International Civil Aviation Organization

AERONAUTICAL TELECOMMUNICATION NETWORK IMPLEMENTATION COORDINATION GROUP – EIGHTH WORKING GROUP MEETING (ATNICG WG/8)

Christchurch New Zealand
28 September – 1 October 2010

Agenda Item 3: Planning the use of XML and SWIM related standards in AMHS environment

MET DATA USING WEB SERVICES OVER AMHS

(Presented by USA)

SUMMARY

Information Paper 4 from the 5th meeting of the ATNICG (ATNICG/5-IP/4) invited the meeting to assess if AMHS can be used for exchange of MET data using XML. This paper provides background information on Web Services SOAP Messaging and the WXXM Weather Data Model and then considers a SOAP binding to X.400.

1. Introduction

1.1 Information Paper 4 from the 5th meeting of the ATNICG (ATNICG/5-IP/4) invited the meeting to assess if AMHS can be used for exchange of MET data using XML. This paper first provides background information on Web Services and the use of SOAP Messaging. This paper next describes the WXXM Weather Data Model and provides an example METAR message using WXXM. This paper finally describes how SOAP messaging could be performed over AMHS by describing the possibility of a SOAP binding to X.400.

2. Discussion

2.1 Web Services (XML, XML Schema, SOAP Messaging, WSDL)

2.1.1 XML

The eXtensible Markup Language (XML) has gained wide acceptance as a means to achieve interoperability among applications independent of the hardware on which the applications are deployed, the operating environment of the applications, or the programming language used to develop the applications. XML is a tool for constructing self-describing "documents" that may be exchanged among applications. Furthermore XML may be used to create "definition" files that define the structure and data types of an XML document. An XML

2.1.1.1 Schema Definition (XSD) file formally describes the elements in an XML document. XSD may be used to define data types that are exchanged in SOAP messaging as well as other aspects of the service provided by a producer application; that is, XSD may be used to define what is in a Web Services Description Language (WSDL) document.

2.1.1.2 A “well formed” XML document follows a simple set of rules. The document must have an XML prologue. Following the prologue the document consists of a series of data elements enclosed by a start tag and an end tag. A tagged element may have an optional attribute value enclosed in quotes.

```
<? Xml version='1.0' encoding='utf-8' standalone='yes' ?>
<!- - An XML document describing staff members - ->

<staffList>
        <employee   empID="001" >
                <firstName>John</firstName>
                <lastName>Doe</lastName>
        </employee>
        <employee   empID="200" >
                <firstName>Mary</firstName>
                <lastName>Doe</lastName>
        </employee>
</staffList>
```

The prologue contains the version of XML in which it was written. The encoding utf-8 indicates that the Western European character set is being used. The standalone=yes keyword indicates that an external definition file is not used to validate this document.

The second line is a comment line. The next line is the root element staffList which encloses all other elements in the document. All of the elements in the example have an explicit start and end tag. XML allows another pattern however if there are no nested elements. We may list the tag once and include the / without listing the tag again. For example if we wanted to reserve an employee ID in the above example we could use the following.

```
<employee  empID="001"  />
```

2.1.1.3 The above example is self contained and thus does not contain any data element tags that conflict with others. But what if the company merges with another company which contains different information for each employee; that is the employee element is defined differently. This situation is handled in XML through the use of a *namespace.*

```
<? Xml version='1.0' encoding='utf-8' standalone='yes' ?>
<!- - An XML document describing staff members - ->

<bci:staffList  xmlns:bci='www.bcisse.com/staff'>
        <bci:employee   empID="100" >
                <bci:firstName>John</bci:firstName>
                <bci:lastName>Doe</bci:lastName>
        </bci:employee>
<bci:employee   empID="200" >
                <bci:firstName>Mary</bci:firstName>
                <bci:lastName>Doe</bci:lastName>
        </bci:employee>
</bci:staffList>
```

The xmlns string is a reserved word in XML that indicates that a namespace is being created. Using the prefix bci guarantees uniqueness even if this document is combined with another. Note that the use of the URL in defining the namespace is a matter of convenience. Since the URL is properly registered it can serve as a unique distinguishing element.

2.1.2 XML Schema

2.1.2.1 An XML document follows a set of rules that may be explicitly defined in an XML Schema Definition (XSD) file. The following is an example XSD file. (Note that the prefix xs: is often used rather than xsd:)

```
<? Xml version='1.0' encoding='utf-8' ?>

<xsd:schema  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                    xmlns:staff="http://www.bcisse.com/"
                    targetNamespace=" http://www.bcisse.com/"

<xsd:element name="staffList" type="bci:StaffListType"/>

<xsd:complexType name="StaffListType">
    <xsd:sequence>
       <xsd:element name="employee"  type="bciEmployeeType" minOccurs="1"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="EmployeeType">
    <xsd:sequence>
       <xsd:element name="firstName"  type="xsd:string"/>
       <xsd:element name="lastName"  type="xsd:string"/>
    </xsd:sequence>
     <xsd:attribute name="empID" type="xsd:positiveInteger"/>
</xsd:complexType>

</xsd:schema>
```

An XSD file provides the basic mechanism for defining the structure, content, and data types for XML documents. XSD files are used to define data structures in web services standards; for example, they are used to define the structure of SOAP messages and WSDL service definitions. For web service applications they define application specific data elements and thus are referenced in the application's WSDL documents and can be used to validate the data elements sent in SOAP messages.

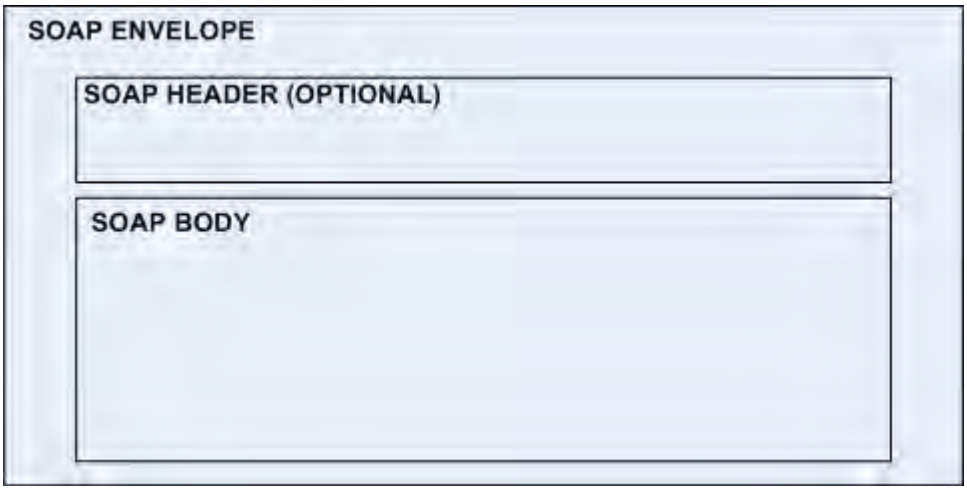
2.1.3 SOAP Messaging

2.1.3.1 SOAP, originally defined as the Simple Object Access Protocol, is a protocol for exchanging XML information in the implementation of web services. The following depicts the general structure of a SOAP message, first as a block diagram and then in XML.

The following is an example of a SOAP request/response exchange pattern from the SOAP 1.1 specification.

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Body>
      <m:GetLastTradePrice xmlns:m="Some-URI">
         <symbol>DIS</symbol>
      </m:GetLastTradePrice>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
   <SOAP-ENV:Body>
      <m:GetLastTradePriceResponse xmlns:m="Some-URI">
         <Price>34.5</Price>
      </m:GetLastTradePriceResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```




```
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">

  <SOAP-ENV:Header> <!--  optional -->
        <!--  Header blocks go here  . . . -->
  </SOAP-ENV:Header>

  <SOAP-ENV:Body>
        <!--  Payload or Fault element goes here  . . . -->
  </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

2.1.3.2 SOAP messages are sent following different message exchange patterns. The particular exchange pattern is specified in the service definition (see section 2.1.4). The exchange patterns are:

- One-way operation – A one-way operation is an operation in which the service endpoint receives a message but does not send a response;

- Request/response operation – A request/response operation is an operation in which the service endpoint receives a message and returns a message in response;

- Notification operation – A notification operation is an operation in which the service endpoint sends a message to a client, but it does not expect to receive a response; and

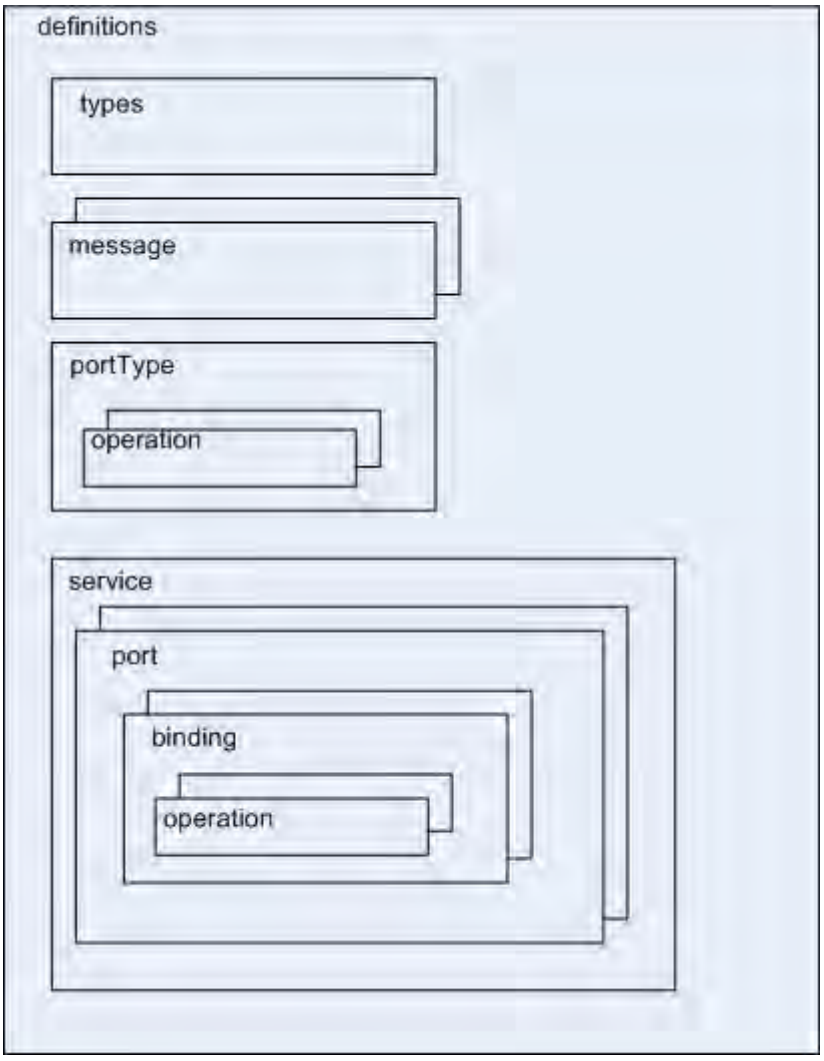
- Solicit/response operation – A solicit/response message is an operation in which the service endpoint sends a message and expects to receive a message in response.

2.1.3.3 Binary data may be sent within a SOAP message. This may be accomplished by encoding the binary data as text using Base 64 encoding. However this is usually only done for small amounts of binary data since Base 64 encoding increases the size of the binary data by 33%. For larger amounts of binary data, the Message Transmission Optimization Mechanism (MTOM) may be used with XML-binary Optimized Packaging (XOP). Using MTOM/XOP binary data is sent in its original binary form, avoiding any increase in size due to encoding it in text. The MTOM specification conceptually defines a method for optimizing SOAP messages by separating out binary data and sending it in separate binary attachments using a MIME Multipart/Related message. The XOP specification defines an implementation for optimizing XML messages using binary attachments in a packaging format that includes but is not limited to MIME messages.

2.1.4 WSDL

2.1.4.1 A web service is made available to clients using the Web Services Description Language (WSDL). A client may retrieve the service definition from a Registry or directly from the URL where the service resides. The retrieved WSDL document contains the following elements:

- <wsdl:types>
- <wsdl:message>
- <wsdl:operation>
- <wsdl:portType>
- <wsdl:service>
- <wsdl:port>
- <wsdl:binding>



The <wsdl:types> element indicates that a WSDL type is being declared. Since according to the SOAP specification only one input and one output is permitted in SOAP messages, it may be necessary to declare a complex type composed of other primitive data types.

The <wsdl:message> element is an abstract element defining the messages exchanged. In the request/response message exchange pattern, there will be an input message, and output message, and optionally a fault message declared.

The <wsdl:operation> element defines the web service operations. It is analogous to a subroutine or method call in an application. The difference is that the operation is performed across the underlying network.

The <wsdl:portType> element is so named because in web services a port is a single web service. The portType is a container for all operartions that one web service can accept.

The <wsdl:binding> element provides a link back to the abstract elements in the WSDL document. It is a container for physical aspects of the service such as the encoding style.

The <wsdl:port> element contains the actual location of the service.

The <wsdl:service> element is a container for all ports that are represented by a WSDL document.

The <wsdl:definitions> element is the root element in a WSDL document.

The following is an example WSDL document is from the WSDL 1.1 specification.

```
<?xml version="1.0"?>
<definitions name="StockQuote"

targetNamespace="http://example.com/stockquote.wsdl"
        xmlns:tns="http://example.com/stockquote.wsdl"
        xmlns:xsd1="http://example.com/stockquote.xsd"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns="http://schemas.xmlsoap.org/wsdl/">

   <types>
     <schema targetNamespace="http://example.com/stockquote.xsd"
          xmlns="http://www.w3.org/2000/10/XMLSchema">
       <element name="TradePriceRequest">
         <complexType>
            <all>
               <element name="tickerSymbol" type="string"/>
            </all>
         </complexType>
       </element>
       <element name="TradePrice">
         <complexType>
            <all>
               <element name="price" type="float"/>
            </all>
         </complexType>
       </element>
     </schema>
   </types>

   <message name="GetLastTradePriceInput">
      <part name="body" element="xsd1:TradePriceRequest"/>
   </message>

   <message name="GetLastTradePriceOutput">
      <part name="body" element="xsd1:TradePrice"/>
   </message>

   <portType name="StockQuotePortType">
      <operation name="GetLastTradePrice">
        <input message="tns:GetLastTradePriceInput"/>
        <output message="tns:GetLastTradePriceOutput"/>
      </operation>
   </portType>

   <binding name="StockQuoteSoapBinding"type="tns:StockQuotePortType">
      <soap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http"/>
```

```
    <operation name="GetLastTradePrice">
      <soap:operation
       soapAction="http://example.com/GetLastTradePrice"/>
        <input>
          <soap:body use="literal"/>
        </input>
        <output>
          <soap:body use="literal"/>
        </output>
    </operation>
  </binding>

  <service name="StockQuoteService">
    <documentation>My first service</documentation>
    <port name="StockQuotePort" binding="tns:StockQuoteBinding">
      <soap:address location="http://example.com/stockquote"/>
    </port>
  </service>

</definitions>
```

2. 2 The Weather Data Model

2.2.1 The Weather Data Model is actually three data models: a top level conceptual model, a logical or structural model, and a physical exchange model. (See the WXXM Primer which may be retrieved from https://wiki.ucar.edu/display/NNEWD/WXXM) The logical model called WXXM: the Weather Exchange Model is generally used to refer to all three models.

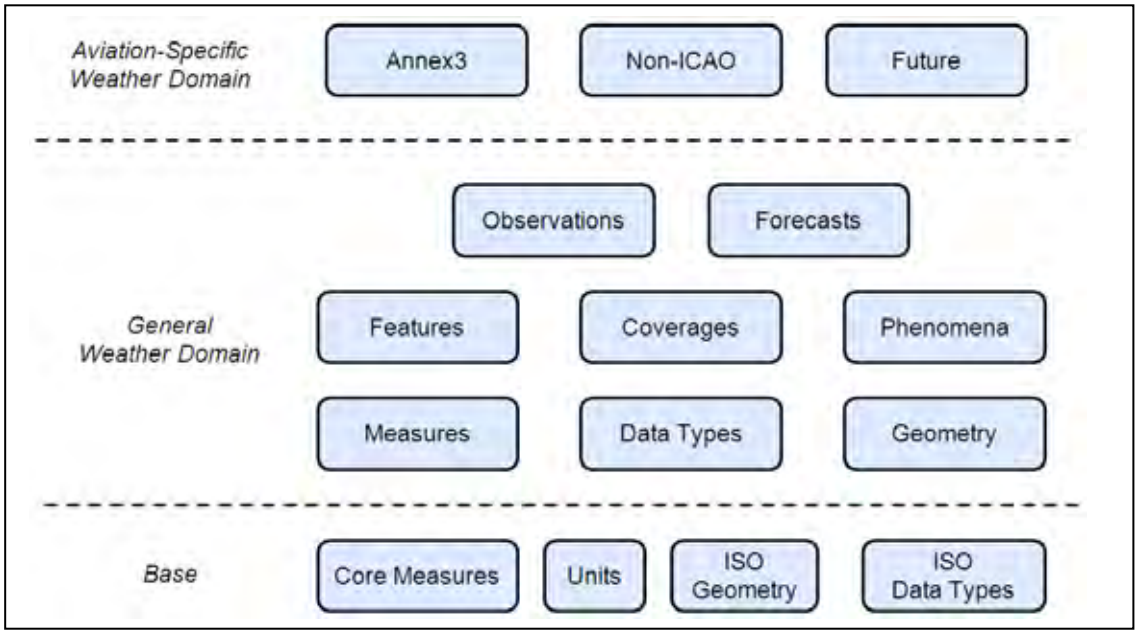
1. The **Weather Conceptual Model (WXCM)** provides a high-level, implementation-independent look at how weather data concepts are connected.

2. The **Weather Exchange Model (WXXM)** provides a logical and structural view of the same data and describes the interrelationships of every weather data concept.

3. The **Weather Exchange Schema (WXXS)** is an XML-formatted implementation of the exchange model.

2.2.2 The Weather Conceptual Model (WXCM) is depicted in the following figure. The conceptual model is defided heirarchically into three layers. The conceptual model is defined using the ISO 19000 series of specifications which define a conceptual schema language and encodings for geographic information.

The Base layer contains primitive types of geographic information. The General Weather Domain level contains a set of packages that describe general weather data concepts. The Aviation-specific Weather Domain level contains packages specific to the aviation community. This level is split into two main packages, a generic aviation package, and the more specific ICAO package. This layer also provides for future aviation packages.



2.2.3 The Weather Exchange Model (WXXM) a logical data model that is based on the high-level WXCM packages. This model further defines elements and their relationships with consideration for data exchange. WXXM is described using UML and provides a taxonomy of the relationships between weather concepts.

2.2.4 The Weather Exchange Schema (WXXS) is a machine-generated implementation of the Weather Exchange Model. The Weather Exchange Schema is a set of XSD files and thus represents an implementation-specific version of the Weather Exchange Model. The WXXS schema documents may be imported into WSDL documents in web service applications. The result is that aviation elements can be exchanged in the body of SOAP messages. The following is an example of a METAR report generated using WXXS schema documents.

```
<avwx:METAR
xmlns:avwx="http://www.eurocontrol.int/avwx/1.1"
xmlns:wx="http://www.eurocontrol.int/wx/1.1"
xmlns:wxont="http://wmo.int/ontologies/wx.owl#"
xmlns:om="http://www.opengis.net/om/1.0/gml32"
xmlns:gml="http://www.opengis.net/gml/3.2"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
gml:id="id0">

<avwx:rawText>
METAR KTTN 051853Z 04011KT 1/2SM VCTS SN FZFG BKN003 OVC010 M02/M02
  A3006 RMK AO2 TSB40 SLP176 P0002 T10171017=
</avwx:rawText>

<!--
Aerodrome weather observation.
-->
```

```
 <avwx:aerodromeWxObservation>
  <wx:Observation gml:id="id6">

    <om:samplingTime>
     <gml:TimeInstant gml:id="id8">
       <gml:timePosition>2008-11-04T12:00:00Z</gml:timePosition>
     </gml:TimeInstant>
    </om:samplingTime>

    <om:procedure xlink:href="urn:fdc:faa.gov:Sensor:WeatherStation:01234"/>

    <!-- Observed property links to higher-level Ontology  concept that
        corresponds to result type
    -->
    <om:observedProperty
xlink:href="http://www.eurocontrol.int/ont/avwx/1.1/wx.owl#AerodromeWx"/>

    <!-- Feature of interest links to Aerodrome feature within this
        METAR instance.
    -->
    <om:featureOfInterest xlink:href="#id2"/>

    <!-- Result contains weather properties relevant to Aerodrome area of
        interest
    -->
    <om:result>
     <avwx:AerodromeWx gml:id="id10">
       <avwx:airPressure uom="mBar">900</avwx:airPressure>
       <avwx:airTemperature uom="C">30</avwx:airTemperature>
       <avwx:dewpointTemperature uom="C">20</avwx:dewpointTemperature>
       <avwx:verticalVisibility uom="NM">2</avwx:verticalVisibility>
       <avwx:windDirection uom="deg">30</avwx:windDirection>
       <avwx:horizontalVisibility gml:id="hv1">
         <avwx:minimumVisibility uom="NM">5</avwx:minimumVisibility>
         <avwx:directionMinimum>NW</avwx:directionMinimum>
       </avwx:horizontalVisibility>
       <avwx:windSpeed uom="kt">15</avwx:windSpeed>
       <avwx:qnh uom="mBar">900</avwx:qnh>
       <avwx:qfe uom="mBar">900</avwx:qfe>
       <avwx:cloudCondition gml:id="cc1">
          <wx:base uom="ft">2000</wx:base>
          <wx:cloudType>CUMULUS</wx:cloudType>
       </avwx:cloudCondition>
       <avwx:cloudCondition gml:id="cc2">
         <wx:base uom="ft">15000</wx:base>
          <wx:cloudType>CIRRUS</wx:cloudType>
       </avwx:cloudCondition>
       <avwx:seaWx>
         <avwx:SeaWx gml:id="id18">
          <avwx:surfaceTemperature uom="C">20</avwx:surfaceTemperature>
          <avwx:seaState>CALM RIPPLED</avwx:seaState>
         </avwx:SeaWx>
       </avwx:seaWx>
     </avwx:AerodromeWx>
    </om:result>

  </wx:Observation>
```

```
</avwx:aerodromeWxObservation>

<!--
  Aerodrome is specified outside the context of individual observations
  and forecasts within METAR report and referenced within each
  observation/forecast to reduce redundancy in report instances.
-->
<avwx:appliesTo>
  <avwx:Aerodrome gml:id="id2">
    <gml:identifier  codeSpace="urn:icao:code:Aerodrome:">DEN</gml:identifier>
    <gml:name>BOS</gml:name>
    <gml:location>
      <gml:Point  srsName="urn:ogc:crs:EPSG:4979"  srsDimension="3" gml:id="id4">
        <gml:pos>40.0 -70.0 1000.0</gml:pos>
      </gml:Point>
    </gml:location>
  </avwx:Aerodrome>
</avwx:appliesTo>

<avwx:stationId codeSpace="urn:icao:code:weatherStation:">KDEN</avwx:stationId>
<avwx:automated>true</avwx:automated>
<avwx:missing>false</avwx:missing>

</avwx:METAR>
```

2. 3 SOAP Messaging over AMHS

2.3.1 SOAP messages are typically sent over HTTP; however, other bindings are possible. The 1.2 version of the SOAP specification defines a SOAP Protocol Binding Framework. The formal set of rules for carrying a SOAP message within or on top of another protocol (underlying protocol) for the purpose of exchange is called a binding. The SOAP Protocol Binding Framework provides general rules for the specification of protocol bindings. The framework also describes the relationship between bindings and SOAP nodes that implement those bindings. Bindings other than HTTP can be created by specifications that conform to the binding framework introduced in Chapter 4 of SOAP 1.2.

2.3.2 It would be possible to define a binding to to carry SOAP messages over AMHS (i.e., over X.400). Bindings for SOAP over SMTP have already been defined and implemented. However there are several considerations to meet the general criteria of Chapter 4 of SOAP 1.2.

First of all, the assumed end-to-end architecture for carrying SOAP over AMHS should be defined. For example the assumed operating model could be IPv4 packet networks using SOAP over HTTP connected to the AMHS messaging network using SOAP over X.400.

The basic method of encapsulating SOAP messages needs to be developed (assuming that a translating gateway approach would not be followed.) This needs to be developed for SOAP messages that are all XML as well as SOAP messages with binary attachments using MTOM/XOP.

In addition conventions for and/or restrictions to end-to-end message exchange patterns to be supported need to be defined. X.400 like SMTP is essentially a one-way asynchronous transmission media. Whether to and how to interact with the normal request/response exchange pattern which is supported naturally in HTTP needs to be defined also.

3. Action Required by the Meeting

The meeting is invited to:

a) Note the background information on web services

b) Note the background information on WXXM

c) Note the concept of SOAP messages carrying Met data using WXXM standards over AMHS is different from previous discussions about “XML over AFTN or over AMHS”.

d) Assess if and when to proceed in defining a SOAP over X.400 transport binding.

– – – – – – – – – – –