Open Geospatial Consortium, Inc.

Date: 2008-01-16

Reference number of this document: OGC 08-009r108-009r1

Version: 0.1.0

Category: Discussion Paper

Editor: Bastian Schäffer

OWS 5 SOAP/WSDL Common Engineering Report

Warning

This document is not an OGC Standard. This document presents a discussion of technology issues considered in an initiative of the OGC Interoperability Program. This document does not represent an official position of the OGC. It is subject to change without notice and may not be referred to as an OGC Standard. However, the discussions in this document could very well lead to the definition of an OGC Standard. Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Preface

This document presents the results of the OWS 5 GPW-SOAP/WSDL thread. This group focused on creating general recommendations and guidelines for WSDL/SOAP support to existing and future OGC Web Services.

Contents

Page

Figures

Page

Tables

Page

Listings

Page

OWS 5 SOAP/WSDL Common Engineering Report

1 Introduction

1.1 Scope

This OGC document reports the results achieved in the OWS5 GPW-SOAP/WSDL thread which is focused on creating general recommendations and guidelines for WSDL/SOAP support to existing OGC Web Services.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The OGC shall not be held responsible for identifying any or all such patent rights.

1.2 Document contributor contact points

All questions regarding this document should be directed to the editor or the contributors:

Name	Organization
Bastian Schäffer	IfGI
Rüdiger Gartmann	IfGI/ConTerra
Alessandro Triglia	OSS Nokalva, Inc
Farrukh Najmi	Wellfleet software
David Rosinger	Intergraph

1.3 Revision history

Date	Release	Editor	Primary clauses modified	Description
2007-09-10	0.0.1	Bastian Schäffer	Initial Document	Initial Document
2007-09-13	0.0.1	Bastian Schäffer	5	Paragraph 6 added
2007-09-13	0.0.1	Rüdiger Gartmann	7	Paragraph 8 added
2007-09-15	0.0.1	Farrukh Najmi & Bastian Schaeffer	6.1	Paragraph added
2007-09-15	0.0.1	Farrukh Najmi	6.2	Paragraph added
2007-09-15	0.0.1	Alessandro Triglia & Farrukh	6.3	Paragraph added

		Najmi		
2007-09-15	0.0.1	Alessandro Triglia	6.4	Paragraph added
2007-09-15	0.0.1	Alessandro Triglia	6.5	Paragraph added

1.4 Future work

Future work may include but is not limited to:

- Describe Fault handling
- Give advice on specific services

2 References

The following documents are referenced in this document. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. For undated references, the latest edition of the normative document referred to applies.

http://www.w3.org/TR/wsdl, *Web Services Description Language (WSDL) 1.1*.

http://www.w3.org/TR/soap, *SOAP Version 1.2*.

3 Terms and definitions

4 Conventions

4.1 Abbreviated terms

ISO	International Organization for Standardization
KVP	Key-Value Pair
OGC	Open Geospatial Consortium
UDDI	Universal Description, Discovery and Integration
WCS	Web Coverage Service

WFS	Web Feature Service
WMS	Web Map Service
WPS	Web Processing Service
WSDL	Wed Service Description Language
XML	eXtended Markup Language

5 ER Topic overview

This document reports the activity of the OWS 5 GPW-SOAP/WSDL thread. This group has focused on creating general recommendations and guidelines for WSDL/SOAP support to existing and future OGC Web Services.

Besides brief introductions to basic technologies such as SOAP, WSDL and related aspects, basic guidelines discussed during this testbed are listed in this document. In the course of this testbed, the recommendations have to be validated on the basis of actual implementations.

6 WSDL

6.1 Introduction

The OWS 1 testbed identified the need for an Interface Definition Language to describe OGC services and WSDL was chosen as an instance of such a language [Atkinson & Martel, 2002]. The Web Service Description Language (WSDL) is characterized briefly WSDL in version 1.1 in the following sections.

6.1.1 The anatomy of a WSDL file

The Web Service Description Language (WSDL) is an XML based language designed to describe Web Services [W3C, 2001]. This section characterizes briefly the WSDL standard in version 1.1. WSDL describes a Web Service as a set of interfaces, which interact via operations with a requestor. The operations are based on defined addresses and communication protocols. A WSDL specification is divided into two major parts. The abstract part describes the interface, its operations and input and output messages in a transport and wire-independent manner. The second part is the concrete part of the description, where bindings denote the transport and wire formats for interfaces. A service endpoint associates the network address with a binding. Finally, a service clusters the endpoints that implement a common interface. Figure 1 shows the conceptual WSDL component model.

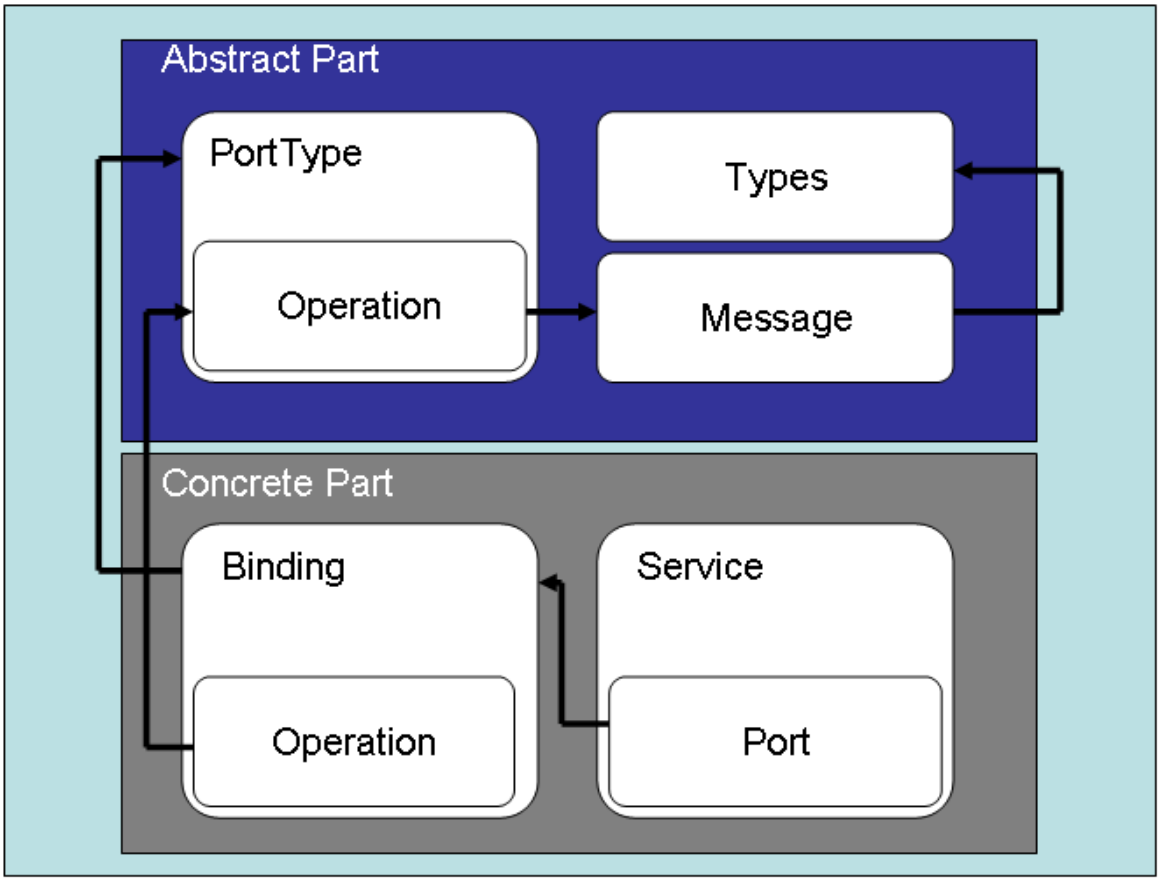

Figure 1: Conceptual WSDL design

To achieve a full service description, WSDL makes use of the following language elements which are briefly described in the following:
The `<definitions>` element is the root element of every WSDL document. The following elements are all child elements of this root. Datatypes are specified by means of the `<types>` elements, which are used for the message exchange. WSDL is not limited to a specific type system. Thus, any XML schema can be imported.
The `<message>` elements describe the input messages received and output messages send. Each message consists of `<part>` elements, which are typed by the `<type>` elements. This approach applies an early binding pattern, since all message types have to be known at design time.
The supported interface of a Web Service is defined by a `<portType>` element. `<PortType>` elements combine a set of operations and corresponding messages involved, independent from the actual implementation. The WSDL specification supports four different types of message exchange patterns:

1) Request/Response: The Web Service receives a message, and sends a correlated response message

2) One-Way: The Web Service receives a message but does not respond.

3) Notification: The Web Service sends a message

4) Solicited-Request: The Web Service sends a message to a client, and the client answers with a correlated response message

The `<binding>` elements specify the transport protocol details and data formats for the operations and messages described by a particular `<portType>`-element. WSDL defines default bindings for SOAP, HTTP-GET/POST and MIME and allows KVP and XML requests (see section 5.2.5). `<Port>` elements define an address for a binding. Clients can interact with the operations offered in the `<portType>` element via this address and the transport protocols and formats defined in the `<binding>` element. Finally, the `<service>` element groups a set of related ports together.

6.1.2 WSDL versions

WSDL 1.1 is the version chosen for this document, since it is most used and supported by existing tools. Especially BPEL4WS [Andrews et al., 2003], as the BPEL standard used in this testbed, references WSDL 1.1. Furthermore, WSDL is recommended by WS-I Basic Profile 1.2, which shall be used in conjunction with WSDL 1.1 [WS-I, 2007]. WSDL 2.0 (formerly 1.2 by renamed because of its substantial differences from 1.1) is at the moment in draft [W3C, 2007] and not widely supported yet.

6.2 WSDL Guidelines

6.2.1 How to publish a WSDL file for a Web Service

There are many ways to publish a WSDL file for a Web Service. The mainstream IT world has established three major ways:

- The WSDL file can be offered on the web site of the organization that publishes the web service. This approach allows humans to find the WSDL file but is not sufficient for automatic use.

- The WSDL file can be published through public and private registries. UDDI would be the choice for general Web Services and CSW for OGC Web Services. This approach follows the publish-find-bind pattern and thus allows humans and services to discover the WSDL file in a standardized manner.

- The web service itself can also publish the WSDL file. AXIS and the .NET frameworks follow the convention of http://url:port/service/xx?WSDL. This is sufficient for a pragmatic approach, but fails for multiple WSDL files describing specific aspects of a Web Service.

Keeping the three approaches in mind, option number two is the recommended way to publish a WSDL file for a Web Service, since it allows humans and machines to discover WSDL files in a standardized way applying the publish find-bind-pattern.

6.2.2 WSDL vs. GetCapabilities

WSDL is a common way to describe Web Service interfaces [W3C, 2001]. The OGC GetCapababilities operation is created partially with the same intend. However, WSDL focuses only on the explicit interface description by listing all offered operations and their input and output messages including their payload types. The GetCapabilities operation response also lists all operations, but does not specify the actual messages and their types. On the other hand, meta information like bounding boxes or available layers are describe by GetCapabilities. This leads to the conclusion, that a WSDL cannot replace the GetCapabilities operation in any sense, since both approaches have an intersection but are not congruent.

From a SOAP based SOA perspective, the WSDL is always the entry point to discover the Web Service. While from an OGC point of view, the GetCapabilities operation is the common entry point. To overcome these complementary approaches, the GetCapabilites response should list a path to the WSDL files describing the OGC Web Service. Therefore, a <WSDL> element as a child of the root <Capabilities> element shall contain a reference the WSDL file as presented in listing 1.

```
<Capabilities>
...
   <WSDL xlink:href="http://foo.bar/xx?WSDL" />
</Capabilities>
```

Listing 1: WSDL description in a getCapabilities response

This approach enables OGC Web Services with SOAP bindings to be discovered via the GetCapabilities operation and to get additional information through the referenced WSDL file. On the other side a Web Service can also be discovered via an initial WSDL file and additional meta data can be obtained through the GetCapabilities operation described in the WSDL.

6.2.3 How many WSDL files should be offered by a Web Service

IONIC provided a set of WSDL files[1] for this testbed. These WSDL files have already proven to be working and were used as the basis for further developments. Following IONIC's approach, only one top-level WSDL file shall be created and published. This top-level WSDL file can import a set of WSDL files for specific parts, for instance a WSDL for the abstract part and a WSDL describing only the concrete part of the WSDL (see section 5.1.1-The anatomy of a WSDL file). Listing 2 shows an example.

[1] http://portal.opengeospatial.org/index.php?m=projects&a=view&project_id=241&tab=2&artifact_id=23098

```
<wsdl:import namespace="http://www.opengis.net/wfs/responses"
             location="http://foo.bar/wfs-responses.wsdl"/>
```

Listing 2: Sample import of another WSDL

The result will be a directed acyclic graph of referencing WSDL files. Each node will be a part of a WSDL file and each arc a WSDL import. However, only the root element (WSDL) shall be visible from outside. Figure 2 presents this approach.

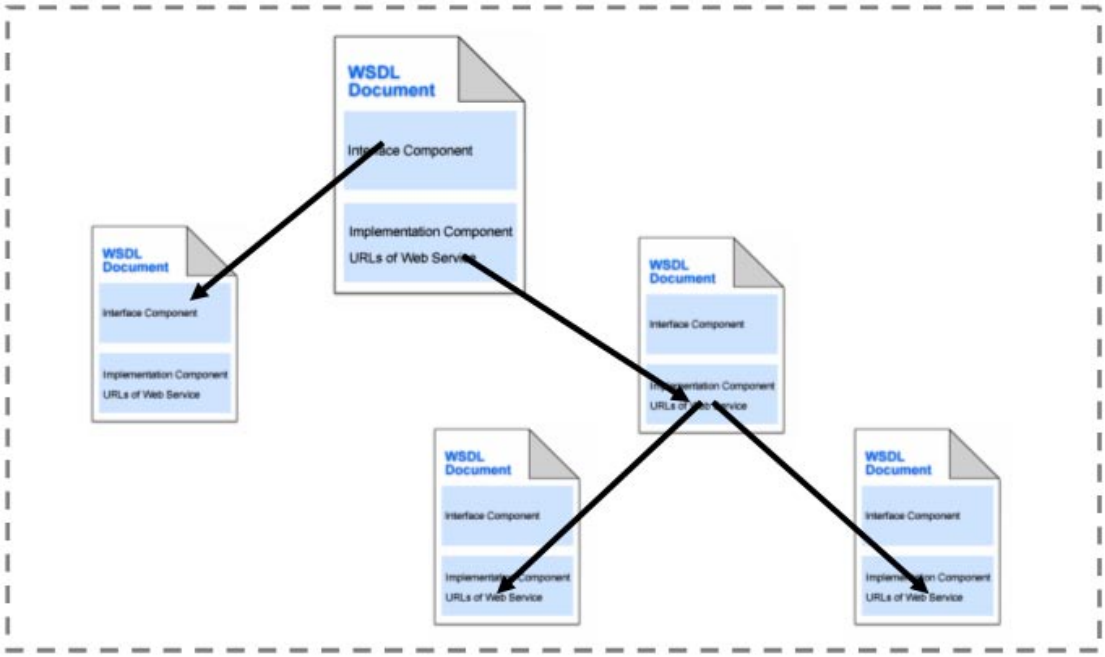

Figure 2: Conceptual WSDL composition

This approach allows the modularization of a WSDL file and therefore fosters reusability and cost effective maintenance and is especially a useful way to deal with application schemas and data views.

6.2.4 How to describe late binding operations with WSDL

As described in Section 5.1.1, WSDL applies an early binding approach. Each message payload is required to have a well-defined type at design time. This enables clients to create stubs on the fly based on the contract between the client and the server described by WSDL. In the OGC world, some services have operations, which apply a late binding approach. The actual type of a (response) message depends on the request and therefore can only be determined at runtime. As said before, some services with late binding operation already exist (e.g. WFS getFeature, WPS execute) and it can be thought of others for the future.

To overcome these complementary approaches, only general guidelines can be provided. For every service and use case, a specific solution has to be determined. In general, there are three basic approaches:

1) A WSDL template can be provided, which makes use of `xsd:anyType` as the underlying payload type. This approach would lead to a generic WSDL with a weak contract between client and server, but might be sufficient in a chaining scenario for a BPEL engine.

> 1.1) If an application schema exists, the `xsd:anyType` types can be substituted with the corresponding parts from the application schema
>
> 1.2) If a large application schema exists and only a subset is used, the `xsd:anyType` types can be substituted by a partial application schema. This prevents clients to build a large stack of unused stub objects and would increase performance. Furthermore, a partial application schema can be regarded as a dataview and thus might help to describe dataviews with WSDL.

2) Split the late binding operation in a number of early binding operations and create for each early binding operation a WSDL description. This allows the early binding operation to have concrete schemas for its message payloads, since the return schema is specific for this operation. Furthermore, the GetCapabilities must list all WSDLs and indicate how to fetch them. One drawback of this approach is that it is only applicable to a very small set of operations.

Example:

Problem: A specific WPS instance offers only two processes (in principle it can have n processes). In this case *Buffer* and *Intersect*. The *Buffer* process returns GML 2.1 and the *Intersect* process returns KML 2.1. Both processes can be executed through the same operation: *execute*. Depending on the execute request and the given process Identifier parameter either the *Buffer* process is executed and GML is returned or the *Intersect* process is executed and KML is returned. But which process will be executed and therefore what kind of payload XML schema will be returned can only be determined at runtime.

Solution: Create a WSDL file WPS_Buffer.wsdl, which has the usual set of WPS operations, but specifies a concrete output schema for the execute operation (GML 2.1). Create a WPS_Intersect.wsdl analogous to the WPS_Buffer.wsdl with a KML response schema.

3) Split the late binding operation in a number of early binding operations. In other words, for each potential payload schema, an operation is created which has a `_schemaName` suffix. This is only applicable to a small set of operations.

Example:

Problem: A specific WPS instance offers only two processes (in principle it can have n processes). In this case *Buffer* and *Intersect*. The *Buffer* process returns GML 2.1 and the *Intersect* process returns KML 2.1. Both processes can be executed through the same operation: execute. Depending on the execute request and the given process Identifier parameter either the *Buffer* process is executed and GML is returned or the *Intersect* process is executed and KML is returned. But which process will be executed and therefore what kind of payload XML schema will be returned can only be determined at runtime.

Solution: Create a WSDL file, which has the usual set of WPS operations plus two execute operation: execute_buffer with a concrete GML output schema. And execute_intersect with a KML output schema.

6.2.5 Request, Response and Binding Permutations

This IPR focuses on SOAP bindings for OGC Web Services. According to the TC resolution from June 2006, SOAP should be only optional to existing HTTP-GET and HTTP-POST bindings. The HTTP 1.1 specification [W3C, 1999] defines: “The GET method means retrieve whatever information (in the form of an entity) is identified by the Request-URI.” In the OGC world, the Request-URI looks like:

```
http://<server_address>/<service_path>?<parameter_list>
```

The `<parameter_list>` is also called query string and contains a list of key-value pairs (KVPs) in the form of `<paramter_name>=<value>`. Each KVP should be separated by the („&“) sign. The URL length is in principle no bounded, but should not be exceeded by 255 Byte. The response of an operation invoked with HTTP-GET is not limited [W3C, 1999]. Thus the response can be also XML documents or binary data as the two common OGC response types.

HTTP-POST is described by the HTTP 1.1 specification as “The POST method is used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line.” In the OGC world, an XML encoded request is send via HTTP-POST. The response is analogous to the HTTP-GET method.

As described in section 6.2, the SOAP body can contain any XML document like the HTTP-POST request. Chapter 7 describes a way to wrap KVP request in SOAP.

These facts lead to the following matrix:

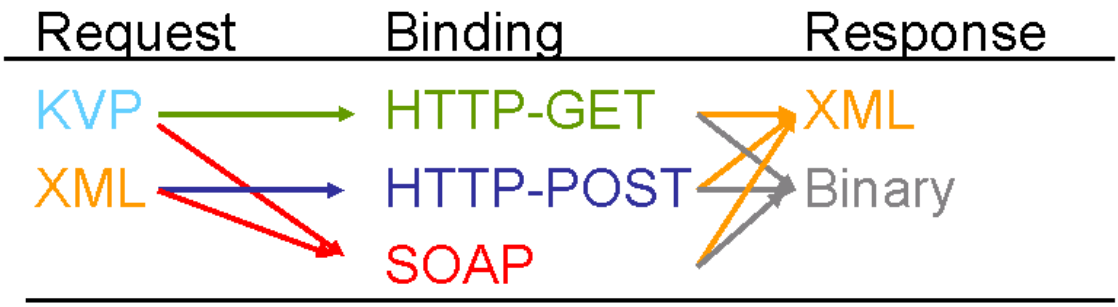

Figure 3: Request-Binding-Response permutations

However, even if these twelve options are possible to be described in WSDL, there are numerous obstacles.

OWS-5 View on WSDL and KVP requests and non-SOAP bindings

The WSDL 1.1 specification describes HTTP-GET/POST bindings. However, the TC from September 2006 asked for SOAP bindings in WSDL only. To avoid the description of KVP in WSDL and complications with the WS-I Basic Profile, the description of HTTP-GET/POST bindings should remain in the GetCapabilities.

6.2.6 How to describe an operation with a XML request, an XML response and a HTTP-SOAP binding

As discussed in section 7.6.3, the SOAP document/literal style shall be used for XML requests with SOAP.

According to the WSDL 1.1 specification, the recommended way to invoke a service via SOAP document/literal is to create a single part message as can be seen in exemplarily listing 3.

```
<message name="GetCapaMessage_POST">
    <part name="request" element="wms:GetCapabilities"/>
</message>
```

Listing 3: Sample XML request message

The XML response message shall simply reference the root element of the schema of the XML returned. An example is presented below:

```
<message name="GetCapaResult">
    <part name="response" element="wms:WMT_MS_Capabilities"/>
</message>
```

Listing 4: Sample XML response message

The operation should be described in the abstract WSDL part by simply using the previously described input and output messages as seen in listing 5:

```
<operation name="GetCapabilities">
    <input message="tns:GetCapaMessage_POST "/>
    <output message="tns:GetCapaResult"/>
</operation>
```

Listing 5: Sample operation using a XML request and response message

In the concrete WSDL part, the binding element shall reference the portType from the abstract part and make use of the <soap:binding style="document"> binding as described in the WSDL 1.1 specification. The operation element shall reference the corresponding operation from the abstract part and use <soap:body use="literal"/> as input and output. An example is presented in listing 6:

```
<soap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="GetCapabilities ">
        <soap:operation soapAction="http://www.opengis.net/wms/requests#
        GetCapabilities"/>
    <input>
        <soap:body use="literal"/>
    </input>
  <output>
        <soap:body use="literal"/>
    </output>
</operation>
```

Listing 6: Sample SOAP binding for an operation with an XML request and response message

6.2.7 How to describe an operation with a XML request, a binary response and a HTTP-SOAP binding,

As discussed in section 7.6.3, the SOAP document/literal style shall be used for XML requests with SOAP.

According to the WSDL 1.1 specification, the recommended way to invoke a service via SOAP document/literal is to create a single part message as can be seen in exemplarily listing 7.

```
<message name="GetCapaMessage_POST">
    <part name="request" element="wms:GetCapabilities"/>
</message>
```

Listing 7: Sample XML request message

Since the response is plain binary and does not have a schema to reference, the response message shall be a single part message using the an element described in the <types> section and using type="xs:base64Binary". An example is presented in listing 7:

```
<types>
   <schema…>
     …
      <element name="binaryPayload" type="xs:base64Binary"/>
     </schema>
</types>

<message name="GetMapResult">
     <part name="response1" element="tns:binaryPayload"/>
</message>
```

Listing 8: Sample plain binary response message

The operation should be described in the abstract WSDL part by simply using the previously described input and output messages as seen in figure 8:

```
<operation name="GetMap">
     <input message="tns:GetMapMessage_POST "/>
     <output message="tns:GetMapResult"/>
</operation>
```

Listing 9: Sample operation using a XML request message and a plain binary response message

In the concrete WSDL part, the binding element shall reference the <portType> from the abstract part and make use of the <soap:binding style="document"> binding as described in the WSDL 1.1 specification. The operation element shall reference the corresponding operation from the abstract part and use <soap:body use="literal"/> as input and output. An example is presented in listing 9:

```
<soap:binding style="document"
          transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="GetCapabilities ">
     <soap:operation soapAction="http://www.opengis.net/wms/requests#
     GetCapabilities"/>
   <input>
     <soap:body use="literal"/>
   </input>
   <output>
     <soap:body use="literal"/>
   </output>
</operation>
```

Listing 10: Sample SOAP Binding for an operation with a XML request message and a binary response message

6.2.8 How to address security and licensing preconditions

WSDL as an integral part of the Web Service operational model, is sufficient for describing Web Service interfaces, but fails for defining access constraints and preconditions. Web Services Policy Framework (WS-Policy) is a standard to overcome these limitations [W3C, 2006a]. WS-Policy defines abstract XML policy elements which consist of a set of policy alternatives and are based on a set of policy assertions. A policy assertion can describe for instance required security tokens, supported encryption algorithms, and privacy rules. Other standards can extend these abstract policies with concrete polices. The Web Service Security Policy Language (WS-SecurityPolicy) specification allows the definition of concrete policies such as whether SOAP messages should be signed or encrypted and which algorithms should be used [OASIS, 2007].

In order to integrate WS-Policy with WSDL, Web Services Policy Attachment (WS-PolicyAttachment) [W3C, 2006a] can be used to include WS-Policy in WSDL. This enables requestors to obtain the Web Service interface description and potential access constraints, requirements and precondition in one single WSDL file. Listing 9 shows an example.

These policies can be defined on different levels to address specific needs. For example, a user is required to login before invoking an operation, but the response messages does not need to have authentication requirements. In this case, an input message policy subject has to be created for all inbound messages. Or if all operations are required to have the same policy requirement, the endpoint policy subject should be defined instead of duplicating the same operation policy subject for every operation in your endpoint.

As described in figure 4, each level is a Policy Subject. A Policy Subject is an entity (endpoint, message, operation, service) which can be associated with a policy. A Policy Scope is the union of Policy Subjects to which a Policy may apply. A Policy Attachment associates a policy with one or more Policy Scopes. An Effective Policy, for a given Policy Subject, is the combination of relevant policies. The relevant policies are those attached to Policy Scopes that contain the Policy Subject.

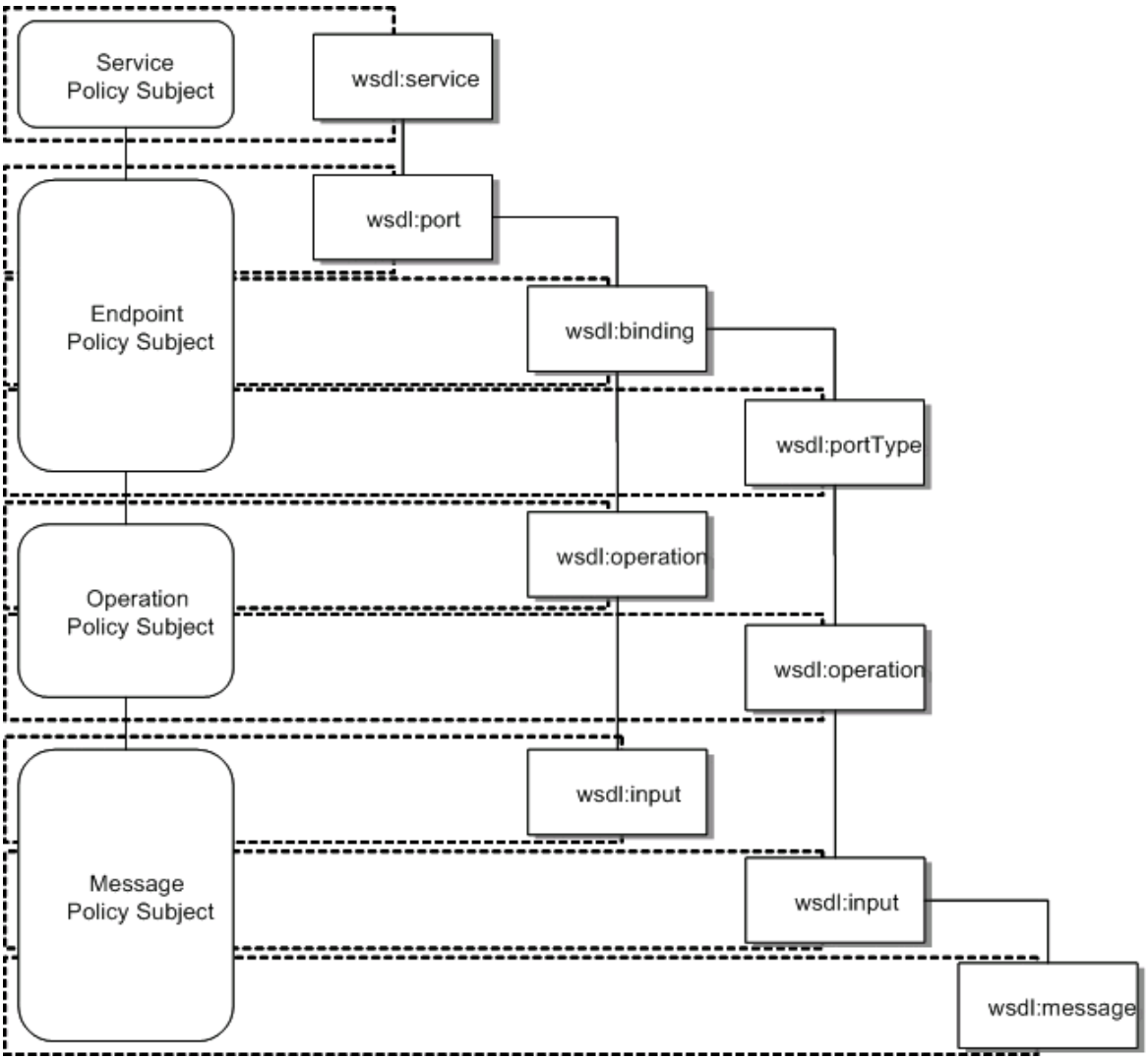

Figure 4: WS-Policy structure in relation to WSDL taken from [W3C, 2006b]

The policy defined in *wsdl:service* is considered as the Service Policy Subject. The Endpoint Policy Subject is an effective policy merged from *wsdl:port*, *wsdl:portType*, and *wsdl:binding*. The Operation Policy Subject is described by *wsdl:portType/wsdl:operation* and *wsdl:binding/wsdl:operation*. The following *WSDL1.1* elements are used to calculate the Message Policy Subject: *wsdl:message*, *wsdl:portType/wsdl:operation/wsdl:input*, *wsdl:portType/wsdl:operation/wsdl:output*, *wsdl:portType/wsdl:operation/wsdl:fault*, *wsdl:binding/wsdl:operation/wsdl:input*, *wsdl:binding/wsdl:operation/wsdl:output*, and *wsdl:binding/wsdl:operation/wsdl:fault.*

<wsdl:definitions name="StockQuote"
targetNamespace="http://www.fabrikam123.example.com/stock/binding"

```
xmlns:tns="http://www.fabrikam123.example.com/stock/binding"
xmlns:fab="http://www.fabrikam123.example.com/stock"
xmlns:rmp="http://schemas.xmlsoap.org/ws/2005/02/rm/policy"
xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsoap12="http://schemas.xmlsoap.org/wsdl/soap12/"
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd">
    <wsp:Policy wsu:Id="RmPolicy">
        <rmp:RMAssertion>
            <rmp:InactivityTimeout Milliseconds="600000"/>
            <rmp:BaseRetransmissionInterval Milliseconds="3000"/>
            <rmp:ExponentialBackoff/>
            <rmp:AcknowledgementInterval Milliseconds="200"/>
        </rmp:RMAssertion>
    </wsp:Policy>
    <wsp:Policy wsu:Id="X509EndpointPolicy">
        <sp:AsymmetricBinding>
            <wsp:Policy>
                <!-- Details omitted for readability -->
                <sp:IncludeTimestamp/>
                <sp:OnlySignEntireHeadersAndBody/>
            </wsp:Policy>
        </sp:AsymmetricBinding>
    </wsp:Policy>
    <wsp:Policy wsu:Id="SecureMessagePolicy">
        <sp:SignedParts>
            <sp:Body/>
        </sp:SignedParts>
        <sp:EncryptedParts>
            <sp:Body/>
        </sp:EncryptedParts>
    </wsp:Policy>
    <wsdl:import namespace="http://www.fabrikam123.example.com/stock"
location="http://www.fabrikam123.example.com/stock/stock.wsdl"/>
    <wsdl:binding name="StockQuoteSoapBinding" type="fab:Quote">
        <wsoap12:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
        <wsp:PolicyReference URI="#RmPolicy" wsdl:required="true"/>
        <wsp:PolicyReference URI="#X509EndpointPolicy" wsdl:required="true"/>
        <wsdl:operation name="GetLastTradePrice">
            <wsoap12:operation
soapAction="http://www.fabrikam123.example.com/stock/Quote/GetLastTradePriceReq
uest"/>
            <wsdl:input>
                <wsoap12:body use="literal"/>
```

```
                    <wsp:PolicyReference URI="#SecureMessagePolicy"
wsdl:required="true"/>
                </wsdl:input>
                <wsdl:output>
                    <wsoap12:body use="literal"/>
                    <wsp:PolicyReference URI="#SecureMessagePolicy"
wsdl:required="true"/>
                </wsdl:output>
            </wsdl:operation>
        </wsdl:binding>
</wsdl:definitions>
```

Listing 11: **WS-Policy and WSDL taken from taken from [W3C, 2006b]**

7 SOAP

7.1 Introduction

SOAP is a lightweight XML based protocol created for structured information exchange in a decentralized, distributed environment. It is part of the W3C definition of a Web Service as the underlying interaction protocol.

SOAP 1.1 and SOAP 1.1 with Attachments are earlier versions of SOAP that have been submitted to W3C as a note. Although these versions will not be approved as a W3C Recommendation, they do have numerous implementations. The SOAP 1.2 specification has been approved by W3C Recommendation and has many improvements since SOAP 1.1. SOAP 1.1 is based on XML 1.0 while SOAP 1.2 is based on XML Infoset. In SOAP 1.2 binding to an underlying protocol to specify the XML serialisation is left to binding specifications. SOAP 1.2 - Part 2 defines a binding to the HTTP protocol and uses XML 1.0 as the serialisation of the SOAP message infoset.

This document recommends OGC specifications to use the SOAP 1.2 specification for defining their SOAP bindings.

7.2 The anatomy of a SOAP message

A SOAP message is an XML based document that can be structured into an optional *Header* part and a mandatory *Body* part enclosed in an Envelope element. The *Header* part is intended to contain meta data, security tokens or other information not directly related to the actual request. The message payload is intended to be placed in the *Body* part. Listing 12 shows an empty message structure.

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope">
  <soap:Header>
  </soap:Header>
  <soap:Body>
  </soap:Body>
```

```
</soap:Envelope>
```

Listing 12: Empty SOAP message

7.3 Introduction to MTOM

MTOM is, strictly speaking, a W3C Recommendation (SOAP Message Transmission Optimization Mechanism) that specifies an efficient mechanism for the exchange of SOAP 1.2 messages carrying one or more large binary data blocks. The term MTOM is also used in a more general way to refer either to that mechanism or to an analogous mechanism applicable to SOAP 1.1.

MTOM completely relies on another W3C Recommendation, XOP (XML-binary Optimized Packaging), to provide a concrete wire format for the optimized SOAP messages containing a binary payload. Since MTOM uses XOP, its output is a SOAP message encoded as a MIME multipart/related entity which carries an XML document in its root part and a binary data block in each of its other parts. The XML document in the root part is the SOAP envelope transformed in accordance with XOP.

MTOM ensures that when two SOAP nodes exchange SOAP messages containing a binary payload, they will be able to do so (by using XOP) in an interoperable way.

MTOM is conceptually very different from SOAP with Attachments, even though the resulting wire formats are similar. The differences can be summarized as follows:

In SOAP with Attachments, the “SOAP message” is the XML document present in the root part of the MIME multipart entity. The body of the SOAP envelope contains explicit references to other parts of the MIME multipart entity, which may contain arbitrary data (each binary data block is regarded as “attached” to the SOAP message, not as part of it). The MIME multipart representation is defined as a binding of SOAP (on a par with the HTTP binding of SOAP).

In MTOM, the “SOAP message” is the entire MIME multipart entity, including both the root part (XML) and the binary parts. Conceptually, the SOAP message contains all the binary data blocks, each encoded in base64. This is also reflected in the XSD schema, which typically contains one or more element declarations of type xsd:base64Binary, one for each binary data block that is intended to be placed (**not** encoded in base64) in its own part of the MIME multipart entity. On the wire, though, there will be a MIME multipart entity whose root part contains special references to the other parts (in accordance with XOP) in place of the base64 strings.

The advantages of MTOM over SOAP with Attachments derive from the fact that in MTOM all the data items that are thought of as being part of a message (body or header) can be actually processed as an integral part of the SOAP message, whatever their type (XML or binary), because for most purposes (except for transmission purposes) they are

contained in the SOAP message. MTOM achieves this goal while still allowing efficient transmission of the data from one SOAP node to the next.

Much of what is specified in MTOM is conceptual. A naïve implementation of MTOM might do many conversions of binary data to and from base64 when creating, encoding, decoding, and processing SOAP messages, but a properly designed MTOM tool can avoid that overhead.

7.4 Introduction to XOP

XOP defines an alternate serialisation of XML infoset that uses a MIME multipart/related package, with an XML document as the root part. That root part closely resembles the normal XML serialisation of the document. The only difference is that any base64-encoded data is represented in a separate MIME part without base64 encoding and simply referenced from the serialized form of the original document.

The benefit of XOP is that it allows efficient serialization of XML Documents that contain binary data without incurring the penalty for base 64 encoding and while using the familiar Mime-Multi-part packaging mechanism.

The limitation of XOP is that it can only data within the XML infoset that is of type xs:base64binary. An important advantage of XOP is that because it operates at the XML infoset level, it can easily be applied to efficiently serializing binary attachments within HTTP GET bindings equally as well as SOAP bindings using a single consistent and standards-based mechanism.

This document recommends OGC specifications to use the XOP specification as the basis for defining attachments as binary content when defining HTTP binding for their interfaces.

7.5 Introduction to Fast Infoset

The Fast Infoset standard was jointly developed by ISO/IEC and ITU-T, and its official name is ITU-T Rec. X.891 | ISO/IEC 24824-1, "Information technology – Generic applications of ASN.1: Fast infoset". The standard can be downloaded for free from the ITU-T website.

Fast Infoset specifies a representation of an instance of the W3C XML Information Set using binary encodings.

The Fast Infoset technology provides an alternative to XML 1.0 syntax as a means of representing instances of the XML Information Set. This representation generally provides smaller encoding sizes and faster processing than an XML 1.0 representation.

Fast Infoset specifies the use of several techniques that minimize the size of the encodings (called "fast infoset documents") and that maximize the speed of creating and

processing such documents. These techniques include the use of dynamic tables (for both character strings and qualified names), initial vocabularies, and external vocabularies.

Fast Infoset can be used to represent XML infoset instances for which no schema exists, as well as XML infoset instances that conform to a schema. In the latter case, the knowledge of the schema may make it easier for the producer of a fast infoset document to identify the range of potential optimizations that can be applied when producing the document, but the consumer of the document will not need to know the schema in order to read the document.

Fast Infoset supports a wide range of optimization techniques, some of which may be effectively applied to GML documents or other large XML documents occurring in the geospatial domain.

An example of such optimization techniques is the use of an external vocabulary. Typically, an external vocabulary is most useful with small documents, because it tends to reduce document size by a relatively fixed amount of bytes. Since the reduction in size is relatively fixed, it quickly becomes negligible as the size of the documents grows. However, the use of an external vocabulary may also have a positive effect on document creation time or on document processing time, in which case it may be beneficial when applied to large documents as well.

Another optimization technique available in Fast Infoset is the use of the so-called "encoding algorithms" ("float", "double", "int", "long", "base64", and so on). The purpose of these encoding algorithms is to allow the direct encoding of floating point values in a binary floating-point format (IEEE 754), the direct encoding of integer values as binary integers (16-bit, 32-bit, or 64-bit), and so on, thus avoiding multiple conversions between the in-memory binary representation and the character-string representation used in an ordinary XML document. Lists of integers and lists of floating point numbers are also optimized. The "base64" encoding algorithm allows the direct inclusion of one or more binary data blocks in the content of an element, and eliminates the need to perform a conversion from binary to Base64 (when creating a document) and from Base64 to binary (when processing a document). This achieves the same goal as XOP and MTOM (efficient transmission of XML documents containing both XML and binary data) but with less overhead.

7.6 General SOAP Guidelines

7.6.1 Which Transport Protocol should be used

The SOAP 1.2 specification allows the use of several transport protocols such as HTTP, SMPT, FTP. To maintain compliancy with the WS-Basic Profile 1.2, only HTTP as the transport protocol is permitted. According to the WS-I Basic Profile: “A `wsdl:binding` element must specify the HTTP transport protocol with SOAP binding. Specifically, the `transport` attribute of its `soapbind:binding` child MUST have the value http://schemas.xmlsoap.org/soap/http”.

7.6.2 What should be part of the header/body

As discussed in section 6.2, a SOAP message has always two sections: The header and the body. Applying the separation of concerns pattern, the SOAP-Body shall be used only for transmitting the actual OWS Service request. For instance, the body of a WMS request message shall only contain the required and optional parameters such as SERVICE, REQUEST, BBOX in an XML representation.

On the other hand, the SOAP-Header is reserved for optional elements (NOT parameters) in order to invoke the service. These optional parameters could be identity tokens, licenses or other elements that are not necessarily required by the implementation specification but state by WS-Policy in the preconditions (See section 5.2.7).

7.6.3 Which SOAP style should be used

SOAP allows different styles and encodings. A SOAP binding can be either in the RPC style or the Document style. Both styles can be combined with either an encoded or literal use. With the additional document/literal wrapped approach, there are 5 different permutations:

1) Document/Literal

2) Document/Literal Wrapped

3) Document/encoded

4) RPC/Literal

5) RPC/encoded

OWS-2 pointed out document literal as the OGC binding of choice since it matches best the OGC Web Service world. This document follows this argumentation and recommends Document/Literal style for SOAP bindings.

7.6.4 How to transfer large binary data

Section 6.2 pointed out, that a SOAP message can only contain XML. But a number of OGC Web Services are designed to return binary data such as a WMS. Therefore, this IPR recommends the use of MTOP (see section 6.3) and XOP (see section 6.4) to efficiently transfer (large) binary data with SOAP. This requires especially the use of a message element with an xsd:base64Binary as can be seen in section 5.2.7.

7.6.5 How to transfer large XML data

Numerous OGC Web Services can easily provide large sets of data. For instance, an unfiltered request to a WFS can result in hundreds of megabyte XML data. Besides performance issues while parsing large XML data, the first obstacle is to transfer such a

large dataset over a network. To reduce the actual on-the-wire payload, Fast Infoset (see section 6.5) should be considered as a potential solution for future Web Services.

7.6.6 How to indicate a SOAP binding in the GetCapabilities Response

The GetCapabilites response lists all offered operations as described in section 5.2.2. The Operations Metadata section is also responsible for describing the operation binding. For a SOAP binding, the following pattern shall be used as described in the OWS Common CR and presented in listing 13:

```
<Capabilities>

...

    <DCP>
        <HTTP>
            <Post name="???">
                <Constraint name="PostEncoding">
                    <AllowedValues>
                        <Value>SOAP</Value>
                        <Value>XML</Value>
                    </AllowedValues>
                </Constraint>
            </Post>
        </HTTP>
    </DCP>
...

</Capabilities>
```

Listing 13: **SOAP Binding described in a GetCapabilities Response**

8 Encapsulating GET and POST Interfaces with SOAP

8.1 Motivation

The reason for this mapping is the fact, that certain developments such as GeoRM only focus on supporting SOAP services and do not define GET and POST bindings. The reason for this is that this work heavily relies on mainstream standards such as SAML, XACML and the WS-* family which more or less only support SOAP.

Accepting the fact that certain work within OGC is done exclusively for SOAP services, the question has to be answered how to deal with existing implementations using GET and POST. Besides the GET, POST and SOAP issue GeoRM faces a similar problem: GeoRM wants to add rights management to services which do not support rights management natively. The solution for that is using proxies enabling non-RM-aware services to support rights management (the same holds for the client side).

8.2 Proxies as Enabling Pattern

On the client side this proxy receives a 'bare' service requests, adds all GeoRM-related elements such as identity or license tokens, and forwards the enriched request to the service.

On the service side there is also a proxy component being able to evaluate these tokens and act accordingly. If the GeoRM evaluation is completed successfully, the 'bare' service request is extracted and forwarded to the non-GeoRM-aware service.

The GET, POST and SOAP problem could be solved in a similar way. If there was a well-defined, injective mapping between GET and SOAP and between POST and SOAP, these proxies could do a protocol transformation as well. This would mean, that for instance a client sends a GET request to the client proxy, the client proxy converts this request into SOAP and can now apply the required GeoRM tokens. On the service side a proxy would receive this request, do the GeoRM-related stuff, and finally transform the SOAP request back to GET and send it to the service. An example is shown in **Error! Reference source not found.**5.

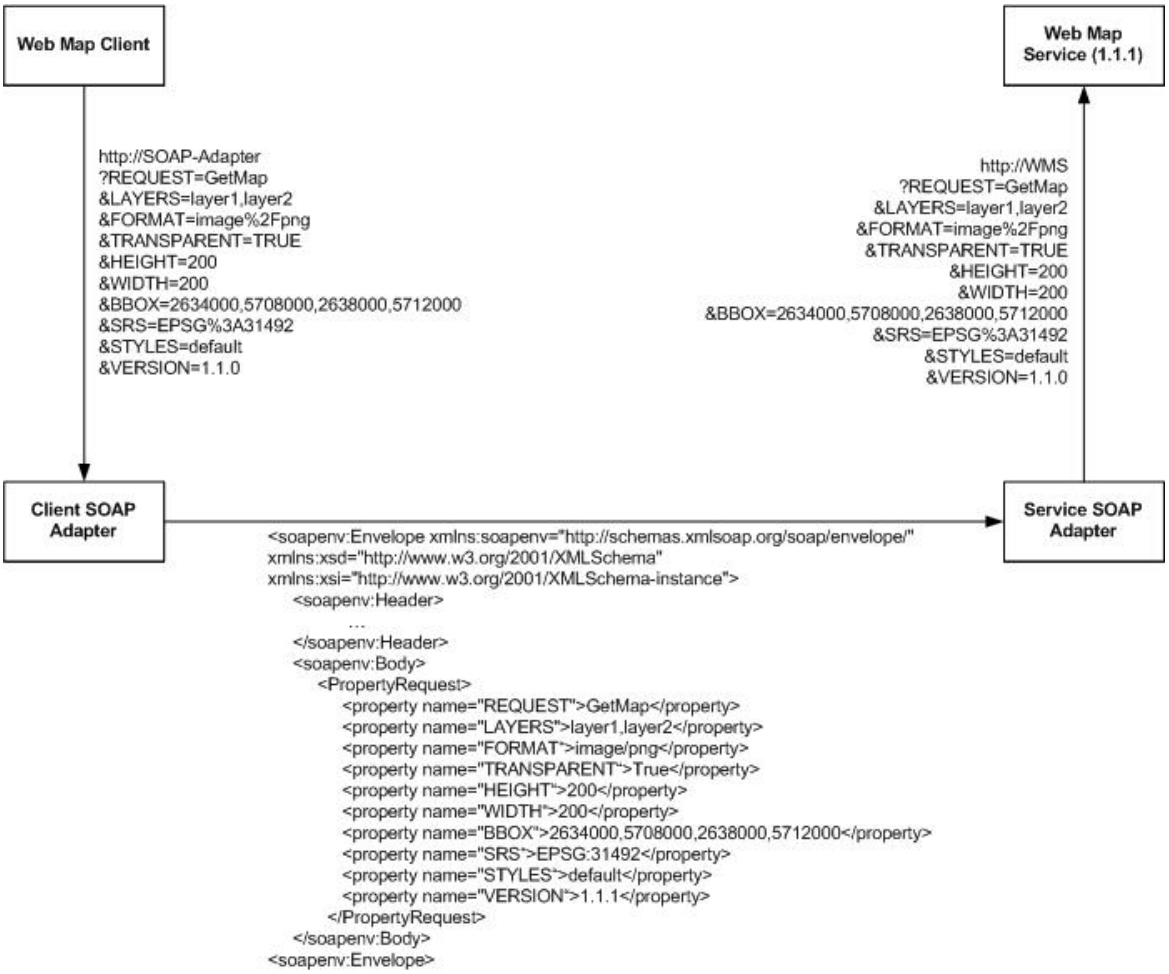

Figure 5: Conversion and Re-Conversion of KVP and XML Encoding

Using SOAP now allows to add Rights Management related information to the request, as shown in **Error! Reference source not found.**6.

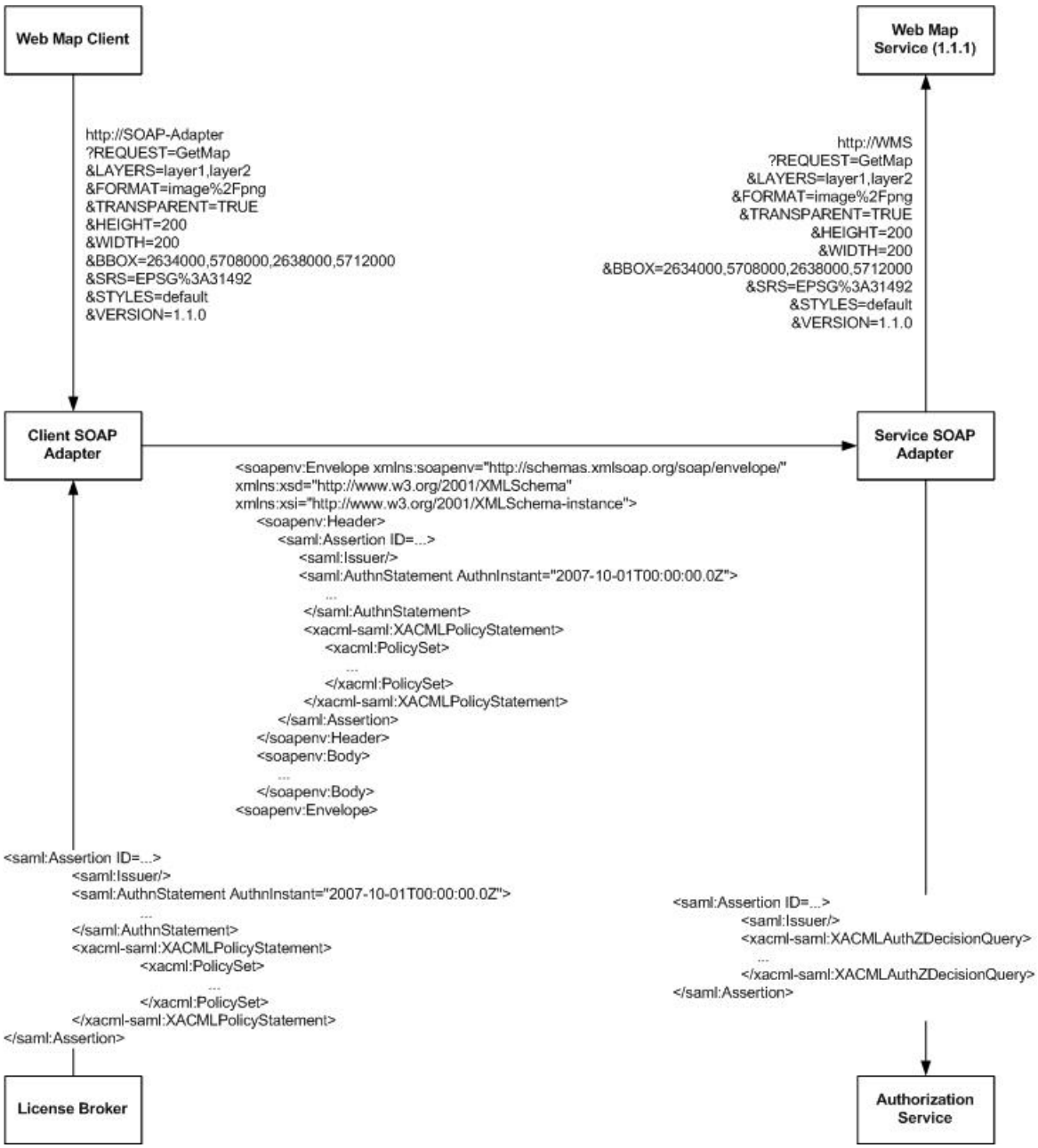

Figure 6: Adding a License Token to the SOAP Request

8.3 Transformation of KVP Encoding

For representing the KVP parameters in XML the following schema can be used:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:element name="PropertyRequest">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="property" minOccurs="0"
            maxOccurs="unbounded">
                    <xs:complexType mixed="true">
```

```
                    <xs:complexContent>
                        <xs:extension type="xs:anyType">
                            <xs:attribute name="name" type="xs:string"
                        use="required"/>
                        </xs:extension>
                    </xs:complexContent>
                </xsd:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:schema>
```

Listing 14: KVP parameter in XML representation schema

The result of such a transformation can be seen in **Error! Reference source not found.**5 above. Each KVP is represented by a `<property name="[Key]">[Value]</property>` tag. A semantic analysis of the request payload will not be performed by the server side proxy. This will be done by the target service based on the reconstructed KVP request.

8.4 Transformation of XML Encoding

POST/XML service interfaces are transformed easily by placing the XML payload into the SOAP body of the SOAP request.

8.5 Capabilities Modifications

A Capabilities document contains for each operation a 'DCPType' node which defines the binding for this operation. The sub node 'HTTP' remains, since also the SOAP binding will be available via HTTP. The sub node under the HTTP node has to be changed from 'Get' or 'Post' to 'SOAP'. The sub node under 'Get' or 'Post' contains the URL to the described operation. This URL has to be changed to the current SOAP wrapper URL.

To identify the original binding of each operation, the `<SOAP>` section has to include a 'constraint' element as follows:

Initial Binding	**Constraint Element**

Get/KVP	`<constraint name="OriginalBinding">`
	`<Value>Get/KVP</Value>`
	`</constraint>`

Post/KVP	`<constraint name="OriginalBinding">` `<Value>Post/KVP</Value>`

	</constraint>
Post/XML	<constraint name=”OriginalBinding”> <Value>Post/XML</Value> </constraint>

Table 1: Binding and Constraint overview

A DCPType element of a capabilities document such as

```
<DCPType>
  <HTTP>
    <Get>
      <OnlineResource xmlns:xlink="http://www.w3.org/1999/xlink"
xlink:href="[ServiceURL?]" xlink:type="simple"/>
    </Get>
  </HTTP>
</DCPType>
```

would then be converted to

```
<DCPType>
  <HTTP>
    <SOAP>
      <OnlineResource xmlns:xlink="http://www.w3.org/1999/xlink"
xlink:href="[WrapperURL]?" xlink:type="simple"/>
      <constraint name="OriginalBinding>
        <Value>Get/KVP</Value>
      </constraint>
    </SOAP>
  </HTTP>
</DCPType>
```

Listing 15: Wrapped Service Capabilities Entry

The constraints element is needed to be able to distinguish between SOAP bindings following a SOAP specification and SOAP bindings resulting from wrapping a GET or POST service. This allows a retransformation to the original protocol.

8.6 Server Side Proxy WSDL Description

The presented proxy approach relies solely on SOAP and therefore describing the service with a WSDL 1.1 [5] document is good practice.

Applying the ideas introduced above, it does not become necessary to distinguish between a KVP or XML OWS requests, since all KVP requests are mapped to XML (see **Error! Reference source not found.**). Furthermore, late binding operations such as

WFS.getFeature do not need to be treated specially, since the proxy approach just forwards the requests and simple xsd:anyType placeholders can be used.

The following sample WSDL should provide an idea on how to incorporate the presented solution into a WSDL:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap12/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:tns="http://new.webservice.namespace"
targetNamespace="http://www.opengis.net/wms/wsdl"
xmlns:wms="http://www.opengis.net/wms"
xmlns:ogcwsdl="http://www.opengis.net/ogc/wsdl">
    <wsdl:types>
        <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified"
targetNamespace="http://www.ifgi.de/kvp2xml">
           <xs:element name="RequestProperty">
              <xs:complexType>
                 <xs:sequence>
                    <xs:element name="property" minOccurs="0"
maxOccurs="unbounded">
                       <xs:complexType mixed="true">
                          <xs:complexContent mixed="true">
                             <xs:extension base="xs:anyType">
                                <xs:attribute name="name" type="xs:string"
use="required"/>
                             </xs:extension>
                          </xs:complexContent>
                       </xs:complexType>
                    </xs:element>
                 </xs:sequence>
              </xs:complexType>
           </xs:element>
        </xs:schema>
        <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
           <xs:element name="binaryPayload" type="xs:base64Binary"/>
        </xs:schema>
        <xs:schema targetNamespace="http://www.opengis.net/wms/wsdl">
           <xs:import namespace="http://www.opengis.net/wms"
schemaLocation="http:/v-ebiz.uni-muenster.de:8083/OWS-5/wms.xsd"/>
        </xs:schema>
    </wsdl:types>
    <wsdl:message name="GetCapaMessage_GET">
        <wsdl:part name="request" type="tns:RequestProperty"/>
    </wsdl:message>
    <wsdl:message name="GetCapaResult">
        <wsdl:part name="response" element="wms:WMT_MS_Capabilities"/>
    </wsdl:message>
    <wsdl:message name="GetMap_GET">
        <wsdl:part name="request" type="tns:RequestProperty"/>
    </wsdl:message>
    <wsdl:message name="GetMapResult">
        <wsdl:part name="response" type="tns:binaryPayload"/>
    </wsdl:message>
    <wsdl:portType name="WMS_Port_Type">
```

```
      <wsdl:operation name="GetCapabilities">
         <wsdl:input message="tns:GetCapaMessage_GET"/>
         <wsdl:output message="tns:GetCapaResult"/>
         <wsdl:fault name="exception" message="ogcwsdl:ServiceExceptionMessage"/>
      </wsdl:operation>
      <wsdl:operation name="GetMap">
         <wsdl:input message="tns:GetMap_GET"/>
         <wsdl:output message="tns:GetMapResult"/>
         <wsdl:fault name="exception" message="ogcwsdl:ServiceExceptionMessage"/>
      </wsdl:operation>
   </wsdl:portType>
   <wsdl:binding name="WMS_SOAP_Binding" type="tns:WMS_HTTP_Port_SOAP">
      <wsdl:documentation>
        WMS interface bound to SOAP over HTTP/1.1.
     </wsdl:documentation>
      <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="GetCapabilities">
         <soap:operation
soapAction="http://www.opengis.net/wms/requests#GetCapabilities"/>
         <wsdl:input>
            <soap:body use="literal" parts="request"/>
         </wsdl:input>
         <wsdl:output>
            <soap:body use="literal"/>
         </wsdl:output>
         <wsdl:fault name="exception">
            <soap:fault use="literal" name="exception"/>
         </wsdl:fault>
      </wsdl:operation>
      <wsdl:operation name="GetMap">
         <soap:operation soapAction="http://www.opengis.net/wms/requests#GetMap"/>
         <wsdl:input>
            <soap:body use="literal"/>
         </wsdl:input>
         <wsdl:output>
            <soap:body use="literal"/>
         </wsdl:output>
         <wsdl:fault name="exception">
            <soap:fault use="literal" name="exception"/>
         </wsdl:fault>
      </wsdl:operation>
   </wsdl:binding>
   <wsdl:service name="WMS">
      <wsdl:port name="WMS_SOAP" binding="tns:WMS_SOAP_Binding">
         <soap:address location="http:/v-ebiz.uni-
muenster.de:8083/kvp2xml/sampleWMS"/>
      </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```

Listing 16: Sample WSDL

Bibliography

[Andrews et al., 2003] Andrews, T., Cubera, F., Dholakia, H., Goland, Y., Klein, J.,Leymann, F., Liu, K., Roller, D. Smith, D., Thatte, S., Trickovic, I. and eerawarana. S. (2003): *Business process execution language for Web Services version 1.1.*, OASIS, Online: http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf (accessed on 10/12/07).

[Atkinson & Martel, 2002] Atkinson, R., Martell, R. (2002): *OWS1 Web Services Architecture*. OGC# 02.022r1

[W3C, 2001] W3C[Ed.] (2001): *Web Service Description Language (WSDL) 1.1*. W3C, Online: http://www.w3.org/TR/wsdl/ (accessed on 10/12/07)

[W3C, 2007] W3C[Ed.] (2007): *Web Service Description Language (WSDL) 2.0*. W3C, Online: http://www.w3.org/TR/wsdl20/ (accessed on 10/12/07)

[W3C, 1999] W3C[Ed.] (1999): *Hypertext Transfer Protocol -- HTTP/1.1*. W3C, Online: http://www.w3.org/Protocols/rfc2616/rfc2616.html (accessed on 10/12/07)

[W3c, 2006a] W3C[Ed.] (2006): *Web Services Policy 1.2 - Framework (WS-Policy)*. W3C, Online: http://www.w3.org/Submission/WS-Policy/ (accessed on 10/12/07)

[W3c, 2006b] W3C[Ed.] (2006): *Web Services Policy 1.2 - Attachment (WS-Policy)*. W3C, Online: ttp://www.w3.org/Submission/WS-PolicyAttachment/ (accessed on 10/12/07)

[WS-I, 2007] Web Services Interoperability Organization[Ed.] (2007): *Basic Profile Version 1.2 Board Approval Draft*. WS-I. Online: http://www.ws-i.org/Profiles/BasicProfile-1.2.html (accessed on 10/12/07)

[OASIS, 2007] OASIS [Ed.] (2007): *WS-SecurityPolicy 1.2*. OASIS. Online : http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702 (accessed on 10/12/07)