are many existing instance documents conforming to GML. Web services can be created to provide services for those instance documents. The data format has already been defined. All the service needs to do is to define actions on that data format.
- New cottage industries could be developed to define actions (and their semantics) for vertical industries, e.g., the forestry industry will define a forestry namespace. Within that namespace they will define all the action types on travel-formatted data, along with the semantics of each action type.

This approach of separating the data from the action on the data has implications on how a service must be described. A service description (capability) must specify 2 things:

- The type of data (data model or data format) it can process, e.g., GML document, image type, etc.
- The types of actions it can perform on the data, e.g., get coverage, get feature action, insert, delete, update actions, etc

5.6 Extensibility

OMF is designed to be extensible in order to accommodate different user needs such as security (digital signature, encryption), reliable delivery.

6 Proven models using messaging

6.1 Regular postal system (Snail mail)

Messaging communication describes something analogous to a package sent via the regular postal system (snail mail). The postal system has proven to be a highly effective, reliable, and somewhat private method for sending packages. It has also proven to be extensible … it still works after hundreds (possibly thousands) of years with an explosion of traffic of packages being sent, not to mention the number of senders and recipients.

Conceptually, a package consists of the contents (payload), the wrapping, and the information on the exterior used to show the destination address, source address, payment stamp, certification (security and reliability), and the inventory form. The diagram below shows such a package.

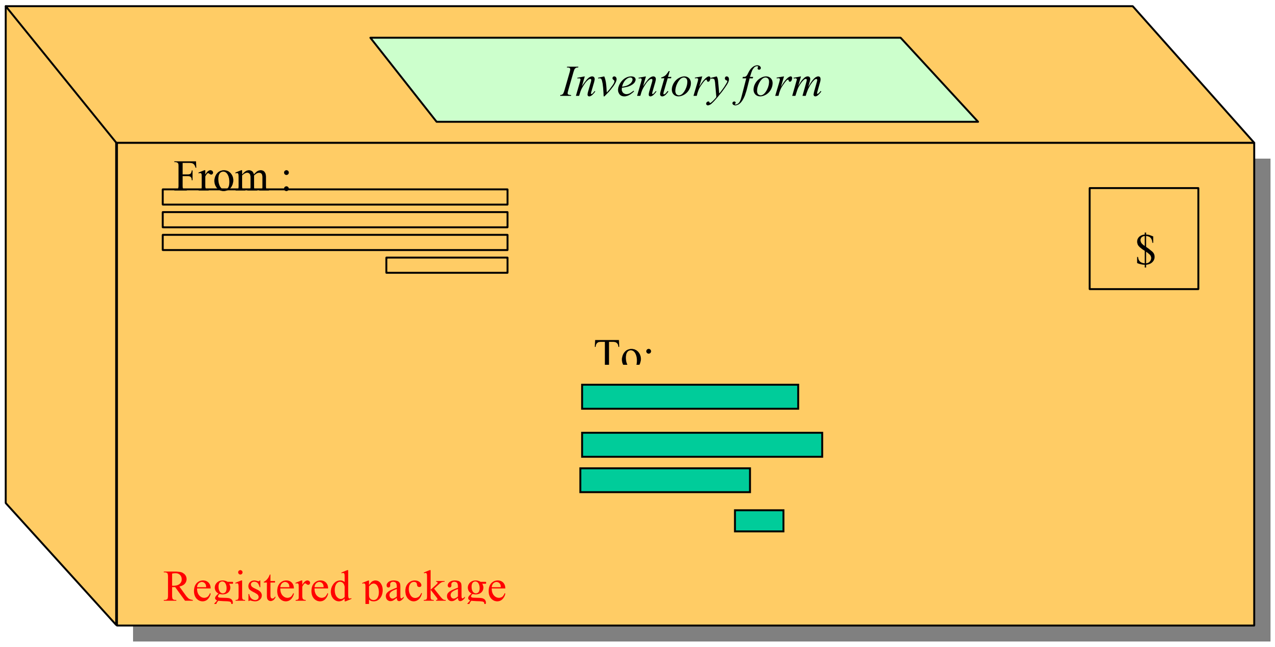


Diagram: Snail Mail Package

Business messages of any kind, whether postal mail, faxes, telegrams and telexes, email or over the web have some characteristics in common. For example, they all have addresses formatted to enable accurate delivery of the message, routing instructions, and dates and times for logging or verification (for example, postmarks on postal mail). These data items, designed to help manage the flow of message traffic, are often grouped together at the top or head of the message (or on an outside envelope), and thus have the names **headers.** This separation of headers from the **body** or **payload** of the message - the business stuff is a common feature of business messages and it is at the core of the

OGC messaging framework specifications. By harnessing the power of such a paradigm and morphing it so that it can be used in the digital universe we will be rewarded with a simple yet extensible framework upon which to build future OGC services.

7 Framework Overview

The proposed framework is based extensively on the analogy with the postal mailing system. It takes in account the architectural considerations and try to make use of the best design patterns applied for XML.

7.1 OGC Message Structure

A postal package consists of the following parts:

- the envelope or wrapping of the packet
- the routing information
- and sometimes an inventory list of the content of the package
- the packing material to segregate the individual items in the contents
- the items in the packing material

Since the messaging framework has been derived from the postal package model, the message package has the same analogous parts:

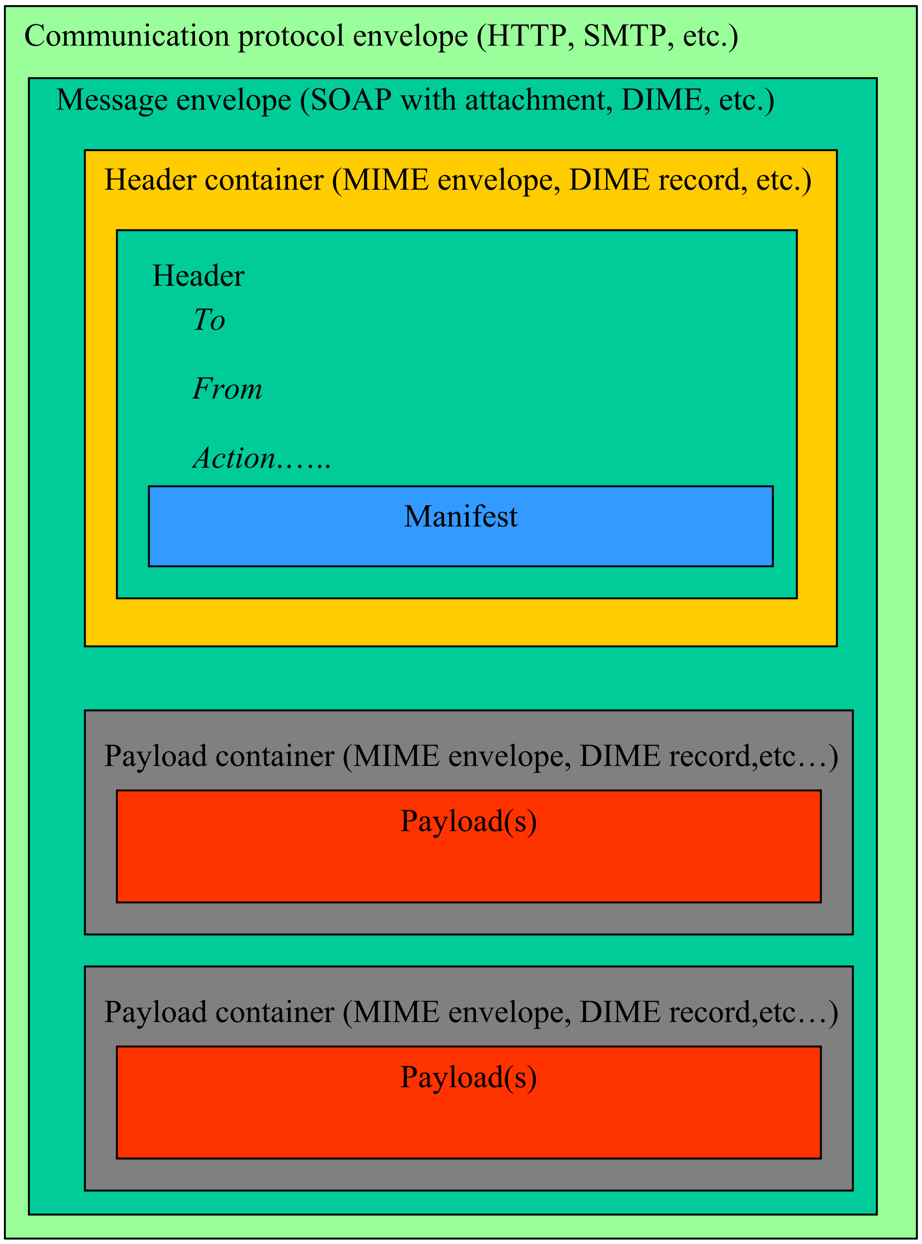
- the message **envelope**
- the **header container** which contains a **header** with addressing information, message identification, expiry information, actions to perform
- sometimes the **manifest** (inventory list) references all items in the payload
- the **payload container** to separate/delineate the individual payload items
- the **payload** which can contain several items (package contents)

To organize the message structure, care has been taken to separate the metadata from the data [**Separate Metadata and Data** design pattern] . This is done by using the **Head-Body** XML design pattern. When creating the structure, the metadata are put first (**Metadata First** XML design pattern). Stream based processors are popular, particularly when message are very large, and may take up large amounts of memory, or when speed of processing is essential. It is often difficult to use stream based processing if the message header is not located at the beginning of the message. The advantage of this approach is that the stream-based processors can easily parse the message without reading the payloads of the message and being able to route the message to the adequate service(s) for processing. It could also send an error to the sender if the message header has problems. This drastically reduces the amount of processing required on the messaging service.

Perhaps needs more elaboration

The following diagram depicts the general abstract structure of the OGC message.

Abstract OGC Message Structure

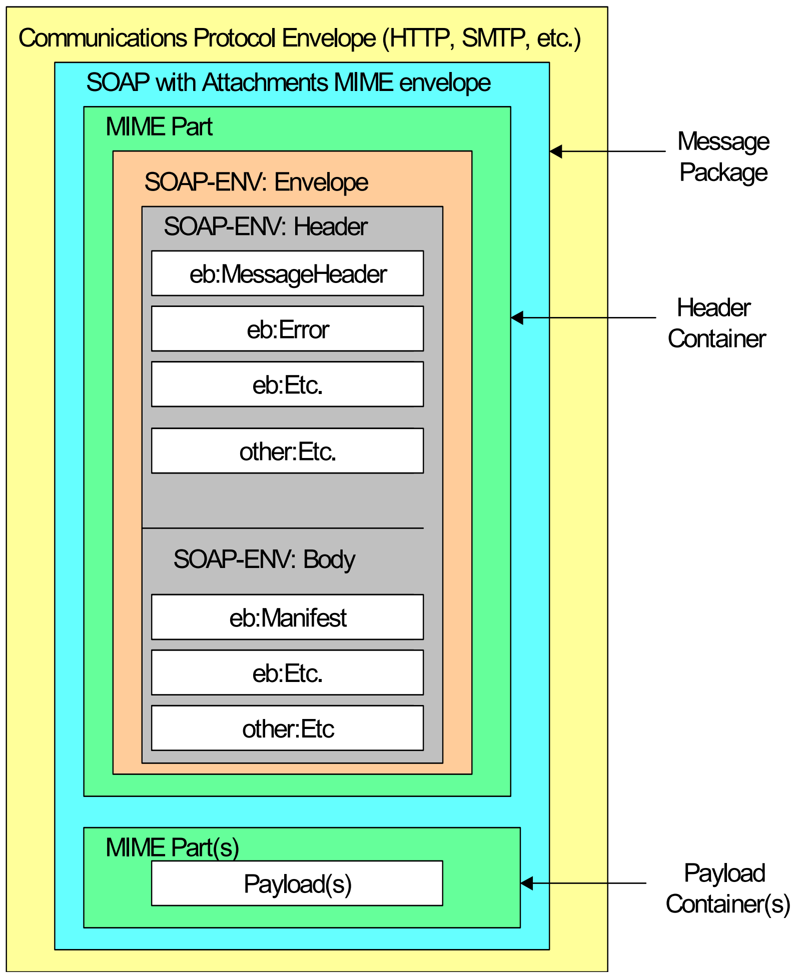

Communication protocol envelope (HTTP, SMTP, etc.)
Message envelope (SOAP with attachment, DIME, etc.)
Header container (MIME envelope, DIME record, etc.)
Header
To
From
Action……
Manifest
Payload container (MIME envelope, DIME record,etc…)
Payload(s)
Payload container (MIME envelope, DIME record,etc…)
Payload(s)

The message can be transported on any communication protocol such as HTTP, SMTP, or TCP. The message is wrapped in an envelope such as SOAP with attachments or a DIME envelope. The message envelope has a header container that could be a MIME part, a DIME record, a JXTA envelope, etc. The header container contains only one header describing the routing and action information and a manifest describing the payloads of the message (if present). The header is extensible to accommodate custom needs such as security information (digital signature, encryption information, etc.). The framework requires the header be encoded in XML. The header container is followed by zero, one or more payload containers which could be MIME part, DIME record for example. Every payload container contains typically one payload items but it is not a requirement. The payload can be any of any type (binary, text, XML, etc.) and size.

The abstract OGC message structure can be easily used by existing messaging protocols such SOAP, ebXML, JXTA or DIME. Because this framework is based on XML, a simple XSLT transformation or a semantic mapping (using an inference engine) makes it easy to convert the OGC message to any of these standards. The mapping mechanism to a specific protocol could be described in the service description (WSDL for example). It is out of scope within this document.

The following diagram is showing the general ebXML message structure. ebXML is using SOAP with attachment for the message envelope and is using MIME part for the header container and payload container. The header is wrapper by using a SOAP envelope.

ebXML Message Structure



The following sections will describe the OMF model more into details. It is important to understand that the model described is normative. The XML schema proposed as is or could be transformed to a specific messaging standard such as ebXML. However the information in the model must be present in the target messaging framework, in order to be usable in the OGC context.

7.2 UML model of OMF

The overall message framework is best depicted by the UML diagram below.

The following sections describe the role of each class.

7.3 Transport classes

7.3.1 Envelope

The **envelope** is an abstraction of a container of message information. It could be manifested for example as a MIME part, a DIME record, a JXTA message, SOAP envelope…. It is not the role of this specification to indicate which mechanism to use. The envelope has two subclasses, the **header container** and the **payload container**. It could be manifested in the same way or differently. The distinction is only logical.

7.3.2 Header container

The **header container** is a "logical" subclass of the envelope. It contains the message **header** information, i.e. the metadata related to the message such as the sender, receiver, and the action to perform on the message, the description of the payloads. The header container format is not specified by the OMF specification like the envelope because it could be manifested as a MIME part or DIME record for example. The header container has only one instance of **header**.

7.3.3 Payload container

A payload container is a "logical" subclass of Envelope. Zero or more **Payload Containers** may be present within a message. If the message contains an application payload, it should be enclosed within a Payload Container. If there is no application payload within the Message Package then a Payload Container must not be present. The contents of each Payload Container must be identified in the Message **Manifest** element within the **header container**. Like the envelope, the payload container is not specified by OMF because it depends on the messaging protocol used. The payload container could be manifested as a MIME part or a DIME record for example.

7.4 Message classes

7.4.1 Header

The **Header** is required for each OGC message and should be placed in **the Header Container.** Only one instance of **Header** is authorized by message. The **Header** contains routing information for the message (To/From, etc.) as well as other context information about the message such as the message identification (**messageID**), reference to the previous message (**refToMessageID**), **time stamp,** expiration information (**expiryTime**), **action** to perform and **manifest** describing the **reference** to the data present either in the **payload container**(s) or elsewhere on the web. Most of this information is optional, except the **Action**.

OMF strongly recommends that the header shall be encoded in XML. A normative XML schema and RDF schema are provided by this specification. Section 8 describes the header element details by using XML schema. RDF schema is provided in annex B.

7.5 Manifest

The **Manifest** describes the contents of the **Resource**, which could be a **payload** of the message. This is done by using the **Reference** object. It makes it possible for recipients to check the content integrity and to determine whether the contents can be processed before their opening. It also helps with payload extraction. OMF requires a **Manifest** if there is any data associated with the message (directly or indirectly). The most obvious data is found in the payload (direct case), but a Manifest is also required if the message references data elsewhere through an Internet address (indirect case). The **Manifest** shall be encoded in XML according to the OMF schema detailed below in section 9.

7.6 Reference

The key component in the **Manifest** is the **Reference**. If the **Manifest** exist, it must have a least one **Reference**. **Reference** may have zero or more schema associated from which the payload is defined. It could also provide a text description of the payload. The **Reference** refers to a URI that could the URI of a payload in the message or a data resource on the web. This class needs to be extensible to accommodate future needs. More details are given in section 9.

7.7 Schema

The **schema** provides the data model of a payload. The model can be described using a DTD, XML schema, RDF schema, DAML, Entity-Relation schema. The schema object gives the internet address where it can be accessed and the version of the schema if needed.

7.8 Error

ErrorList contains a list of **Error** messages indicate that an error occurred in a previous message. The error is a standard message sent by the OGC message service. (Need more elaboration).

7.9 Acknowledgement

Acknowledgement is a standard message sent by an OGC messaging service and refers to a message by using its identifier **MessageID**.

7.10 Payload

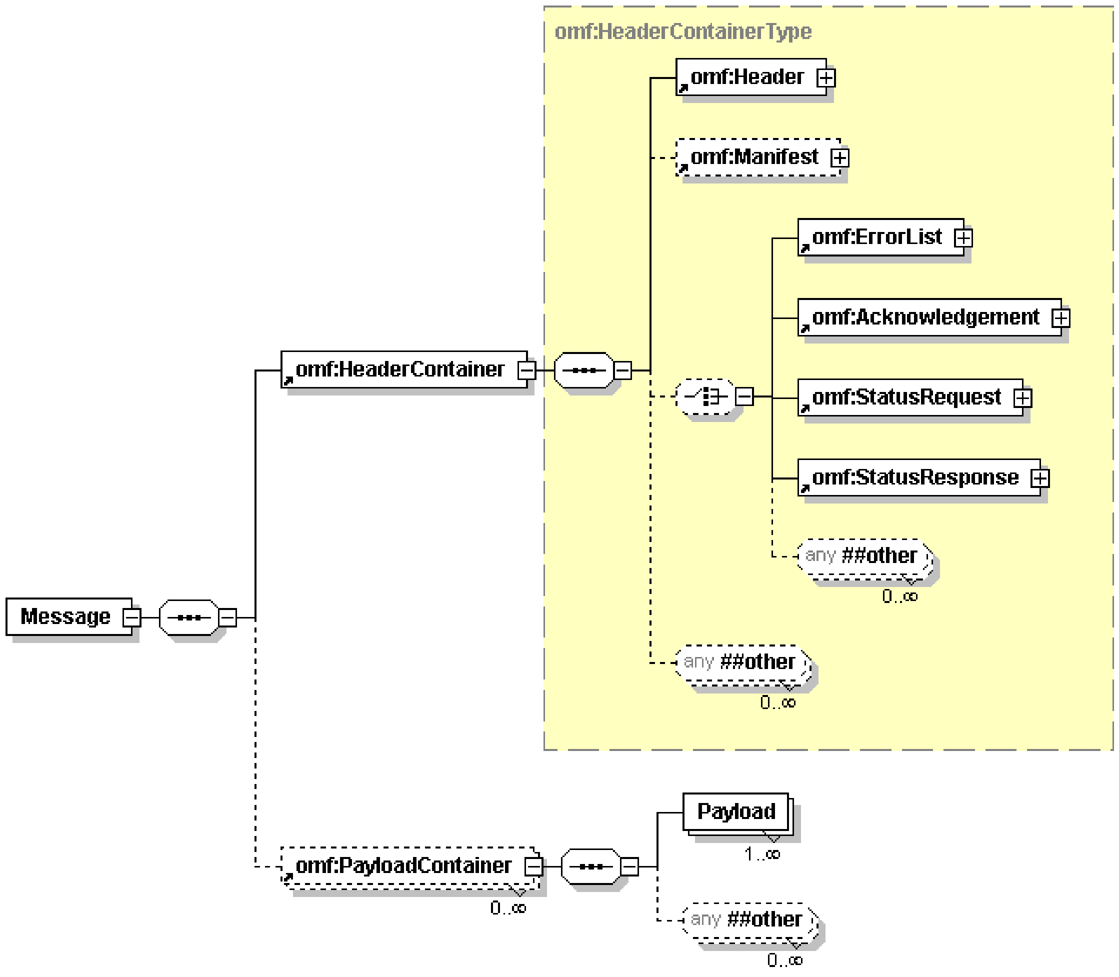
If a message has a **payload container** it must have at least one or more **payload** items. Each **payload** item must be delineated in some way and must be uniquely identified within the message's context.

The OMF Specification makes no provision, nor limits in any way, the structure or content of application payloads. Payloads may be simple-plain-text objects or complex nested multipart objects. The specification of the structure and composition of payload objects is the prerogative of the organization defining the OGC service processes or information exchange using the **OMF.**

TO DO: Need to elaborate on chunked payload

8 Header Element Details

8.1 Message element



8.2 Header element overview

The fragment below defines the normative **Header** XML Schema.

Normative Header XML schema definition:

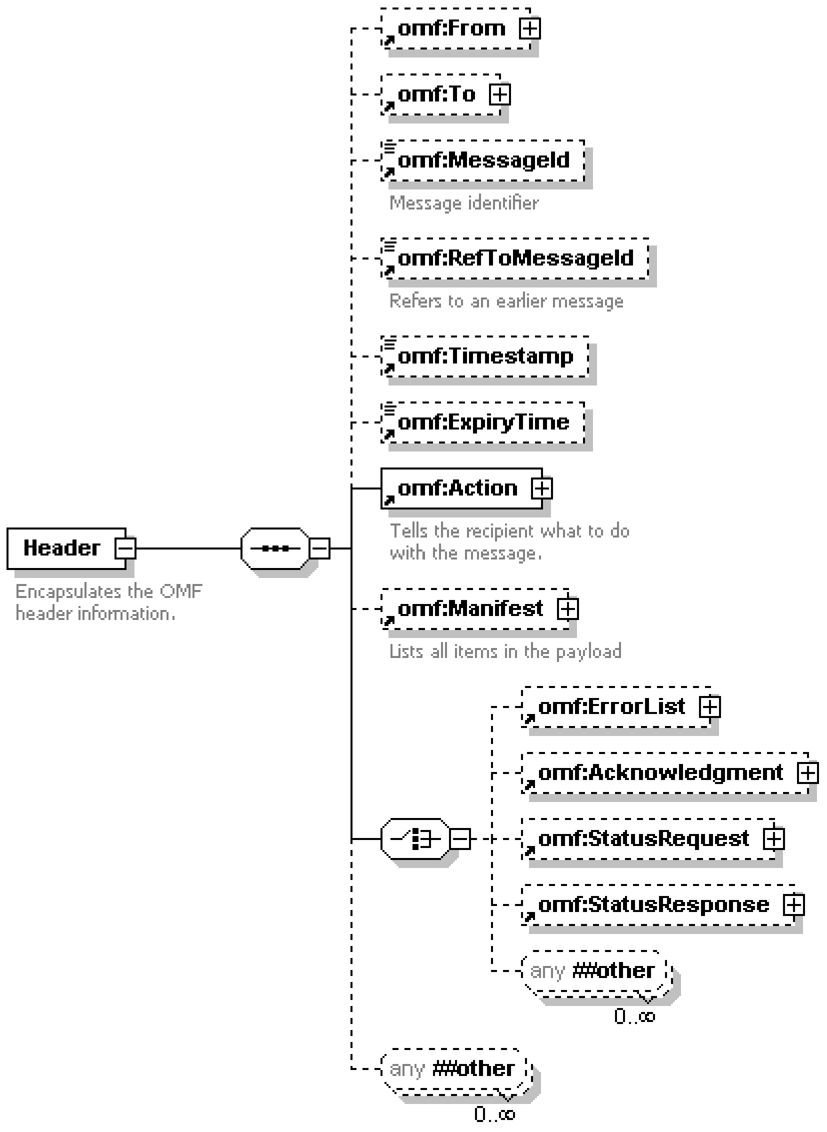
```
  <element name="Header">
        <annotation>
            <documentation>Encapsulates the OMF header
                           information.</documentation>
        </annotation>
        <complexType>
            <sequence>
                <element ref="omf:From" minOccurs="0"/>
                <element ref="omf:To" minOccurs="0"/>
                <element ref="omf:MessageId" minOccurs="0"/>
                <element ref="omf:RefToMessageId" minOccurs="0"/>
                <element ref="omf:Timestamp" minOccurs="0"/>
                <element ref="omf:ExpiryTime" minOccurs="0"/>
                <element ref="omf:Action"/>
                <element ref="omf:Manifest" minOccurs="0"/>
                <choice>
                    <element ref="omf:ErrorList" minOccurs="0"/>
                    <element ref="omf:Acknowledgment" minOccurs="0"/>
                    <element ref="omf:StatusRequest" minOccurs="0"/>
                    <element ref="omf:StatusResponse" minOccurs="0"/>
                    <any namespace="##other" processContents="lax" minOccurs="0"
                                          maxOccurs="unbounded"/>
                </choice>
                <any namespace="##other" processContents="lax" minOccurs="0"
                        maxOccurs="unbounded"/>
            </sequence>
            <attributeGroup ref="omf:headerExtension.grp"/>
        </complexType>
    </element>
```

8.2.1 ##other element

Many **Header** element children have a **##other** element that allows for foreign namespace-qualified element content to be added for extensibility. The extension element must be namespace-qualified in accordance with XMLNS [XMLNS] and **must** belong to a foreign namespace. A foreign namespace is one that is NOT **http://www.opengis.net/schema/omf.xsd.** The wildcard elements are provided wherever extensions might be required for private or future extension of the framework. Implementations of the message handler may ignore the namespace-qualified element and its content.

The **Header** can be used as is, in a message or could be transformed to a specific messaging protocol such as ebXML by using a semantic mapping or a XSLT transformation. The mechanisms to publish the target messaging framework and transform the normative message to this target messaging framework are out of scope in this document.

The following diagram illustrates the **Header** element.



8.2.2 id attribute

Many elements of the header have an **id** attribute which is an XML ID that **may** be added to provide for the ability to uniquely identify the element within the header. This **may** be used when applying a digital signature to the **Message** as individual message extension elements can be targeted for inclusion or exclusion by specifying a URI of "#<idvalue>" in the **Reference** element.

8.2.3 version attribute

The **Header** has a **required version** attribute indicating the version of the OGC Header Specification. Its purpose is to provide future versioning capabilities. For conformance to this specification, all of the version attributes on any extension elements defined in this specification **must** have a value of "1.0". In the future, an OGC message may have a version value higher than "1.0". An implementation conforming to this specification that receives a message with a version higher than "1.0" **may** process the message if it recognizes the version identified and is capable of processing it. It **must** respond with an error (details TBD) if it does not recognize the identified version.

The following sections detail the **Header's** children.

8.3 To and From

To and **From** indicates the sender and receiver of the message. The OGC framework makes these elements **optional** in the case of synchronous calls such as in HTTP Get. In the case of asynchronous communication, the **To** and **From** are **required** in order to send back the response at a later time.

Both **To** and **From** are of type **Actor** and are defined as follows:

The **Actor** is defined by the following XML schema fragment:

Normative PartyId XML schema definition:

```
<element name="Actor" type="omf:ActorType">
<complexType name="ActorType">
       <sequence>
           <element ref="omf:PartyId" maxOccurs="unbounded"/>
           <element name="Role" type="string" minOccurs="0"/>
       </sequence>
</complexType>
```

PartyId is defined as follows:

Normative PartyId XML schema definition:

```
<element name="PartyId" type="omf:IdentifierType"/>
<complexType name="IdentifierType">
       <simpleContent>
           <extension base="omf:non-empty-string">
               <attribute name="type" type="omf:non-empty-string"/>
           </extension>
       </simpleContent>
</complexType>
```

The **actor** type contains multiple **PartyId** elements; all members of the list **must** identify the same **Party**. Unless a single **type** value refers to multiple identification systems, the value of any given **type** attribute **must** be unique within the list of **PartyId** elements contained within either the **From** or **To** element. This mechanism is particularly useful when transport of a message between the parties may involve multiple intermediaries. More generally, the **From** Party should provide identification in all domains it knows in support of intermediaries and destinations that may give preference to particular identification systems.

The **Actor** type contains zero or one **Role** child element that, if present, shall immediately follow the last **PartyId** child element. It is recommended to use URI to define the role. The definition of the role is out of scope in this specification. It could be done for example in a Collaboration Protocol Agreement (CPA) or in the capabilities of the service.

Example: The following fragment demonstrates usage of the **From** and **To** elements.

```
<omf:From>
   <omf:PartyId omf:type="urn:duns">123456789</omf:PartyId>
   <omf:PartyId omf:type="SCAC">RDWY</omf:PartyId>
   <omf:Role>http://rosettanet.org/roles/Buyer</omf:Role>
</omf:From>
<omf:To>
   <omf:PartyId>mailto:joe@example.com</omf:PartyId>
   <omf:Role>http://rosettanet.org/roles/Seller</omf:Role>
</omf:To>
```

8.4 MessageId element

The **MessageId** is optional.

Normative MessageId XML schema definition:

```
<element name="MessageId" type="omf:non-empty-string"/>
```

The **MessageId** element uniquely identifies an individual message. It is useful for persistency and asynchronous calls. Associating the request with the response/acknowledgement is done with the **RefToMessageId** element.

The **MessageId** can be generated by the client or it can be negotiated with a service. The **MessageId** value must be globally unique. It is suggested that a URI be used with the time the message was create appended. An example is provided in section 8.6.

8.5 RefToMessageId element

The **RefToMessageId** element has a cardinality of zero or one. When present, it must contain the **MessageId** value of an earlier message to which this message relates. If there is no earlier related message, the element must not be present. An example is provided in section 8.6.

Normative RefToMessageId XML schema definition:

```
<element name="RefToMessageId" type="omf:non-empty-string"/>
```

For **Error** messages, the **RefToMessageId** element is required and its value must be the **MessageId** value of the message in error.

8.6 Timestamp Element

The OPTIONAL **Timestamp** is a value representing the time that the message header was created conforming to a dateTime [XMLSchema] and MUST be expressed as UTC.

TimeStamp element definition:

```
<element name="TimeStamp" type="dateTime"/>
```

Normative TimeStamp XML schema definition:

```
<element name="TimeStamp" type="dateTime"/>
```

8.7 ExpiryTime element

If the **ExpiryTime** element is present, it must be used to indicate the time, expressed as UTC, by which a message should be delivered to the **To** party. It must conform to an XML Schema **dateTime**.

Normative ExpiryTime XML schema definition:

```
<element name="ExpiryTime" type="dateTime"/>
```

In this context, the **ExpiryTime** has expired if the time of the internal clock, adjusted for UTC, of the receiver is greater than the value of **ExpiryTime** for the message.

If the **To** party's receives a message where **ExpiryTime** has expired, it shall send a message to the **From** party, reporting that the **ExpiryTime** of the message has expired. This message shall be comprised of an ErrorList containing an error with the error **code** attribute set to **ExpiryTimeExpired** and the severity attribute set to **Error**.

The following XML fragment demonstrates the MessageData element's structure:

```
<omf:Header>
       <omf:MessageId>20021215111212@mymessageid.com</omf:MessageId>
    <omf:TimeStamp>2002-12-15T11:00:00</omf:ExpiryTime>
       <omf:ExpiryTime>2002-12-15T11:15:00</omf:ExpiryTime>
       <omf:RefToMessageId>2002-12-15T11:12:15@anothermsgid.com
</omf:RefToMessageId>
</omf:Header>
```

8.8 Action element

The Action specifies the type of activities or processing desired for the message, and must be understood by the service identified in the child Service. The **Action** specifies the process to call in the service. If the values of either the **Service** or **Process** element are unrecognized by the recipient, then it must report an error with an error **code** of **NotRecognized** and a severity of **Error**.

The **Action** header element is a required element and is the most important of the header. There can only be one such element per message.

The action supports a wildcard element to allow future extensibility.

Normative Action XML schema definition:

```
<element name="Action">
       <complexType>
           <complexContent>
               <extension base="omf:ActionType"/>
           </complexContent>
       </complexType>
</element>

<complexType name="ActionType">
       <sequence>
           <element ref="omf:Service"/>
           <element ref="omf:Process"/>
           <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
       </sequence>
       <anyAttribute namespace="##other"/>
</complexType>
```

This element prevents the need for the endpoint to read or process the payload body. The end point becomes a simple router dispatching payloads to the proper services. Each OGC service needs to define the action set it understands.

8.8.1 Service element

The **Service** element is a required element in Action. There can only be one such element per message.

Normative Service XML schema definition:

```
<element name="Service" type="anyURI"/>
```

The **Service** element denotes the OGC service type that processes the message at the destination. Do not confuse the service "type" with specific service end points. By denoting the service type we add another level of decoupling in space. The client does not need to know which service instance or instances handle the message. One or more services handlers could process the message such as in a P2P space based environment. When you have an end point having multiple services (such as image archive service with WFS, WCS, and SCS interfaces) this tag allows routing to the message to the proper service. The router could broadcast the message to all known service handlers.

Note:
URIs in the Service element that start with the namespace **urn:opengis:services:msg-service** are reserved for use by this specification.

The content of the **Service** element MUST be a URI [RFC2396]. If the value of the **Service** element is not unrecognized by the service, then it must report the error with an error **code** of **NotRecognized** and a **severity** of **Error**.

The following example would route the message to a WFS.

```
<omf:Service>urn:ogc:services:wfs</omf:Service>
```

8.8.2 Process element

The **Process** element is a required element in **Action**. Its normative XML schema is defined as follows:

Normative Process XML schema definition:
`<element name="Process" type="anyURI"/>`

The required **Process** element identifies a process within a **Service** that handle the Message. **Process** SHALL be unique within the **Service** in which it is defined. The value of the **Process** element is specified by the designer of the **servic**e or standardized by OGC. If the values of either of the **Process or Service** element are unrecognized by the service, then it must report the error with an error code of NotRecognized and a severity of Error.

An example of the Process (getFeature) supported by WFS can encoded as follows:

```
<omf:Action>
 <omf:Service>urn:opengis:services:wfs</omf:Service>
 <omf:Process>urn:opengis:services:wfs:getFeature</omf:Process>
</omf:Action>
```

Or, according the discussion point above:

```
<omf:Action>
 <omf:Service>urn:opengis:services:wfs</omf:Service>
 <omf:Process>getFeature</omf:Process>
</omf:Action>
```

9 Manifest Details

9.1 Manifest element

The **Manifest** MAY be present in the **Header**. The **Manifest** element is a composite element consisting of one or more **Reference** lelements. Each **Reference** element identifies payload data associated with the message, whether included as part of the message as payload document(s) contained in a **Payload Containe**r, or remote resources accessible via a URL. In the later case, the receiver will have to access the payload in two steps. This could be useful to reduce the size of the message.

The purpose of the **Manifest** is:

- to make it easier to directly extract a particular payload associated with the message
- to allow an application to determine whether it can process the payload without having to parse it.

The **Manifest** element is required when there is a payload. When the payload is empty the manifest must not be part of the message.

The **Manifest** element is comprised of one or more **Reference** elements. Each **Reference** element describes one external resource or one payload item.

Normative Process XML schema definition:

```
<element name="Manifest">
   <complexType>
      <sequence>
         <element ref="omf:Reference" maxOccurs="unbounded"/>
      </sequence>
      <attributeGroup ref="omf:idVersionGroup"/>
   </complexType>
</element>
<element name="Reference">
    <complexType>
      <complexContent>
         <extension base="omf:ReferenceType">
            <anyAttribute namespace="##other"/>
         </extension>
      </complexContent>
   </complexType>
</element>
<element name="Schema">
   <complexType>
      <attribute name="location" type="anyURI" use="required"/>
      <attribute ref="omf:version" use="optional"/>
   </complexType>
</element>
```

9.2 Reference element

The **Reference** element is a child of the manifest. If a manifest exists then there must be at least one **Reference** element. The **Reference** element is a composite element consisting of zero or more **Schema** elements.

A reference must point to a payload or a remote resource bu using a URI. It does so with attributes:

- **id** – an XML ID for the **Reference** element which is optional.This id could be used by digital signature for example.

- **xlink:type** – this attribute defines the element as being an simple link. It has a fixed value of 'simple'.
- **xlink:href** – this required attribute has a value that is the URI of the payload object referenced. It shall conform to the specification criteria for a simple link. It shall conform to the XLINK specification criteria for a simple link. The URI shall use the URI scheme "**cid**" to reference payload items. The URI can use any other URI scheme as long as the URI is resolvable.
- **xlink:role** – this attribute identifies some resource that describes the payload object or its purpose. If present, then it could have a value that is a valid URI or is a simple text string describing the role (URI is recommended).

9.2.1 Reference element validation

If an **xlink:href** attribute contains a URI that is a content id (URI scheme "**cid**") then a payload container must be identified by the content-id (MIME contented or DIME record identifier, etc,…). If it is not, then the error shall be reported to the **From** party with an error **code** of **LinkProblem** and a severity of **Error**.

If an **xlink:href** attribute contains a URI, not a payload container content id (URI scheme "**cid**"), and the URI cannot be resolved, it is an implementation decision whether to report the error. If the error is to be reported, it shall be reported to the **From** party with an error **code** of **LinkProblem** and a severity of **Error**.

Normative Reference XML schema definition:

```
<element name="Reference">
    <complexType>
        <complexContent>
            <extension base="omf:ReferenceType">
                <anyAttribute namespace="##other"/>
            </extension>
        </complexContent>
    </complexType>
</element>
<complexType name="ReferenceType">
    <sequence>
        <element ref="omf:Schema" minOccurs="0" maxOccurs="unbounded"/>
        <element name="description" type="string" minOccurs="0"
                maxOccurs="unbounded"/>
        <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
    <attribute ref="omf:id"/>
    <attribute ref="xlink:href" type="anyURI" use="required"/>
    <attribute ref="xlink:role"/>
    <anyAttribute namespace="##other"/>
</complexType>
```

Note: If a payload item exists, which is not referenced by the **Manifest**, that payload item can be discarded or an error can be reported (the service decides).

9.3 Schema element

If the item being referenced has schema(s) of some kind that describe it (e.g. an XML Schema, DTD and/or a database schema), then the **Schema** element **should** be present as a child of the **Reference** element. It provides a means of identifying the schema and its version defining the payload object identified by the parent **Reference** element. The **Schema** element contains the following attributes:

- **location** – the required URI of the schema
- **version** – a version identifier of the schema

Normative Schema XML schema definition:

```
<element name="Schema">
    <complexType>
        <attribute name="location" type="anyURI" use="required"/>
        <attribute ref="omf:version" use="optional"/>
    </complexType>
</element>
```

9.4 Manifest example

```
<omf:Manifest>
        <omf:Reference omf:id="payref01" xlink:href="cid:payload-item-1"
                       xlink:role="urn:opengis:services:wos:metadata"
                        xlink:type="simple"/>
        <omf:Reference omf:id="payref02"
                       xlink:href="cid:payload-item-2"
                        xlink:role=" urn:opengis:services:wos:data
                        xlink:type="simple">
        <omf:Schema omf:version="1.0"
omf:location="http://www.opengis.net/remotesensing/formats.rdfs#LandsatHDF_EOS_Band1"/>
        </omf:Reference>
</omf:Manifest>
```

10 Error Messages

Error messages indicate that an error occurred in a previous message.

10.1 Definitions:

For clarity, two phrases are defined for use in this section:

- "**message in error**" – A message containing or causing an error or warning of some kind.

- "**message reporting the error**" – A message containing an **ErrorList** element that describes the warning(s) and/or error(s) found in a message in error.

10.2 Error Scope

Errors associated with data communications protocols are detected and reported using the standard mechanisms supported by that data communications protocol and do not use the error reporting mechanism described here.

Errors reported via the OMF error message apply to:

- OMF semantic errors
- OMF validation errors
- Security errors

10.3 Error Elements

10.3.1 ErrorList element

The existence of an **ErrorList** element within a **Header** element indicates the message identified by the **RefToMessageId** element has an error. The **ErrorList** element is only used if reporting an error or warning on a previous message.

This element is optional. If there are no errors to be reported then the **ErrorList** element **must not** be present.

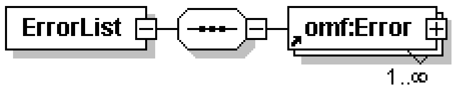
If errors exist is a message then the message reporting the error **must** have the following elements with the specified values:

- The **RefToMessageId must** be present and **must** identify the message in error. It must be have the same value as the message in error's **MessageId**.
- The **Service** element **must** be set to: **urn:opengis:services:msg-service.**
- The **Process** element **must** be set to **MessageError** or **urn:opengis:services:msg-service:message-error.**

Normative Header XML schema definition:

```
<element name="ErrorList">
    <complexType>
        <sequence>
            <element ref="omf:Error" maxOccurs="unbounded"/>
        </sequence>
        <attribute name="highestSeverity" type="omf:severity.type"
                   use="required"/>
    </complexType>
</element>
```

```
<simpleType name="severity.type">
    <restriction base="NMTOKEN">
        <enumeration value="Warning"/>
        <enumeration value="Error"/>
    </restriction>
</simpleType>
```



The **ErrorList** element consists of the following attributes:

- **highestSeverity** - The **highestSeverity** attribute contains the highest severity of any of the **Error** elements. Specifically, if any of the **Error** elements have a severity of **Error**, **highestSeverity** must be set to **Error**; otherwise, **highestSeverity** must be set to **Warning**.

10.3.2 Error element

Normative Header XML schema definition:

```
<element name="Error">
    <complexType>
        <sequence>
            <element ref="omf:Description" minOccurs="0"/>
            <any namespace="##other" processContents="lax" minOccurs="0"
                  maxOccurs="unbounded"/>
        </sequence>
        <attribute ref="omf:id"/>
        <attribute name="context" type="anyURI"
                   default="urn:opengis:services:msg-service"/>
        <attribute name="code" use="required">
            <simpleType>
                <restriction base="omf:non-empty-string">
                    <enumeration value="NotRecognized"/>
                    <enumeration value="NotSupported"/>
                    <enumeration value="ExpiryTimeLapsed"/>
                    <enumeration value="Inconsistent"/>
                    <enumeration value="DeliveryFailure"/>
                    <enumeration value="SecurityFailure"/>
                    <enumeration value="LinkProblem"/>
                    <enumeration value="Unknown"/>
                </restriction>
            </simpleType>
        </attribute>
        <attribute name="severity" type="omf:severity.type" use="required"/>
        <attribute name="location" type="omf:non-empty-string"/>
        <anyAttribute namespace="##other" processContents="lax"/>
    </complexType>
</element>
```

The **ErrorList** element must not exist along with an **Acknowledgement** element.

An **Error** element consists of the following attributes:

- **context** - The **context** attribute identifies the namespace or scheme for the error codes (**code** attribute). It must be a URI. Its default value is **urn:ogc:omf:errors**. If it does not have the default value, then it indicates an implementation of this specification has used its own **code** attribute values. Use of a **context** attribute value other than the default is NOT RECOMMENDED. In addition, an implementation of this specification should not use its own error **code** attribute values if an existing error **code** as defined in this section has the same or very similar meaning.

- **code** - This attribute is required. It indicates the nature of the error in the erroneous message. Valid values for the error **code** and a description of the code's meaning are given in the next section.

- **severity** - The **required** severity attribute indicates the severity of the error. Valid values are:

  - Warning - This indicates other messages in the conversation could be generated in the normal way in spite of this problem.

  - Error - This indicates there is an unrecoverable error in the message and no further message processing should occur. Appropriate failure conditions should be communicated.

- **location** - The location attribute points to the part of the message containing the error. If an error exists in an OMF element and the containing document is "well formed", then the content of the location attribute must be an XPointer or XPath. If the error is associated with a payload, then location contains the payload identifier of the payload in error, using the URI scheme "cid".

Should we use an Xpointer, Xpath, or even, Xlink? I don't know the differences between the three to say. ebXML uses Xpointer.

10.3.3 Description element

The content of the **Description** element provides a narrative description of the error in the language defined by the **xml:lang** attribute. The XML parser or other software validating the message typically generates the message. The content is defined by the vendor/developer of the software that generated the **Error** element.

Normative Header XML schema definition:

```
<element name="Description">
    <complexType>
        <simpleContent>
            <extension base="omf:non-empty-string">
                <attribute ref="xml:lang" use="required"/>
            </extension>
        </simpleContent>
    </complexType>
</element>
```

10.4 Error code descriptions

Error Code	**Long Description**
NotRecognized	Although the document is well formed and valid, the element or attribute contains a value that could not be recognized thus it could not be used by service.
NotSupported	Although the document is well formed and valid, a module present consistent with the rules and constraints contained in this specification, but is not supported by the service processing the message.
ExpiryTimeLapsed	A message has been received that arrived after the time specified in the **ExpiryTime** element.
Inconsistent	Although the document is well formed and valid, according to the rules and constraints contained in this specification content of an element or attribute is inconsistent with content of other elements or their attributes.
DeliveryFailure	A message has been received that either probably or definitely could not be sent to its next destination. Note: if severity is set to **Warning** then there is a small probability that the message was delivered.
SecurityFailure	Validation of signatures or checks on the authenticity or authority of the sender of the message has failed.
LinkProblem	A URI link could not be resolved. If the severity is set to **Warning** then it is left to the application to determine how to handle the condition.
Unknown	Indicates that an error has occurred not covered explicitly by any of the other errors. The content of the **Error** element should be used to indicate the nature of the problem.

10.5 ErrorList Samples

Service error & Process (XML fragment):

```
<omf:ErrorList omf:highestSeverity="error">
       <omf:Error omf:id="error_001" omf:code="NotRecognized"
                  omf:severity="Error"
                  omf:location="/Header/Action/Service">
           <omf:Description xml:lang="en-US">
               Unrecognized service urn:ogc:services:wcs
           <omf:Description>
       </omf:Error>
       <omf:Error omf:id="error_002" omf:code="NotRecognized"
                  omf:severity="Error"
                  omf:location="/Header/Action/Process">
           <omf:Description xml:lang="en-US">
               Unrecognized process GetCaps
           <omf:Description>
       </omf:Error>
</omf:ErrorList>
```

Action error (full XML):

```
<?xml version="1.0" encoding="UTF-8"?>
<omf:Header xmlns:omf="http://www.opengis.net/schema/omf.xsd"
               xmlns:xlink="http://www.w3.org/1999/xlink"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:schemaLocation="http://www.opengis.net/schema/omf.xsd"
               omf:version="1.0">
    <omf:RefToMessageId>20021215111212@messageid.com</omf:RefToMessageId>
   <omf:Action>
        <omf:Service>urn:ogc:omf:service</omf:Service>
    <omf:Process>MessageError</omf:Process>
   </omf:Action>
   <omf:ErrorList highestSeverity="Error">
       <omf:Error code="NotRecognized" severity="Error">
           <omf:Description xml:lang="en-US">
               Action not recognized</omf:Description>
           </omf:Error>
   </omf:ErrorList>
</omf:Header>
```

Expiry time error (full XML):

```
<?xml version="1.0" encoding="UTF-8"?>
<omf:Header xmlns:omf="http://www.opengis.net/schema/omf.xsd"
               xmlns:xlink="http://www.w3.org/1999/xlink"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:schemaLocation="http://www.opengis.net/schema/omf.xsd"
               omf:version="1.0">
       <omf:RefToMessageId>20021215111212@messageid.com</omf:RefToMessageId>
    <omf:Action>
           <omf:Service>urn:ogc:omf:service</omf:Service>
        <omf:Process>MessageError</omf:Process>
    </omf:Action>
    <omf:ErrorList highestSeverity="Error">
       <omf:Error code="ExpiryTimeExpired" severity="Error">
           <omf:Description xml:lang="en-US">
               Action not recognized</omf:Description>
           </omf:Error>
    </omf:ErrorList>
</omf:Header>
```

Security error (XML fragment):

```
<omf:ErrorList omf:highestSeverity="error">
       <omf:Error omf:id="error_107"
                  omf:code="SecurityFailure" omf:severity="Error"
                  omf:location="URI_of_ds:Signature">
           <omf:Description xml:lang="en-US">
              Signature validation failed.
           <omf:Description>
       </omf:Error>
</omf:ErrorList>
```

11 Acknowledgement Messages

An acknowledgement message signals to the requestor that the service has received the message, is able to process it, and that a response is or will be available. Acknowledgement messages are optional but are highly **recommended**.

Acknowledgement messages and error messages are highly analogous in that they both report a status back to the **From** party.

11.1 Acknowledgement Elements

11.1.1 Acknowledgement element

The **Acknowledge** element is very simple in that it contains one child element, namely the **TimeStamp** element.

Normative Header XML schema definition:

```
<element name="Acknowledgment">
       <complexType>
           <sequence>
               <element ref="omf:Timestamp" />
               <any namespace="##other" processContents="lax" minOccurs="0"
                 maxOccurs="unbounded"/>
           </sequence>
       </complexType>
   </element>
```

The **Acknowledgement** element must not exist along with an **ErrorList** element.

If a message contains an **Acknowledgement** element then the message must also contain the following elements and values:

- The **RefToMessageId must** be present and **must** identify the message being acknowledged. It must be have the same value as the original message's **MessageId**.

- The **Service** element **must** be set to **urn:ogc:omf:service**.

- The **Process** element **must** be set to **Acknowledgement**.

11.1.2 TimeStamp element

The **required Timestamp** element is a value representing the time that the message being acknowledged was received by the service generating the acknowledgment message. It must conform to a **dateTime** as defined by XML Schema.

11.2 Acknowledgement example

Example 1:

```
<?xml version="1.0" encoding="UTF-8"?>
<Header xmlns="http://www.opengis.net/schema/omf.xsd"
           version="1.0">
       <RefToMessageId>20021215111212@messageid.com</RefToMessageId>
    <Action>
           <Service>urn:ogc:omf:service</Service>
        <Process>Acknowledgement</Process>
    </Action>
       <Acknowledgement>
        <TimeStamp>2002-08-15T11:12:14</TimeStamp>
       </Acknowledgement>
</Header>
```

Example 2:

```
<?xml version="1.0" encoding="UTF-8"?>
<Header xmlns="http://www.opengis.net/schema/omf.xsd"
           version="1.0">
    <From></From>
    <To></To>
       <MessageId>23432@service_message_id.com</MessageId>
       <RefToMessageId>20021215111212@messageid.com</RefToMessageId>
    <Action>
           <Service>urn:ogc:omf:service</Service>
        <Process>Acknowledgement</Process>
    </Action>
       <Acknowledgement>
        <TimeStamp>2002-08-15T11:12:14</TimeStamp>
       </Acknowledgement>
</Header>
```

12 Payload Details

Zero or more **Payload Containers** MAY be present within a **Message.** If the **Message** contains an application payload, it SHOULD be enclosed within a **Payload Container.**

If there is no application payload within the **Message Package** then a **Payload Container** MUST NOT be present. The contents of each **Payload Container** MUST be identified in the message **Manifest** element within the Header.

This specification makes no provision, nor limits in any way, the structure or content of application payloads. Payloads MAY be simple plain text objects or complex nested multipart objects. The specification of the structure and composition of payload objects is the prerogative of the organization defining the business process or information exchange using the OGC **Message Service**.

12.1.1 Example of a Payload Container

The following fragment represents an example of a **Payload Container** and a payload using MIME.

```
Content-ID: <domainname.example.com>         MIME
Content-Type: application/xml
                                                         Payload
                                                         Container
<CityModel>
   <CityMember>                              Payload
   ……
   </CityMember>
</CityModel>
```

13 Status Messages

This element is used when you want to know what the status of a specific message processing. This is extremely useful for the Notification Service. I have suggested using OMF for the Notification Service, because the overlap is important.
Here's the semantic of the status type:

- **UnAuthorized**: the Message Status Request is not authorized or accepted.
- **NotRecognized**: the message identified by the **RefToMessageId** element in the **StatusResponse** element is not recognized.
- **Received**: the message identified by the **RefToMessageId** element in the **StatusResponse** element has been received by the MSH.
- **Processed**: the message identified by the **RefToMessageId** element in the **StatusResponse** element has been processed by the MSH.
- **Forwarded**: the message identified by the **RefToMessageId** element in the **StatusResponse** element has been forwarded by the MSH to another MSH

  For the image archive we do not need it the Status services. However, for asynchronous communication, it will be needed.

14 Bindings

14.1 Binding to HTTP, SOAP, and multipart MIME

Bindings are best shown with examples. Thus several examples are given. Note, for clarity and brevity, none of the examples show any binary data sent in the message. The phrase "... Binary image data not shown..." replaces the binary data.

Example 1:

This example performs a WFS transaction packaged in OMF.

To show that the message packaging style is valuable we are going to insert a number of features into the WFS using GML. The WFS transaction is an example taken directly from the OpenGIS Web Feature Server Specification.

Request

```
Request:
POST /servlet/handler HTTP/1.1
Host: www.example.com
soapAction: ""
Content-type: multipart/related; MimeBoundary="MBndry12345678"; type="text/xml";

--MBndry12345678
Content-ID: message_header
Content-Type: text/xml

<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:xlink="http://www.w3.org/1999/xlink"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
     xmlns:omf="http://www.opengis.net/schema/omf.xsd"
     xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
               http://www.opengis.net/schema/omf.xsd”>
   <soap:Header>
      <omf:Header omf:version="1.0">
         <omf:From>
            <omf:PartyId>urn:duns:123456789</omf:PartyId>
         </omf:From>
         <omf:To>
            <omf:PartyId>urn:duns:912345678</omf:PartyId>
         </omf:To>
         </omf:Action>
            <omf:Service>urn:ogc:services:wfs</omf:Service>
            <omf:Process>Transaction</omf:Process>
         </omf:Action>
         <omf:MessageId>20001209-133003-28572@example.com</omf:MessageId>
         <omf:Timestamp>2001-02-15T11:12:12</omf:Timestamp>
      </omf:Header>
   </soap:Header>
   <soap:Body>
      <omf:Manifest omf:id="Manifest">
         <omf:Reference omf:id="item01"
                        xlink:href="wfs-transaction"
```

```
                       xlink:role="Data" xlink:type="simple">
            <omf:Schema location="http://www.opengis.net/wfs/transaction.dtd"
                       version="1.0" />
         </omf:Reference>
      <omf:Manifest>
   </soap:Body>
</soap:Envelope>

--MBndry12345678
Content-ID: wfs-transaction
Content-Type: text/xml

<Transaction>
<!-- This example is copied directly from the WFS Specification. -->
<Insert>
   <INWATERA_1M>
      <INWATERA_1M.ID>150<\INWATERA_1M.ID>
      <INWATERA_1M.F_CODE>ABCDE</INWATERA_1M.F_CODE>
      <INWATERA_1M.HYC>152</INWATERA_1M.HYC>
      <INWATERA_1M.TILE_ID>250</INWATERA_1M.TILE_ID>
      <INWATERA_1M.FAC_ID>111</INWATERA_1M.FAC_ID>
      <INWATERA_1M.WKB_GEOM>
         <gml:Polygon gid="1" srsName="http://…/epsg.xml#EPSG:4326">
            <gml:outerBoundaryIs>
               <gml:LinearRing>
                  <gml:coordinates>
                  -98.54,24.26 ...
                  </gml:coordinates>
               </gml:LinearRing>
            </gml:outerBoundaryIs>
         </gml:Polygon>
      </INWATERA_1M.WKB_GEOM>
   </INWATERA_1M>
   <INWATERA_1M>
      <INWATERA_1M.ID>111<\INWATERA_1M.ID>
      <INWATERA_1M.F_CODE>FGHIJ</INWATERA_1M.F_CODE>
      <INWATERA_1M.HYC>222</INWATERA_1M.HYC>
      <INWATERA_1M.TILE_ID>333</INWATERA_1M.TILE_ID>
      <INWATERA_1M.FAC_ID>444</INWATERA_1M.FAC_ID>
      <INWATERA_1M.WKB_GEOM>
         <gml:Polygon gid="1" srsName="http://…/epsg.xml#EPSG:4326">
            <gml:outerBoundaryIs>
               <gml:LinearRing>
                  <gml:coordinates>
                  -99.99,22.22 ...
                  </gml:coordinates>
               </gml:LinearRing>
            </gml:outerBoundaryIs>
         </gml:Polygon>
      </INWATERA_1M.WKB_GEOM>
   </INWATERA_1M>
</Insert>
</Transaction>

--MBndry12345678--
```

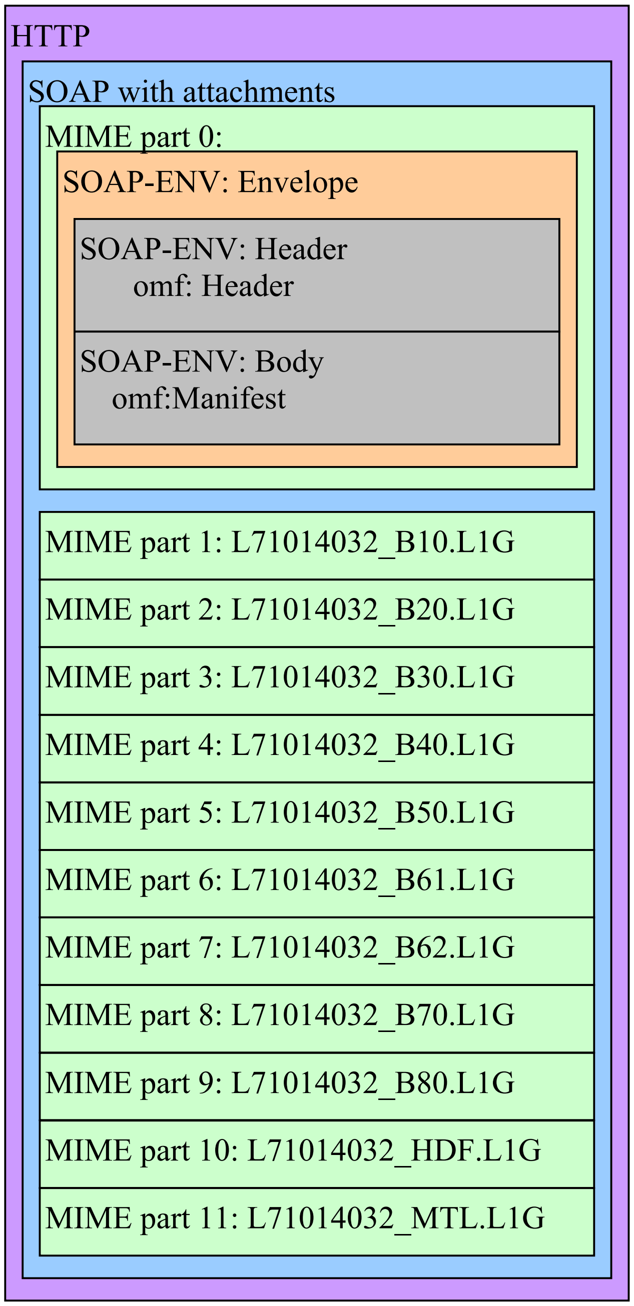
Example 2:

This example inserts multiple related resources into the WCS.

We want to insert into the image archive a Landsat image which consists of 9 files. There are nine band files, an HDF file, and a native metadata Landsat file. For our example we'll use the following files:

L71014032_B10.L1G	Band 1
L71014032_B20.L1G	Band 2
L71014032_B30.L1G	Band 3
L71014032_B40.L1G	Band 4
L71014032_B50.L1G	Band 5
L71014032_B61.L1G	Band 6
L72014032_B62.L1G	Band 7
L72014032_B70.L1G	Band 8
L72014032_B80.L1G	Band 9
L71014032_HDF.L1G	HDF
L71014032_MTL.L1G	Metadata

The image below describes how the packaging is done.


HTTP
SOAP with attachments
MIME part 0:
SOAP-ENV: Envelope
SOAP-ENV: Header
omf: Header
SOAP-ENV: Body
omf:Manifest
MIME part 1: L71014032_B10.L1G
MIME part 2: L71014032_B20.L1G
MIME part 3: L71014032_B30.L1G
MIME part 4: L71014032_B40.L1G
MIME part 5: L71014032_B50.L1G
MIME part 6: L71014032_B61.L1G
MIME part 7: L71014032_B62.L1G
MIME part 8: L71014032_B70.L1G
MIME part 9: L71014032_B80.L1G
MIME part 10: L71014032_HDF.L1G
MIME part 11: L71014032_MTL.L1G

The example HTTP POST message shows how the files should be packaged in the message. The example is a mapping (binding) from the OMF normative XML Schema to SOAP with attachments using HTTP and multipart MIME.

Request

```
POST /servlet/handler HTTP/1.1
Host: www.example.com
soapAction: ""
Content-type: multipart/related; MimeBoundary="MBndry12345678"; type="text/xml";

--MBndry12345678
Content-ID: message_header
Content-Type: text/xml

<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:xlink="http://www.w3.org/1999/xlink"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
     xmlns:omf="http://www.opengis.net/schema/omf.xsd"
     xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
               http://www.opengis.net/schema/omf.xsd”>
   <soap:Header>
      <omf:Header omf:version="1.0">
         <omf:From>
            <omf:PartyId>urn:duns:123456789</omf:PartyId>
         </omf:From>
         <omf:To>
            <omf:PartyId>urn:duns:912345678</omf:PartyId>
         </omf:To>
         </omf:Action>
            <omf:Service>urn:ogc:services:wcs</omf:Service>
            <omf:Process>Insert</omf:Process>
         </omf:Action>
         <omf:MessageId>20001209-133003-28572@example.com</omf:MessageId>
         <omf:Timestamp>2001-02-15T11:12:12</omf:Timestamp>
      </omf:Header>
   </soap:Header>
   <soap:Body>
      <omf:Manifest>
         <omf:Reference omf:id="item01"
                        xlink:href="L71014032_B10.L1G"
                        xlink:role="Data" xlink:type="simple">
         </omf:Reference>
         <omf:Reference omf:id="item02"
                        xlink:href="L71014032_B20.L1G"
                        xlink:role="Data" xlink:type="simple">
         </omf:Reference>
         <omf:Reference omf:id="item03"
                        xlink:href="L71014032_B30.L1G"
                        xlink:role="Data" xlink:type="simple">
         </omf:Reference>
         <omf:Reference omf:id="item04"
                        xlink:href="L71014032_B40.L1G"
                        xlink:role="Data" xlink:type="simple">
         </omf:Reference>
         <omf:Reference omf:id="item05"
                        xlink:href="L71014032_B50.L1G"
                        xlink:role="Data" xlink:type="simple">
         </omf:Reference>
         <omf:Reference omf:id="item06"
                        xlink:href="L71014032_B61.L1G"
                        xlink:role="Data" xlink:type="simple">
         </omf:Reference>
         <omf:Reference omf:id="item07"
                        xlink:href="L71014032_B62.L1G"
                        xlink:role="Data" xlink:type="simple">
         </omf:Reference>
         <omf:Reference omf:id="item08"
                        xlink:href="L71014032_B70.L1G"
                        xlink:role="Data" xlink:type="simple">
         </omf:Reference>
         <omf:Reference omf:id="item09"
                        xlink:href="L71014032_B80.L1G"
                        xlink:role="Data" xlink:type="simple">
         </omf:Reference>
         <omf:Reference omf:id="item10"
                        xlink:href="L71014032_HDF.L1G"
                        xlink:role="Data" xlink:type="simple">
         </omf:Reference>
         <omf:Reference omf:id="item11"
                        xlink:href="L71014032_MTL.L1G"
                        xlink:role="Data" xlink:type="simple">
            <omf:Description xml:lang="en-US">Native Landsat
                            metadata</omf:Description>
         </omf:Reference>
         <omf:Reference omf:id="item12"
                        xlink:href="item12-metadata"
                        xlink:role="Metadata" xlink:type="simple">
            <omf:Schema omf:location="http://www.opengis.net/iso19115.rdfs"
                        omf:version="1.0" />
            <omf:Description xml:lang="en-US">
                This refences the special metadata payload.</omf:Description>
         </omf:Reference>
      </omf:Manifest>
   </soap:Body>
</soap:Envelope>

--MBndry12345678
Content-ID: item12-metadata
Content-Type: text/xml
This payload (MIME part) describes contains the metadata that describes each
payload.  Whether we use ISO121195, RDF, or something else to describe the data
this payload must be processed.

--MBndry12345678
Content-ID: L71014032_B10.L1G
Content-Type: image
   ... Binary image data not shown...

--MBndry12345678
Content-ID: L71014032_B20.L1G
Content-Type: image
   ... Binary image data not shown...

--MBndry12345678
Content-ID: L71014032_B30.L1G
Content-Type: image
   ... Binary image data not shown...

--MBndry12345678
Content-ID: L71014032_B40.L1G
Content-Type: image
```

```
   ... Binary image data not shown...

--MBndry12345678
Content-ID: L71014032_B50.L1G
Content-Type: image
   ... Binary image data not shown...

--MBndry12345678
Content-ID: L71014032_B61.L1G
Content-Type: image
   ... Binary image data not shown...

--MBndry12345678
Content-ID: L71014032_B62.L1G
Content-Type: image
   ... Binary image data not shown...

--MBndry12345678
Content-ID: L71014032_B70.L1G
Content-Type: image
   ... Binary image data not shown...

--MBndry12345678
Content-ID: L71014032_B80.L1G
Content-Type: image
   ... Binary image data not shown...

--MBndry12345678
Content-ID: L71014032_HDF.L1G
Content-Type: image
   ... Binary image data not shown...

--MBndry12345678
Content-ID: L71014032_MTL.L1G
Content-Type: text/plain

GROUP = LPGS_METADATA_FILE
  GROUP = METADATA_FILE_INFO
    REQUEST_ID = "LLITE"
    PRODUCT_CREATION_TIME = 2002-07-19T21:07:11Z
    STATION_ID = "EDC"
    LANDSAT7_XBAND = "0"
    GROUND_STATION = "GNC"
    LPS_PROCESSOR_NUMBER = 0
    DATEHOUR_CONTACT_PERIOD = "0125515"
    SUBINTERVAL_NUMBER = "00"
  END_GROUP = METADATA_FILE_INFO
  GROUP = PRODUCT_METADATA
    PRODUCT_TYPE = "L1G"
    EPHEMERIS_TYPE = "PREDICTIVE"
    SPACECRAFT_ID = "Landsat7"
    SENSOR_ID = "ETM+"
    ACQUISITION_DATE = 2001-09-12
    WRS_PATH = 014
    STARTING_ROW = 032
    ENDING_ROW = 032
    BAND_COMBINATION = "123456678"
    PRODUCT_UL_CORNER_LAT = 41.2998960
    PRODUCT_UL_CORNER_LON = -76.3282345
  END_GROUP = PROJECTION_PARAMETERS
   ... More Landsat metadata not shown...
--MBndry12345678--
```

Example 3:

This example inserts a JPEG image in to the image archive. It does not contain any metadata.

Request

```
POST /image-archive HTTP/1.1
Host: www.example.com
soapAction: ""
content-type: multipart/related; type="text/xml"; boundary="----PartBoundary"
content-length: 112275

------PartBoundary
Content-Type: text/xml

<?xml version="1.0" encoding="UTF-8"?>
<soap-env:Envelope xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:omf="http://www.opengis.net/schema/omf.xsd"
    xmlns:xlink="http://www.w3.org/1999/xlink">
<soap-env:Header>
    <omf:Header>
    <omf:Action>
        <omf:Service>urn:opengis:services:image-archive</omf:Service>
        <omf:Process>insert</omf:Process>
    </omf:Action>
    <omf:MessageId>d883f8dc-1b40-4a9c-be72-4a4e015d4b96</omf:MessageId>
    </omf:Header>
</soap-env:Header>
<soap-env:Body>
    <omf:Manifest>
        <omf:Reference xlink:href="cid:content1040679532663"
            xlink:role="urn:opengis:services:image-archive:roles:data"
            xlink:type="locator"/>
    </omf:Manifest>
</soap-env:Body>
</soap-env:Envelope>
------PartBoundary
Content-Type: image/jpeg
Content-Id: content1040679532663
Content-Disposition: attachment; filename=SharbatGula2.jpg
content-length: 111310

   ... Binary image data not shown...
------PartBoundary--
```

Response

```
------=_Part_1_3786945.1040679560663
Content-Type: text/xml

<?xml version="1.0" encoding="UTF-8"?>
<soap-env:Envelope
    xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:omf="http://www.opengis.net/schema/omf.xsd"
    xmlns:xlink="http://www.w3.org/1999/xlink">
    <soap-env:Header>
```

```
        <omf:Header>
    <omf:Action>
        <omf:Service>urn:opengis:services:image-archive</omf:Service>
        <omf:Process>transaction-response</omf:Process>
    </omf:Action>
    <omf:MessageId>215c4768-dd7b-4114-a3c4-8fb33ba7e8e4</omf:MessageId>
    <omf:RefToMessageId>d883f8dc-1b40-4a9c-be72-a4e015d4b96</omf:RefToMessageId>
    </omf:Header>
    </soap-env:Header>
    <soap-env:Body>
        <omf:Manifest>
        <omf:Reference xlink:href="cid:Content1040679300245"
    xlink:role="urn:opengis:services:image-archive:roles:transaction-response"
    xlink:type="locator"/>
        </omf:Manifest>
    </soap-env:Body>
    </soap-env:Envelope>

------=_Part_1_3786945.1040679560663
Content-Type: text/xml
Content-Id: Content1040679300245


<?xml version="1.0" encoding="UTF-8"?>
<ia:TransactionResponse xmlns:ia="http://www.opengis.net/iarchive">
   <ia:InsertResult handle="handleData0">
      <ia:InsertedContent msg-content-ref="content1040679532663"
           oid="urn:uuid:a023cd00-0284-40f4-b369-14f78cffacce" />
      <ia:Status>SUCCESS</ia:Status>
   </ia:InsertResult>
   <ia:Status>SUCCESS</ia:Status>
</ia:TransactionResponse>

------=_Part_1_3786945.1040679560663--
```

Example 4:

This example adds a TIFF and DAT file to the image archive. It also shows how metadata could be harvested from the DAT and sent as RDF.

This message has four multipart MIME parts. The first, as required, is the SOAP part while the second contains the Image Archive transaction describing the other payloads and what to do with them.

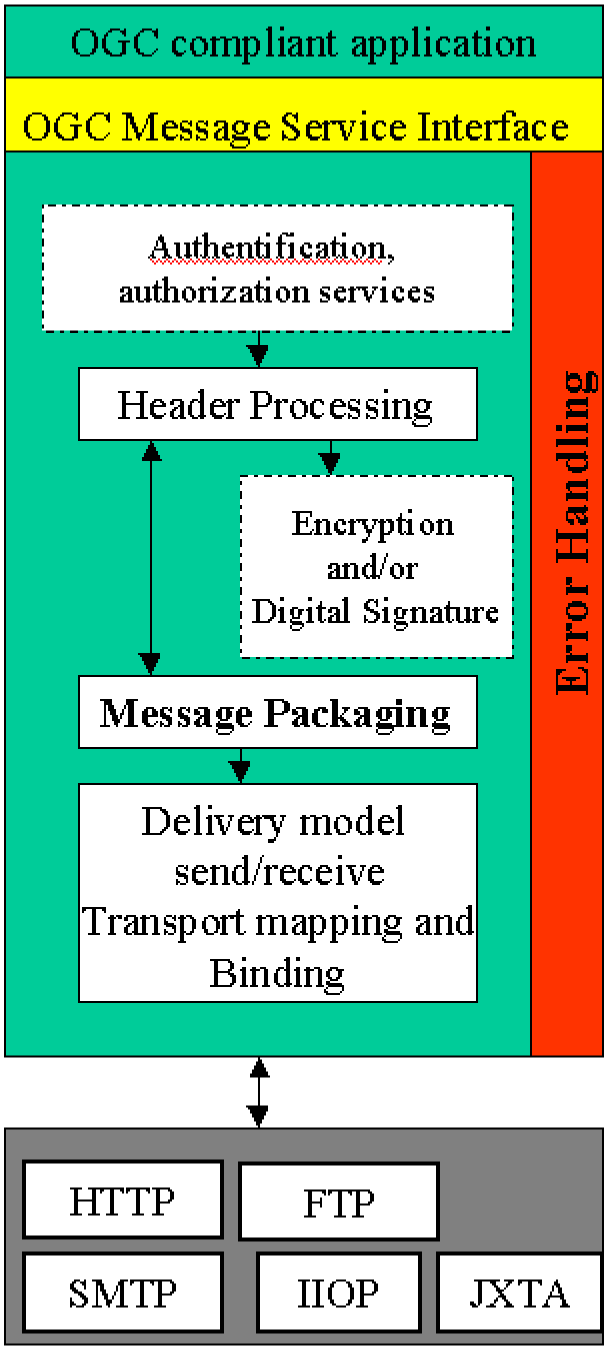
Request

```
POST /image-archive HTTP/1.1
content-type: multipart/related; type="text/xml"; boundary="----=PartBoundary”
content-length: 1342929
soapaction: ""
host: localhost:8080

------=PartBoundary
Content-Type: text/xml

<?xml version="1.0" encoding="UTF-8"?>
<soap-env:Envelope xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/"
       xmlns:omf="http://www.opengis.net/schema/omf.xsd"
      xmlns:xlink="http://www.w3.org/1999/xlink">
   <soap-env:Header>
       <omf:Header>
           <omf:Action>
               <omf:Service>urn:opengis:services:image-archive</omf:Service>
               <omf:Process>transaction</omf:Process>
           </omf:Action>
           <omf:MessageId>fee06ff0-f23f-47f1-9138-8de11bb45494</omf:MessageId>
       </omf:Header>
   </soap-env:Header>
   <soap-env:Body>
       <omf:Manifest>
           <omf:Reference xlink:href="cid:transaction_request_payload"
      xlink:role="urn:opengis:services:image-archive:roles:transaction-request"
      xlink:type="locator"/>
           <omf:Reference xlink:href="cid:DATFile0"
              xlink:role="urn:opengis:services:image-archive:roles:data"
              xlink:type="locator"/>
           <omf:Reference xlink:href="cid:TIFFFile0"
              xlink:role="urn:opengis:services:image-archive:roles:data"
              xlink:type="locator"/>
           <omf:Reference xlink:type="locator"
               xlink:href="cid:Metadata-ca8231ff-b9bc-4a14-86d4-48d225b2fdf9"
               xlink:role="urn:opengis:services:image-archive:roles:metadata"/>
       </omf:Manifest>
   </soap-env:Body>
</soap-env:Envelope>
------=PartBoundary
Content-Type: text/xml
Content-Id: transaction_request_payload
content-length: 935

<?xml version="1.0" encoding="UTF-8"?>
<ia:Transaction xmlns:ia="http://www.opengis.net/iarchive"
       xmlns:omf="http://www.opengis.net/schema/omf.xsd"
       xmlns:xlink="http://www.w3.org/1999/xlink"
       handle="TransactionHandle1040693031153">
   <ia:Insert handle="Insert692c16da-7674-4f74-adbd-5a6cf91b6628">
      <omf:Reference xlink:href="DATFile0"
             xlink:role="urn:opengis:services:image-archive:roles:data"
             xlink:type="locator" />
   </ia:Insert>
   <ia:Insert handle="Insert516f726a-7352-42f5-8f7e-1556c3d03d72">
      <omf:Reference xlink:href="TIFFFile0"
             xlink:role="urn:opengis:services:image-archive:roles:data"
             xlink:type="locator" />
   </ia:Insert>
   <ia:Insert handle="InsertOp14b9d4c8-a227-47eb-a002-0db17bad3263">
      <omf:Reference xlink:href="Metadata-ca8231ff-b9bc-4a14-86d4-48d225b2fdf9"
             xlink:role="urn:opengis:services:image-archive:roles:metadata"
             xlink:type="locator" />
   </ia:Insert>
</ia:Transaction>
------=PartBoundary
Content-Type: text/plain
Content-Id: DATFile
Content-Disposition: attachment; filename=uav1.dat
content-length: 177

Date: 4/18/2002
Time: 7:54:07 AM
Latitude: 3646.7210 N
Longitude: 07615.9970 W
```

```
Altitude:  704.4 M
Speed: 000.0
Course: 000.0
Pitch: 0.000000000000
Roll: 0.000000000000
------=PartBoundary
Content-Type: image/tiff
Content-Id: TIFFFile
Content-Disposition: attachment; filename=uav1.tif
content-length: 1334556

    ... Binary image data not shown...

------=PartBoundary
Content-Type: text/xml
Content-Id: Metadata-ca8231ff-b9bc-4a14-86d4-48d225b2fdf9
content-length: 5442

<rdf:RDF
  xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
  xmlns:NS0='urn:opengis:taxomomy:sensor#'
  xmlns:NS1='urn:opengis:services:sps:taxomomy#'
  xmlns:NS2='http://www.opengis.net/wcs#'
  xmlns:NS3='http://www.fgdc.org/remotesensing-extension#'
  xmlns:rdfs='http://www.w3.org/2000/01/rdf-schema#'
  xmlns:NS4='http://www.geodatasystems.com/uav#'>
  <rdf:Description rdf:about='urn:uuid:b0034e91-1397-47d7-8d2b-821811e10325'>
    <rdf:type rdf:resource='http://www.fgdc.org/remotesensing-
extension#Positional_Information'/>
    <NS3:projection_center_x_position>-
76.26661666666666</NS3:projection_center_x_position>

<NS3:projection_center_y_position>36.77868333333333</NS3:projection_center_y_pos
ition>
    <NS3:projection_center_z_position>704.4</NS3:projection_center_z_position>
    <NS3:kappa>0.0</NS3:kappa>
    <NS3:omega>-0.0</NS3:omega>
    <NS3:phi>0.0</NS3:phi>
    <NS3:roll>0.0</NS3:roll>
    <NS3:pitch>0.0</NS3:pitch>
    <NS3:attitude_angular_unit>decimal degree</NS3:attitude_angular_unit>
    <NS3:projection_center_unit>decimal degree</NS3:projection_center_unit>
    <NS4:speed>0.0</NS4:speed>
    <NS4:course>0.0</NS4:course>
    <NS3:rotationSequence>123</NS3:rotationSequence>
  </rdf:Description>
  <rdf:Description rdf:about='DATFile0'>
    <rdf:type
        rdf:resource='http://www.geodatasystems.com/uav#UAVNavigationInfo'/>
    <NS4:navigation_info_of rdf:resource='TIFFFile0'/>
    <NS4:original_filename>uav1.dat</NS4:original_filename>
    <NS2:nativeFormat rdf:resource='http://www.geodatasystems.com/uav#DAT'/>
  </rdf:Description>
  <rdf:Description rdf:about='http://www.opengis.net/wcs#TIFF_6.0'>
    <rdf:type rdf:resource='http://www.opengis.net/wcs#Format'/>
    <NS2:formatName>TIFF 6.0</NS2:formatName>
    <NS2:nativeFormat>image/tiff</NS2:nativeFormat>
    <NS2:nativeFormat>image/tif</NS2:nativeFormat>
    <NS2:abstract>TIFF format version 6.0</NS2:abstract>
  </rdf:Description>
  <rdf:Description rdf:about='http://www.geodatasystems.com/uav#DAT'>
    <rdf:type rdf:resource='http://www.opengis.net/wcs#Format'/>
    <NS2:formatName>DAT</NS2:formatName>
    <NS2:nativeFormat>text/plain</NS2:nativeFormat>
    <NS2:abstract>Format used by GDS to store navigation metadata for UAV
        video image. The DAT file name uses the time stamp of the acquisition
        followed by the extension &apos;.DAT&apos;.</NS2:abstract>
  </rdf:Description>
  <rdf:Description rdf:about='TIFFFile0'>
    <rdfs:label>uav1.tif</rdfs:label>
    <rdfs:comment>Video imagery uav1.tif acquired for by GDS UAV</rdfs:comment>
    <NS4:original_filename>uav1.tif</NS4:original_filename>
    <NS2:nativeFormat
        rdf:resource='http://www.opengis.net/wcs#TIFF_6.0'/>
    <rdf:type
        rdf:resource='http://www.opengis.net/wcs#RectifiableGridCoverage'/>
    <rdf:type
        rdf:resource='http://www.fgdc.org/remotesensing-
extension#Georeferenceable_Raster'/>
    <rdf:type rdf:resource='urn:opengis:services:sps:taxomomy#Observation'/>
    <NS3:has_instrument_georeferencing
       rdf:resource='urn:uuid:d70d872d-9aef-413d-8e9d-285776733629'/>
    <NS2:acquisitionTime>04/18/2002 07:54:07 AM</NS2:acquisitionTime>
    <NS4:navigation_info rdf:resource='DATFile0'/>
    <NS3:instrument_information
        rdf:resource='urn:uuid:2be659b4-c679-4688-bdca-141a302b9941'/>
    <NS3:video-member-of
        rdf:resource='urn:uuid:7eb2bdc7-ba3e-438f-8a2d-a807d24f9b48'/>
    <NS3:image-member-of
        rdf:resource='urn:uuid:7eb2bdc7-ba3e-438f-8a2d-a807d24f9b48'/>
    <NS1:taskID>can-1</NS1:taskID>
    <NS4:averageGroundElevation>32.0</NS4:averageGroundElevation>
    <NS4:averageGroundElevationUnit>m</NS4:averageGroundElevationUnit>
  </rdf:Description>
  <rdf:Description rdf:about='urn:uuid:7eb2bdc7-ba3e-438f-8a2d-a807d24f9b48'>
    <rdf:type rdf:resource='http://www.fgdc.org/remotesensing-
extension#ImageCollection'/>
    <rdf:type rdf:resource='http://www.fgdc.org/remotesensing-
extension#VideoCollection'/>
    <rdf:type
rdf:resource='urn:opengis:services:sps:taxomomy#ObservationCollection'/>
    <NS1:taskID>can-1</NS1:taskID>
    <NS3:collectedByMission
        rdf:resource='urn:uuid:5db4cd00-0c8c-4ab4-a226-cfbf6f9c3315'/>
    <NS3:image-member rdf:resource='TIFFFile0'/>
    <NS3:video-member rdf:resource='TIFFFile0'/>
    <NS3:instrument_information
        rdf:resource='urn:uuid:2be659b4-c679-4688-bdca-141a302b9941'/>
  </rdf:Description>
  <rdf:Description rdf:about='urn:uuid:d70d872d-9aef-413d-8e9d-285776733629'>
    <rdf:type rdf:resource='http://www.fgdc.org/remotesensing-
extension#Instrument_Specific_Georeferencing'/>
    <NS3:position rdf:resource='urn:uuid:b0034e91-1397-47d7-8d2b-821811e10325'/>
  </rdf:Description>
  <rdf:Description rdf:about='urn:uuid:2be659b4-c679-4688-bdca-141a302b9941'>
    <rdf:type rdf:resource='urn:opengis:taxomomy:sensor#VideoCamera'/>
    <NS0:focalLength>28.0</NS0:focalLength>
    <NS0:chipSizeX>8.1</NS0:chipSizeX>
    <NS0:chipSizeY>6.33</NS0:chipSizeY>
    <NS0:model>C814</NS0:model>
    <NS0:manufacturer>Sony</NS0:manufacturer>
  </rdf:Description>
  <rdf:Description rdf:about='urn:uuid:5db4cd00-0c8c-4ab4-a226-cfbf6f9c3315'>
    <rdf:type rdf:resource='http://www.fgdc.org/remotesensing-
extension#Mission'/>
    <NS3:missionDescription>An example mission</NS3:missionDescription>
    <NS1:taskID>can-1</NS1:taskID>
    <NS3:collects rdf:resource='urn:uuid:7eb2bdc7-ba3e-438f-8a2d-a807d24f9b48'/>
  </rdf:Description>
</rdf:RDF>
------=PartBoundary--
```

14.2 OGC Message Handling (TO DO)

This diagram shows how the server can handle messages.


OGC compliant application
OGC Message Service Interface
Authentification, authorization services
Header Processing
Encryption and/or Digital Signature
Message Packaging
Delivery model send/receive Transport mapping and Binding
Error Handling
HTTP
FTP
SMTP
IIOP
JXTA

Annex A: OMF XML Schema (Normative)

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v5 rel. 2 U (http://www.xmlspy.com) by Keens Steven
(PCI Geomatics) -->
<!-- Create by Stephane Fellah and Steven Keens at PCI Geomatics Inc. -->
<schema targetNamespace="http://www.opengis.net/schema/omf.xsd"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:omf="http://www.opengis.net/schema/omf.xsd"
xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="qualified" version="1.0">
    <import namespace="http://www.w3.org/1999/xlink"
schemaLocation="..\..\gml\3.0\xlink\xlinks.xsd"/>
    <import namespace=http://www.w3.org/XML/1998/namespace
            schemaLocation="xml.xsd"/>
    <element name="Message">
        <complexType>
            <sequence>
                <element ref="omf:HeaderContainer"/>
                <element ref="omf:PayloadContainer" minOccurs="0"
                    maxOccurs="unbounded"/>
            </sequence>
        </complexType>
    </element>
    <element name="HeaderContainer" type="omf:HeaderContainerType"
substitutionGroup="omf:Envelope"/>
    <element name="Header">
        <annotation>
            <documentation>Encapsulates the OMF header
                           information.</documentation>
        </annotation>
        <complexType>
            <sequence>
                <element ref="omf:From" minOccurs="0"/>
                <element ref="omf:To" minOccurs="0"/>
                <element ref="omf:MessageId" minOccurs="0"/>
                <element ref="omf:RefToMessageId" minOccurs="0"/>
                <element ref="omf:Timestamp" minOccurs="0"/>
                <element ref="omf:ExpiryTime" minOccurs="0"/>
                <element ref="omf:Action"/>
                <any namespace="##other" processContents="lax" minOccurs="0"
                    maxOccurs="unbounded"/>
            </sequence>
            <attribute ref="omf:id"/>
            <attribute ref="omf:version" use="required"/>
            <anyAttribute namespace="##other" processContents="lax"/>
        </complexType>
    </element>
    <element name="MessageId" type="omf:non-empty-string">
        <annotation>
            <documentation>Message identifier</documentation>
        </annotation>
    </element>
    <element name="RefToMessageId" type="omf:non-empty-string">
        <annotation>
            <documentation>Refers to an earlier message</documentation>
        </annotation>
    </element>
    <element name="Timestamp" type="dateTime"/>
    <element name="ExpiryTime" type="dateTime"/>
    <element name="Actor" type="omf:ActorType">
        <annotation>
            <documentation>An actor represents a requester or provider who might
                request or offer a service</documentation>
        </annotation>
    </element>
    <element name="Action">
        <annotation>
            <documentation>Tells the recipient what to do with the
                message.</documentation>
        </annotation>
        <complexType>
            <complexContent>
                <extension base="omf:ActionType"/>
            </complexContent>
        </complexType>
    </element>
    <element name="Service" type="anyURI"/>
    <element name="Process" type="anyURI"/>
    <element name="From" type="omf:ActorType" substitutionGroup="omf:Actor"/>
    <element name="To" type="omf:ActorType" substitutionGroup="omf:Actor"/>
    <element name="PartyId" type="omf:IdentifierType">
        <annotation>
            <documentation>Identifications of the party in all domains in
                support of intermediaries and destinations that may give
                preference to a particular identification system.</documentation>
        </annotation>
    </element>
    <element name="Manifest">
        <annotation>
            <documentation>Lists all items in the payload</documentation>
        </annotation>
        <complexType>
            <sequence>
                <element ref="omf:Reference" maxOccurs="unbounded"/>
            </sequence>
            <attribute ref="omf:id" use="optional"/>
            <attribute ref="omf:version" use="optional"/>
        </complexType>
    </element>
    <element name="Reference">
        <complexType>
            <complexContent>
                <extension base="omf:ReferenceType"/>
            </complexContent>
        </complexType>
    </element>
    <element name="Schema">
        <complexType>
            <attribute name="location" type="anyURI" use="required"/>
            <attribute ref="omf:version" use="optional"/>
        </complexType>
    </element>
    <element name="ErrorList">
        <complexType>
            <sequence>
                <element ref="omf:Error" maxOccurs="unbounded"/>
            </sequence>
            <attribute name="highestSeverity" type="omf:severity.type"
use="required"/>
        </complexType>
    </element>
    <element name="Error">
        <complexType>
            <sequence>
                <element ref="omf:Description" minOccurs="0"/>
                <any namespace="##other" processContents="lax" minOccurs="0"
                                maxOccurs="unbounded"/>
            </sequence>
            <attribute ref="omf:id"/>
            <attribute name="context" type="anyURI"
                    default="urn:opengis:services:msg-service"/>
            <attribute name="code" use="required">
                <simpleType>
                    <restriction base="omf:non-empty-string">
                        <enumeration value="NotRecognized"/>
                        <enumeration value="NotSupported"/>
                        <enumeration value="ExpiryTimeLapsed"/>
                        <enumeration value="Inconsistent"/>
                        <enumeration value="DeliveryFailure"/>
                        <enumeration value="SecurityFailure"/>
                        <enumeration value="LinkProblem"/>
                        <enumeration value="Unknown"/>
                    </restriction>
                </simpleType>
            </attribute>
            <attribute name="severity" type="omf:severity.type" use="required"/>
            <attribute name="location" type="omf:non-empty-string"/>
            <anyAttribute namespace="##other" processContents="lax"/>
        </complexType>
    </element>
    <element name="Acknowledgement">
        <complexType>
            <sequence>
                <element ref="omf:Timestamp"/>
                <element ref="omf:From" minOccurs="0"/>
                <any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
            </sequence>
        </complexType>
    </element>
    <element name="StatusResponse">
        <complexType>
            <sequence>
                <element ref="omf:RefToMessageId"/>
                <any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
            </sequence>
            <attribute name="messageStatus" type="omf:messageStatus.type"
use="required"/>
        </complexType>
    </element>
    <element name="StatusRequest">
        <complexType>
            <sequence>
                <element ref="omf:RefToMessageId"/>
                <any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
            </sequence>
        </complexType>
    </element>
    <element name="Description">
        <complexType>
            <simpleContent>
                <extension base="omf:non-empty-string">
```

```
                    <attribute ref="xml:lang" use="required"/>
                </extension>
            </simpleContent>
        </complexType>
    </element>
    <element name="PayloadContainer" substitutionGroup="omf:Envelope">
        <complexType>
            <complexContent>
                <extension base="omf:PayloadContainerType">
                    <sequence>
                        <element name="Payload" maxOccurs="unbounded"/>
                        <any namespace="##other" processContents="lax"
minOccurs="0" maxOccurs="unbounded"/>
                    </sequence>
                </extension>
            </complexContent>
        </complexType>
    </element>
    <element name="Payload"/>
    <element name="Envelope" type="omf:EnvelopeType"/>
    <complexType name="ActorType">
        <annotation>
            <documentation>An actor represents a requester or provider who might
request or offer a service</documentation>
        </annotation>
        <sequence>
            <element ref="omf:PartyId" maxOccurs="unbounded"/>
            <element name="Role" type="string" minOccurs="0">
                <annotation>
                    <documentation>Identifies an authorized role of the party
sending and.or receiving the message.</documentation>
                </annotation>
            </element>
        </sequence>
    </complexType>
    <complexType name="ActionType">
        <sequence>
            <element ref="omf:Service"/>
            <element ref="omf:Process"/>
            <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
        <anyAttribute namespace="##other"/>
    </complexType>
    <complexType name="ReferenceType">
        <sequence>
            <element ref="omf:Schema" minOccurs="0" maxOccurs="unbounded"/>
            <element ref="omf:Description" minOccurs="0" maxOccurs="unbounded"/>
            <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
        <attribute ref="omf:id"/>
        <attributeGroup ref="xlink:locatorLink"/>
        <attribute name="mimeType" type="string" use="optional"/>
        <anyAttribute namespace="##other"/>
    </complexType>
    <complexType name="IdentifierType">
        <simpleContent>
            <extension base="omf:non-empty-string">
                <attribute name="type" type="omf:non-empty-string"/>
            </extension>
        </simpleContent>
    </complexType>
    <complexType name="EnvelopeType"/>
    <complexType name="HeaderContainerType">
```

```
        <sequence>
            <element ref="omf:Header"/>
            <element ref="omf:Manifest" minOccurs="0"/>
            <choice minOccurs="0">
                <element ref="omf:ErrorList"/>
                <element ref="omf:Acknowledgement"/>
                <element ref="omf:StatusRequest"/>
                <element ref="omf:StatusResponse"/>
                <any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
            </choice>
            <any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
        </sequence>
    </complexType>
    <complexType name="PayloadContainerType"/>
    <simpleType name="status.type">
        <restriction base="NMTOKEN">
            <enumeration value="Reset"/>
            <enumeration value="Continue"/>
        </restriction>
    </simpleType>
    <simpleType name="messageStatus.type">
        <restriction base="NMTOKEN">
            <enumeration value="UnAuthorized"/>
            <enumeration value="NotRecognized"/>
            <enumeration value="Received"/>
            <enumeration value="Processed"/>
            <enumeration value="Forwarded"/>
        </restriction>
    </simpleType>
    <simpleType name="non-empty-string">
        <restriction base="string">
            <minLength value="1"/>
        </restriction>
    </simpleType>
    <simpleType name="severity.type">
        <restriction base="NMTOKEN">
            <enumeration value="Warning"/>
            <enumeration value="Error"/>
        </restriction>
    </simpleType>
    <attribute name="id" type="ID"/>
    <attribute name="version" type="omf:non-empty-string"/>
</schema>
```

Bibliography

[1] ISO 31 (all parts), *Quantities and units*.

[2] IEC 60027 (all parts), *Letter symbols to be used in electrical technology*.

[3] ISO 1000, *SI units and recommendations for the use of their multiples and of certain other units*.