JRC Scientific and Technical Reports

INSPIRE NETWORK SERVICES
SOAP Framework

Matteo Villa, Giovanni Di Matteo
TXT e-solutions

Roberto Lucchi, Michel Millot, Ioannis Kanellopoulos
European Commission
Joint Research Centre
Institute for Environment and Sustainability

EUR 23635 - 2008

JRC EUROPEAN COMMISSION

ies Institute for Environment and Sustainability

TXT e-solutions

The mission of the Institute for Environment and Sustainability is to provide scientific-technical support to the European Union's Policies for the protection and sustainable development of the European and global environment.

European Commission
Joint Research Centre
Institute for Environment and Sustainability

Contact information

Matteo Villa
TXT e-Solutions S.p.A.
Via Frigia, 27
20126 Milano (MI)
ITALY
E-mail: matteo.villa@txt.it
Tel.: +39 02 25771
Fax: +39 02 25771828

Giovanni Di Matteo
TXT e-Solutions S.p.A.
Via Frigia, 27
20126 Milano (MI)
ITALY
E-mail: giovanni.dimatteo@txt.it
Tel.: +39 02 25771
Fax: +39 02 25771828

Roberto Lucchi
European Commission Joint Research Centre
Institute for Environment and Sustainability
Spatial Data Infrastructures Unit
TP262, via Fermi 2749
I-21027 Ispra (VA)
ITALY
E-mail: Roberto.Lucchi@jrc.it
Tel.: +39 0332 78 5325
Fax: +39 0332 78 6325

Michel Millot
European Commission Joint Research Centre
Institute for Environment and Sustainability
Spatial Data Infrastructures Unit
TP262, via Fermi 2749
I-21027 Ispra (VA)
ITALY
E-mail: Michel.Millot@jrc.it
Tel.: +39 0332 78 6146
Fax: +39 0332 78 6325

Ioannis Kanellopoulos
European Commission Joint Research Centre
Institute for Environment and Sustainability
Spatial Data Infrastructures Unit
TP262, via Fermi 2749
I-21027 Ispra (VA)
ITALY
E-mail: Ioannis.Kanellopoulos@jrc.it
Tel.: +39 0332 78 5115
Fax: +39 0332 78 6325

http://ies.jrc.ec.europa.eu/
http://www.jrc.ec.europa.eu/

JRC 48715

EUR 23635 EN
ISBN 978-92-79-10966-9
ISSN 1018-5593
DOI 10.2788/36426

Luxembourg: Office for Official Publications of the European Communities

Printed in Italy

s

Table of Contents

Table of Figures

1. Introduction

The technical specifications included in the INSPIRE Implementing Rules define a *common contract* for all the operation interfaces and data description and representation to be used in the European Spatial Data Infrastructure.

Proposal put forward based on the following arguments mentioned in the Network Services Architecture Document ([INSPIRE D3.5]):
As the INSPIRE directive advises to utilize existing standards, OGC service bindings are taken as a guidance. Existing OGC Web Services (OWS) support a mix of protocols and technology bindings. These are Key-Value-Pairs send via HTTP/GET, XML send via HTTP/POST, SOAP via HTTP/POST and combinations. In addition the World Wide Web Consortium (W3C) suggests the usage of SOAP as a messaging protocol for web services. INSPIRE services should utilize one standard technology binding for all service types. In order to streamline integration and implementation as well as getting a maximum benefit from the offered services, a mix of technologies is to be avoided. Taking all requirements, opportunities and risks into account, the default communication-protocol and binding technology for INSPIRE services should be SOAP (document/literal).

A number of arguments can be listed in favour of SOAP/WSDL approach for the INSPIRE network services:

- *SOAP web services are becoming the standard information technology and thus support more sustainable implementing rules;*
- *SOAP web service ensure smooth and complete integration in development environments;*
- *SOAP web services yield a direct and full integration with other web service environments;*
- *SOAP web services have the possibility to support geo rights management services by using SOAP envelope data (see chapter "Geo Rights Management").*

While the Network Services Architecture Document warns about the current lack of formal endorsement of SOAP in the geospatial standards organisation (OGC, ISO TC211):
"However, since the OGC interface specifications support HTTP/GET or HTTP/POST there are only a few attempts to use SOAP/WSDL for geo-information services. Recent experiments e.g. within OGC but also within EU funded Projects such as ORCHESTRA provided interface specification for SOAP/WSDL based geo-information services but also indicated some issues in realising operational services based on SOAP/WSDL, e.g. a lack of support by current development tools."

Whilst consequently recommending SOAP as service binding technology, INSPIRE Network Services can still reuse OGC service specifications. This is because the service binding technologies used by OGC services are service oriented and SOAP service bindings can follow exactly the same design principle. Therefore, a SOAP service interface is a simple technology bridge without influencing any semantics. In addition as by June 2006, all new and revised OGC service specifications have to provide SOAP bindings.

It is important to stress that this first version of a SOAP framework is only the first step towards its potential **endorsement** by the INSPIRE stakeholders **that, through their implementation of the INSPIRE Network Services will provide important feedback on its appropriateness, feedback potentially recorded in the next version of this document**.

2. Purpose and Scope of the Document

The goal of this document is to provide a definition and rationale for a proposed INSPIRE SOAP framework (SOAP nodes policy, RPC, attachments, WS-I, WSDL…) taking into account the issues and solutions pertaining to the specific geospatial domain (for example GML handling in SOAP messages or interfaces definition of the OGC specifications). For the peculiarities and existing solutions of the geospatial domain we will refer at the findings reported in "SOAP HTTP Binding Status, Survey on OGC and ORCHESTRA specifications relevant for the INSPIRE Network Services"[1], that describes the status of SOAP HTTP binding of those services relevant to INSPIRE Network Services, thus including OGC/ISO specifications (CSW, WMS, WFS, WPS, WCTS and WCS) and implementation (OWS3-5), as well as IST ORCHESTRA relevant services.

The next sections of this document are organised in the following way:

- **Chapter 3** provides normative references.
- **Chapter 4** describes the core aspects that need to be considered in the framework.
- **Chapter 5** provides the definition and rationale for the INSPIRE Network Services SOAP framework, clarifying standard versions and supplying specific references for all the necessary specifications.
- **Chapter 6** contains the conclusions about the current study and proposed framework.
- **Chapter 7 - Annex A** provides a set of technical alternatives for each issue reported in chapter 4, trying to highlight the pros and cons of each potential choice.
- **Chapter 8 - Annex B** reports a comparative analysis describing the most relevant differences between the proposed SOAP framework and the currently available corresponding solutions or change requests in OGC.
- **Chapter 9 - Annex C** is a critical analysis, pointing out the issues emerged when applying the SOAP framework criteria to INSPIRE Discovery and View Services.

[1] http://inspire.jrc.ec.europa.eu/reports/ImplementingRules/network/SOAP_binding_survey.pdf

3. Normative References

The following normative documents contain provisions that, through reference in this text, constitute provisions of 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.

REF	AUTHORS	TITLE	YEAR
EUR 23452 EN - 2008	M. Villa et al.	SOAP HTTP Binding Status, Survey on OGC and ORCHESTRA specifications relevant for the INSPIRE Network Services http://inspire.jrc.ec.europa.eu/reports.cfm	2008
INSPIRE D3.5	NSDT	INSPIRE – Network Services Architecture http://inspire.jrc.ec.europa.eu/reports.cfm	2007
SOAP/1.1	Box D., Ehnebuske D. et al.	Simple Object Access Protocol (SOAP) 1.1 http://www.w3.org/TR/soap/	2000
WSDL/1.1	Christensen E., Curbera F. et al.	Web Services Description Language (WSDL) 1.1 http://www.w3.org/TR/wsdl	2001
WS-I BP/1.2	Ferris C., Karmarkar A. et al.	Basic Profile version 1.2 http://www.ws-i.org/	2007

Table 1 - Normative References

These documents could contain list of normative references that are also applicable to this study.
NOTE: Additional normative references to be confirmed, for a complete set of references see the Bibliography chapter.

4. Core SOAP Framework Aspects

After the analysis performed in [EUR 23452 EN - 2008], several common aspects have been underlined, that need to be taken under consideration during the definition of a first draft for the INSPIRE SOAP framework:

1. standard compliances;
2. underlying protocols binding;
3. use of the SOAP Header;
4. exception report in SOAP encoding;
5. data encoding style and use;
6. binary data transport and representation.

Each one of these points needs to be inspected in function of the geo-spatial domain referenced by INSPIRE and to be studied to find a common solution that best fits the project reality.

4.1 Standard compliances

Many standards are present on the market and they are constantly evolving as SOA is getting more and more spread wide. WS-I (*http://www.ws-i.org*) is probably the most important and known standard organisation for interoperability. In particular, WS-I Basic Profile [WS-I BP/1.2] consists of a set of non-proprietary Web services specifications, along with clarifications, refinements, interpretations and amplifications of those specifications which promote interoperability.

4.2 Underlying Protocol Binding

The conclusion of [EUR 23452 EN - 2008] about the transport protocol to be used is that SOAP will use an HTTP binding: SOAP on HTTP offers all the flexibility needed in terms of message complexity and allows to leverage the XML schema definitions of the existing services.
HTTP is suitable for both the Message-Exchange-Patterns that can be required by INSPIRE services: *one-way* and *request-response*, providing all the needed functionalities.
It could be useful to underline that HTTPS could be as well used as lower level protocol to improve security.

4.3 SOAP Headers

Following SOAP specification [SOAP/1.1], the SOAP *Header* should provide a mechanism for extending a SOAP message in a decentralised and modular way.
Actually no OGC/ORCHESTRA service specification document or change proposal examines the SOAP Header theme, aspect that should instead be analysed to provide a common use inside the INSPIRE domain.

As Aravind Corera asserts in [Corera, 2007], the SOAP message processors or handlers may often need to process data pertaining to security, transactions, routing, etc. and this data need to be conveyed with the SOAP message request or response. It could turn out **not to be a good decision to tie this data with every single message request or response definition**. Such orthogonal data are best suited to be transported using out-of-band mechanisms, such as by piggybacking them on SOAP headers. Information such as user authentication credentials, transaction identifiers, message routing URI identifiers, etc. are perfect candidates to be moved and stored into SOAP headers. These portions of the messages can be processed by intermediary nodes (actors) along the SOAP message-processing pipeline, or by the final recipient node (the web method) itself.

In section 7.3 – “SOAP Headers”, we will analyse how SOAP headers could be used for various uses like security or data encryption, for documentation, providing descriptive comments on the content of the message, for routing policies, affecting the behaviour of the targeted receiver, or to add message meta-data, providing information (potentially) unrelated to the message body.

4.4 Exception report SOAP encoding

A common SOAP framework needs to manage code exceptions in the same way for each web service, to give a common behaviour to the whole domain. The INSPIRE framework will face this need proposing a common structure to be followed each time some sort of exception is raised.
Generally, exception handling involves different kinds of behaviours, as identification of exception, logging and notification; anyway it’s out of the scope of this document to deal with these topics that should be analysed in a more methodological and architectural way; this deliverable will only specify the SOAP message structure of a response message containing a fault.

4.5 Data encoding style and use

SOAP makes use of XML to marshal data that is transported to a software application. Most of the time, SOAP moves data between software objects, but the SOAP specification was intended to be useful for old legacy systems as well as for modern object-oriented systems. Consequently, SOAP defines more than one data encoding method to convert data from a software program into XML format and back again. The SOAP-encoded data is packaged into the body of a message and then sent to a host; the host then decodes the XML-formatted data back into a software object.

As considered in [Cohen, 2003], choosing a kind of encoding or another one can change both the functional results of the web services and the physical structure of the resulting WSDL, so it will be surely useful to inspect all possible kinds of alternatives, to perform the best choice for geospatial domain services.

4.6 Binary data transport and representation

Giving a glance to INSPIRE services functionalities, it is evident that they will often have the need to send quite large amount of binary data as answer to their invocation, as in the case, for instance, of the *Download* service.
Defining an optimal solution for the transport of such a kind of data can turn out to be very performance improving.
Traditionally, two techniques for dealing with opaque data in XML have been used: **"by value"** and **"by reference"**. The former is achieved by embedding opaque data as element or attribute content. XML supports opaque data as content through the use of either *base64* or *hexadecimal* text encoding. This approach is codified by XML Schema in two binary data types: *xs:base64Binary* and *xs:hexBinary*. The lexical representation of the *xs:hexBinary* is a simple hexadecimal character sequence; the lexical representation of *xs:base64Binary* uses the *base64* algorithm as defined by RFC 2045 [RFC2045].

The following XML snippet demonstrates the use of *base64* in a simple XML document.

```
<m:data xmlns:m='http://example.org/people' >
   <photo>/aWKKapGGyQ=</photo>
   <sound>sdcfo2JTiXE=</sound>
   <hash>Faa7vROi2VQ=</hash>
</m:data>
```

Listing 1 – Sample base64 binary data representation

As both [Yang, 2007] and [XML SOAP BD] state, it is well-known that *base64* encoded data expands by a factor of ***1.33x*** original size and that hexadecimal encoded data expands by a factor of ***2x*** (assuming an underlying *UTF-8* text encoding in both cases; if the underlying text encoding is *UTF-16*, these numbers double). Also of concern is the overhead in processing costs (both real and perceived) for these formats, especially when decoding back into raw binary. When comparing *base64* decoding to a straight-through copy of opaque data, the throughput decreases by a factor of ***3x*** or more.

A more common way to reference external opaque data is to simply use a URI as an element or attribute value. XML Schema supports this explicitly through the *xs:anyURI* type.

```
<?xml version="1.0" ?>
<data>
  <photo data="http://example.org/me.jpg"/>
  <sound data="http://example.org/it.wav"/>
  <hash data="http://example.org/my.hsh"/>
</data>
```

Listing 2 – Sample URI binary data link

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

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

Listing 3 – Sample of URI data-type definition by XML schema

What could be useful in a logical way of thinking is to split the Web service/application control flow of the messages from the flow of data: this should be a quite simple solution.
But this different kind of treatment could also bring to some difficulties: it is surely a big mental overhead for developers and technical people who will need to take care of different systems to manage data in different ways and carry out different kinds of concepts. Moreover, we cannot always be sure of the benefits that this technical solution is going to bring: some studies, like [BXSA], reported that if the attached binary data is small, there could even be worsening in the call performances.

5. SOAP framework for INSPIRE Network Services

This chapter proposes a SOAP framework for INSPIRE Network Services, based on the analysis reported in the previous chapter. The rationale of the framework is based on **maintaining conformity to WS-I Basic Profile 1.2** [WS-I BP/1.2]. A detailed description of the available WS-I Basic Profile versions and their differences is reported in Annex A - Section 7.1.
This choice allows the use of standard and common use technologies and it brings only some more formal constraints to the final XML, SOAP and WSDL morphology. Anyway, it should be noticed that both WS-I and other standardization bodies may release new versions of the standards (e.g. WS-I Basic Profile 2.0 has already been announced). While the current version of the SOAP framework is based only on approved standards, section 8 of this document will report the most well known announced releases, that could be taken into account in future revisions of the framework.

5.1 Protocol Binding

SOAP 1.1 will be the initial SOAP version used for the INSPIRE framework.
SOAP 1.1 specification can be accessed at *http://www.w3.org/TR/2000/NOTE-SOAP-20000508/* [SOAP/1.1].

It could be useful to underline that OGC recommends the use of SOAP 1.2 as messaging protocol, as stated in both [08-009r1] and [07-045]. This is going to be the choice proposed by this study too, as soon as the WS-I Basic Profile 2.0 will change from draft to final version, since this new profile version is built on the last version of the SOAP specification.

As already explained in the Protocol Binding paragraph of chapters 4 and as it will be better explained in chapter 7, HTTP will be chosen as transport protocol.
The choice between HTTP 1.0 and HTTP 1.1 is quite prosaic: HTTP 1.1 is the steady and actual standard de facto. There are plenty of studies proving the benefits brought by 1.1 version and this is the approach everybody follows if no other constraints are present.
So **the INSPIRE SOAP framework will lay on HTTP 1.1 as transport protocol.** Its specification is available at *http://www.w3.org/Protocols/rfc2616/rfc2616.html* [HTTP/1.1].
This first release of the INSPIRE SOAP framework will be represented using WSDL 1.1. The WSDL 1.1 specification is accessible at *http://www.w3.org/TR/wsdl* [WSDL/1.1].
XML 1.0 will be the language used for serialising SOAP envelopes as required by WS-I rules.
The WS-I Profile requires XML processors to support both the "UTF-8" and "UTF-16" character encodings, in order to aid interoperability. In this framework UTF-8 character encoding is the preferred choice for SOAP messages.

5.2 SOAP technical aspects

5.2.1 Data encoding

The INSPIRE SOAP framework will use a *Document-literal wrapped* data encoding.

5.2.2 Binary data transport and representation

About binary data transport and codification, **the binomial Message Transmission Optimisation Mechanism 1.0 (MTOM) and XML Optimisation Package 1.0 (XOP) will be used**, to have better performances.
This choice is coherent with what specified by WS-I Basic Profile 1.2 [WS-I BP/1.2].
MTOM specification is published at *http://www.w3.org/TR/2005/REC-soap12-mtom-20050125* [MTOM], its binding for SOAP 1.1 can be retrieved at *http://www.w3.org/Submission/soap11mtom10* [SOAP/1.1-MTOM], while XOP 1.0 standard is on-line at *www.w3.org/TR/xop10* [XOP].
XOP is going to be used encapsulated inside **MIME messages**. Current MIME version is 1.0 and is specified in six linked RFC memoranda: RFC 2045 (*http://www.ietf.org/rfc/rfc2045.txt*), RFC 2046 (*http://www.ietf.org/rfc/rfc2046.txt*), RFC 2047 (*http://www.ietf.org/rfc/rfc2047.txt*), RFC 4288 (*http://www.ietf.org/rfc/rfc4288.txt*), RFC 4289 (*http://www.ietf.org/rfc/rfc4289.txt*) and RFC 2077 (*http://www.ietf.org/rfc/rfc2077.txt*), which together define the specifications.
If we consider large XML data, like GML, the proposed choice is just to encapsulate it in the body of the message, without performing any transmission or data optimisation.

5.2.3 SOAP Headers

After the analysis of the proposed Headers done in paragraph 7.3 "SOAP Headers", we think that these extension features should be used for

- security;
- checksums;
- signatures;
- metadata (multilingualism).

5.2.4 SOAP Exceptions

As already written in "Exception report SOAP Encoding" paragraph, the need to manage fault exception in the common SOAP framework is present and will be fulfilled as described in that section. Here after we quote again the proposed use the use *Fault* element:

```
<soap:Envelope xmlns:soap=http://www.w3.org/2003/05/soap-envelope
xmlns:ows=http://www.opengis.net/ows/1.2>
  <soap:Body>
    <soap:Fault>
      <soap:Code>
        <soap:Value>soap:Server</soap:Value>
      </soap:Code>
      <soap:Reason>
        <soap:Text>A server exception was encountered.<soap:Text>
      </soap:Reason>
      <soap:Detail>
        <ows:ExceptionReport>
          ...
        </ows:ExceptionReport>
```

```
        </soap:Detail>
      </soap:Fault>
    </soap:Body>
</soap:Envelope>
```

Listing 21 – SOAP exception code sample

The <*Code*> element shall have the Value *"soap:server"* indicating that this is a server exception. The Reason element shall have the *Text "Server exception was encountered"*. This fixed string is used since the details of the exception shall be specified in the <*Detail*> element using an *ows:ExceptionReport* element.

5.3 Node Routing Policies

As it will be deepen in Annex A, sub-section "7.3.4 - Node Policies", up to now no routing algorithms or constraints have been specified by current INSPIRE official documents, so this theme is not going to be taken into consideration.
If requests about this issue will be proposed in the next future, they will be analysed in the next framework releases.

6. Conclusions

The proposed INSPIRE SOAP framework has been built following the study previously performed in [EUR 23452 EN - 2008] on the state of SOAP binding in OGC and ORCHESTRA specifications, collecting problems, suggestions and ideas bound to the SOAP communication in geo-spatial domain web-services.

Open choices related to the use of SOAP have been analysed and grouped into macro-topics relevant to INSPIRE use-cases. Particular attention was paid to current standards in order to keep the highest possible degree of interoperability, as this is one of the most important points outlined in INSPIRE project recommendations.
We decided to follow WS-I Basic Profile 1.2 [WS-I BP/1.2], because it guarantees a good interoperability level and because the previous version has been endorsed also by ISO; WS-I Basic Profile version 1.1 has completed the process for becoming ISO standard, precisely ISO/IEC 29361:2008 (for more details on the differences between the two versions see section 7.1).
This INSPIRE SOAP framework is in its first public version; a new version is already planned and it will consider the usage of WS-I Basic Profile version 2.0 (currently available as draft version). The implication on the INSPIRE SOAP framework of such change are summarized in Table 2; as we can notice, no technical issue is expected for a future change of WS-I profile version compliancy. The only significant variation would be the move from SOAP 1.1 version to SOAP 1.2 release. The new version of the INSPIRE SOAP framework will also take into account the adequacy of the proposed framework for other web services like (OGC WFS, WCTS, WCS and WPS).

In the following table we underline the dependencies between the technical choices followed in this framework and the WS-I Basic Profiles dictates.

Technical choice	WS-I dependency	WS-I evolvement dependencies
HTTP 1.1	WS-I/ALL	
XML 1.0	WS-I/1.2	WS-I/2.0 -> OK
SOAP 1.1	WS-I/1.2	WS-I/2.0 -> SOAP 1.2
WSDL 1.1	WS-I/ALL	
MTOM+XOP	WS-I/1.2	WS-I/2.0 -> OK
Fault management	WS-I/1.1 compliant	WS-I/2.0 -> OK
UTF-8	WS-I/ALL	

Table 2 - Framework technology choices bound to WS-I profiles dictates

The protocol bindings, the data encoding style, the use of the SOAP message headers in the INSPIRE web services, the exception management and the way to transport and to represent binary data in SOAP messages are the most relevant topics that have been treated in this document.
All these subjects have been faced following a coherent approach: analysing the problem itself, looking for possible and widely adopted solutions, analysing their advantages and disadvantages, supplying suitable examples and trying to find the most fitting technical answers to the INSPIRE needs.

7. Annex A - Architectural alternatives

In this chapter we are going to analyse the technical aspects reported in the previous chapter, providing an exhaustive report of the standard (W3C and WS-I) solutions, and describing advantages and disadvantages that have been used to define the INSPIRE SOAP framework.

7.1 Standard Profiles

The choice of being WS-I compliant could provide very positive effects to the INSPIRE framework, enhancing the interoperability level under a theoretical point of view: WS-I Basic Profile defines the set of technologies and corresponding versions that facilitate the interoperability among services. A variety of development tools (e.g. .NET, Java Web Service development pack, Axis) is referring to this profile; therefore its usage would also guarantee the sustainability of the development process.

Just to understand the direction where WS-I points to, it could be meaningful to report the main statement present on their web-site: "WS-I is an open, industry organization chartered to promote Web services interoperability across platforms, operating systems, and programming languages. The organization works across the industry and standards organizations to respond to customer needs by providing guidance, best practices, and resources for developing Web services solutions." [WS-I Website, home page, 2002-02-07]

Along the last years, WS-I followed the improvements and the technological evolution of the market, providing new different versions of the WS-I Basic Profile, as long as new standards and new technologies were developed and acquired market.
The first version (1.0) was edited in 2004, while the last one (2.0) is in draft release nowadays (2008). In the following table we try to summarise the main technological choices dictated by the different WS-I profile releases; technology improvements brought in by new versions are reported in bold.

PROFILE VERSION	DATE	TECHNICAL CONSTRAINTS
1.0	2004	HTTP 1.1, XML 1.0, WSDL 1.1, SOAP 1.1, UDDI 2.0, SSL 3.0, TLS 1.0
1.1	2006	HTTP 1.1, XML 1.0, WSDL 1.1, SOAP 1.1, UDDI 2.0, SSL 3.0, TLS 1.0, **SwA**
1.2	2008 (to be finalised)	HTTP 1.1, XML 1.0, WSDL 1.1, SOAP 1.1, UDDI 2.0, SSL 3.0, TLS 1.0, **MTOM+XOP, WS-Addressing 1.0**
2.0	2008 (draft)	HTTP 1.1, **XML Infoset**, WSDL 1.1, **SOAP 1.2, UDDI 3.0,** SSL 3.0, TLS 1.0, MTOM+XOP, WS-Addressing 1.0

Table 3 - WS-I profiles versions

An aspect that certainly strengthens the choice to follow WS-I profiles is that they are also taken under consideration by the International Organization for Standardization (ISO): for instance, Basic Profile Version 1.1, Attachments Profile Version 1.0 and Simple SOAP Binding Profile Version 1.0 have been published by ISO (*http://www.iso.org*) as ISO/IEC 29361, 29362 and 29363.

The INSPIRE SOAP Framework is based on WS-I version 1.2 [WS-I BP/1.2], mainly because it is relying on improved message optimization transmission protocols, as also reported in section 7.6. However, as reported in the specification: “There are a few requirements in the Basic Profile 1.2 that may present compatibility issues with clients, services and their artifacts that have been engineered for Basic Profile 1.1 conformance”. In particular, none of the reported known backward compatibility

issues shall prevent the usage of existing WS-I 1.1 compliant services and clients in the frame of INSPIRE Network Services.

It could be meaningful to underline that the Basic Profile 2.0 [WS-I/2.0] is the first version of the WS-I Basic Profile that changes the version of SOAP in the profile scope from SOAP 1.1 to the W3C SOAP 1.2 Recommendation. As such, clients, servers and the artefacts that comprise a Basic Profile 1.0, 1.1 or 1.2 conformant applications are inherently incompatible with an application that is conformant with the Basic Profile 2.0. However, in general, in the Basic Profile 2.0 the WG members have tried to preserve as much consistency with the guidance and constraints of the Basic Profile 1.0, 1.1 and 1.2 as possible.

UDDI, SSL and TLS are also technologies dealt in the WS-I profiles, but these aspects are not in the scope of this study.

Performing a market analysis is a fundamental input in order to take an interoperability direction, together with the technological proposal of the single standards. Following this trend, we can cite a survey conducted in August 2006 by the SeCSE European project (*http://secse.eng.it/*), about SOA adoption.
The survey has been proposed on several themes related to web-services to several European companies of different size and location; Figure 1 reports the result of the survey concerning adopted standards.

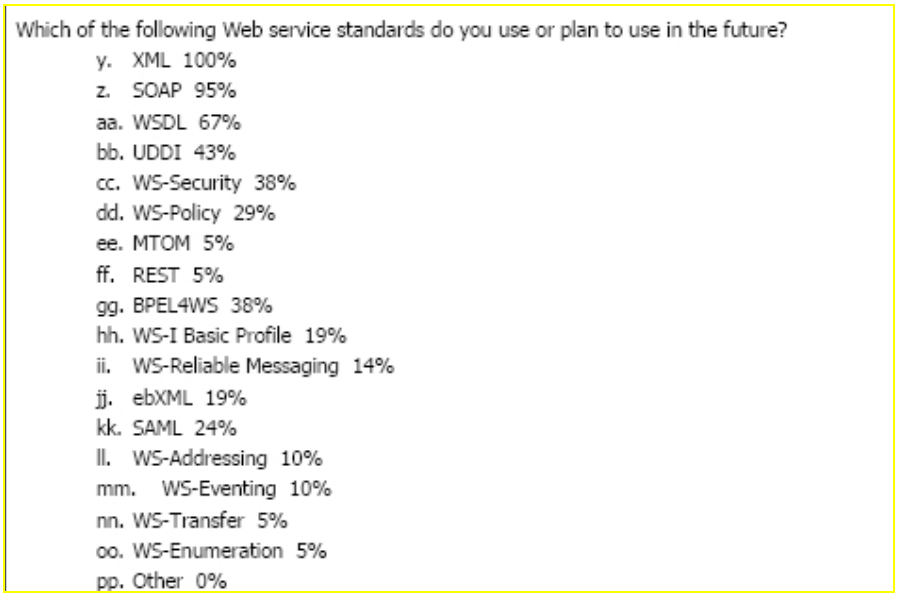

Figure 1 - Standard adoption survey results

As we can see, quite one company each five was using or foresaw to use the WS-I Basic Profile in 2006 and in the last two years the rate has surely grown.

7.2 Underlying Protocols Binding

The SOAP specification allows the use of several transport protocols such as HTTP, SMTP, or FTP. The choice that most make sense under this point of view, keeping in mind the INSPIRE domain and specifications and the choices made by WS-Basic Profiles, is that HTTP will be used as transport protocol.
As already mentioned in section 4.2, this choice covers all the needs of INSPIRE web services and is the standard one for the largest part of SOAP based web-services.

7.3 SOAP Headers

If we analyse the SOAP Header topic, there is no compulsory need to use these extension capabilities provided by SOAP inside the INSPIRE SOAP framework, as neither in OGC, nor in ORCHESTRA, this opportunity is taken under consideration.
Anyway it could turn out to be a good idea to manage some common side aspects of the INSPIRE web services in a modular way, delegating their handling to the Headers.
Drawing on an article written by *Yahoo*'s technical leader Mark Nottingham on SOAP headers classification, in the following sub-paragraphs we will try to propose some ideas on which aspects could be treated as SOAP extensions.

7.3.1 Security

Some services could have the need to grant access to their functionalities, or to some set of data, only to a restricted group of people.
In a web-service this problem can be accomplished in several technical ways and one of these is to exploit the potentials of SOAP Headers, with several benefits, enumerated at the end of this paragraph.

Following [WSS/1.0] instructions, an *Authentication header* block could provide a mechanism for attaching security-related information targeted at a specific recipient in the form of a SOAP actor/role. This may be either the ultimate recipient of the message or an intermediary one. Consequently, elements of this type may be present multiple times in a SOAP message. An active intermediary on the message path MAY add one or more new sub-elements to an existing Authentication header block if they are targeted for its SOAP node, or it MAY add one or more new headers for additional targets.
As stated, a message MAY have multiple *Authentication header* blocks if they are targeted for separate recipients. However, only one *Authentication header* block MAY omit the actor or role attributes. Two *Authentication header* blocks MUST NOT have the same value for actor or role. Message security information targeted for different recipients MUST appear in different header blocks; this is due to potential processing order issues (e.g. due to possible header re-ordering). The *Authentication header* block without a specified actor or role MAY be processed by anyone, but MUST NOT be removed prior to the final destination or endpoint.
As elements are added to an *Authentication header* block, they SHOULD be pre-pended to the existing elements. As such, the *Authentication header* block represents the signing and encryption steps the message producer took to create the message. This pre-pending rule ensures that the receiving application can process sub-elements in the order they appear in the *Authentication header* block, because there will be no forward dependency among the sub elements. It has to be noted that this specification does not impose any specific order of processing the sub elements: the receiving application can use whatever order is required.

A specific *Authentication Header* could house some sort of user credentials to be checked server-side, like in the following example where you can see a SOAP message with a header containing a user name and password:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <AuthHeader xmlns="http://tempuri.org/">
      <UserName>Donald</UserName>
      <Password>Duck</Password>
    </AuthHeader>
  </soap:Header>
  <soap:Body>
    ...
```

```
    </GetQuote>
    </soap:Body>
</soap:Envelope>
```

Listing 4 – Sample Authentication header storing username and password

Another possibility different from the username-password couple could be the transmission of an authentication/identity token checked at server side.

Usually the user credentials are passed to web-methods as input parameters, but using the SOAP header we can be sure that:

- different credentials can be specified for each one of the different recipient of the transmission path;
- only the specified recipient target of the routing will manage the user credentials;
- method signatures would be simplified;
- authentication management would be totally apart from the methods logic and could be changed without modifying the methods signatures.

7.3.1.1 Network Services Drafting Team Proposal

The INSPIRE Network Services Drafting Team has made available a new version of the Network Services architecture ([INSPIRE NSA/3.0]) where the right management issue in INSPIRE is analysed in the context of SOAP binding based INSPIRE Network Services.
The proposed approach, based on the usage of rights management keys included in the header, is totally coherent with the discussed usage of the SOAP header for security aspects.

7.3.2 Checksums & Signatures

Message producers may want to enable message recipients to determine whether a message was altered in transit and/or to verify that the claims in a particular security token apply to the producer of the message.
A specific header could house a string to be used on the receiver's side to check the integrity of the binary data attached.
Another possibility could be that the *Security header* block MAY be used to carry a signature (maybe compliant with the XML Signature specification [XML SSP]) within a SOAP Envelope for the purpose of signing one or more elements stored in the SOAP Body.
Both solutions seem to address the need of INSPIRE web-services: the checksum use could be useful to the *Download* service, where big maps or datasets are sent to the client and the integrity could reveal errors.
The signature use could be useful too, when the INSPIRE consortium needs to send authoritative data, which need a certificate to trust their origin. EU Member States are investing efforts and money in interoperable Electronic Identity Management (eIDM) [eIDM art] and some results from this research could be used in this sense, exploiting the SOAP header structure.

7.3.3 Description & human readable information

A SOAP header could be used also to insert some sort of human readable information or description concerning the message transported in the SOAP body, without being critical to its interpretation.
A possible INSPIRE use could be to equip the downloaded map, for instance in the *Download* service, with some human readable information or description.

7.3.4 Node Policies

Under this hypothetical category, we can gather together all those aspects affecting the behaviour of the targeted receiver, not tightly bound to message semantics.
As simple samples we can mention behaviour like the ones described in *WS-Addressing* [WS-Addr] for *wsa:ReplyTo*, or *wsa:FaultTo*.
Up to now, no need for specific routing algorithms or constraints have been presented in INSPIRE specification, but future needs in this direction could be solved through this kind of header.
For this reason it could be useful to analyse this aspect and see how SOAP can help managing the issue.
Quoting [SOAP/1.2], "SOAP provides a distributed processing model that assumes a SOAP message originates at an initial SOAP sender and is sent to an ultimate SOAP receiver via zero or more SOAP intermediaries".
This routing managing in SOAP can be very useful for some federated aspects like security or QoS, when the domain is not a simple peer-to-peer network, but something more articulated, like in the case of INSPIRE.
SOAP distinguishes between nodes with different roles in the transmission path, assigning a label/role to each one of them. A specific routing/security header can be sent together with the SOAP message, containing information or input to be managed by a peculiar node or set of nodes, being sure it will be handled.
Such in a way all the nodes composing the path can be seen as active members and can have a part in tasks as message distribution or security management.

7.3.5 Metadata & Piggy-backing

The goal of this kind of header is quite straightforward: it should be used to transport some sort of meta-data bound with the SOAP message, or some information that are not strictly connected with the core message transported in the body, but that could be useful for example at the application level.
An example of such a header use could be the *wsa:MessageID* element of *WS-Addressing* [WS-Addr] headers, or the *wsrm:AckRequested* or *wsrm:SequenceAcknowledgement* headers defined in *WS-ReliableMessaging* [WS-RM].

Under the INSPIRE point of view, the multilingualism issue seems to perfectly fall into this sort of situation.
Each request-response couple of messages potentially needs to house the requested idiom code related to the user desired language. To store such an information as an input parameter, could be annoying, performance worsening and uselessly repetitive.
Storing the language code as a meta-data header would bring about several benefits:

- it would simplify the method signature of all the INSPIRE operations, because the language would not be an input parameter anymore;
- it would avoid the user to specify the be-wished language in all its requests, because the header should be set only at the first INSPIRE method invocation; all the following requests to that method or to other ones in that session would automatically incorporate the same header, unless the user doesn't wish to change or remove it;
- it would allow managing multilingualism in an independent way, not bounding it to the single method or service. Such in a way, the eventual changes in the approach to manage the multilingual requests would not influence the methods signatures or the methods logic.

7.4 Exception report SOAP Encoding

SOAP provides a specific construct to represent faults occurred during the execution of a web-service: the <*Fault*> element. This element is used to carry error information in a SOAP message.
For the full syntax of the <*Fault*> element we cross-refer [SOAP/1.1].

Following the various change proposals delivered by OGC, we can track a common structure of a SOAP message when an exception is raised: if an error is detected while processing an operation request encoded in a SOAP envelope, the server shall generate a SOAP response message where the content of the <*Body*> element is a <*Fault*> element containing an <*ExceptionReport*> item.

This shall be done using the following XML fragment:

```
<soap:Envelope xmlns:soap=http://www.w3.org/2003/05/soap-envelope
xmlns:ows=http://www.opengis.net/ows/1.2>
  <soap:Body>
    <soap:Fault>
      <soap:faultcode>soap:Server</soap:faultcode>
      <soap:faultstring>A server exception was encountered.</soap:faultstring>
      <soap:detail>
        <ows:ExceptionReport>
          ...
        </ows:ExceptionReport>
      </soap:detail>
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

Listing 5 – Sample SOAP Exception message

The <*faultcode*> element shall have the value "*soap:server*" indicating that this is a server exception. The <*faultstring*> element shall have the value "*Server exception was encountered*". This fixed string is used since the details of the exception shall be specified in the <*Detail*> element using an <*ows:ExceptionReport*> element.

7.5 Data Encoding style and use

Following W3C definitions [WSDL/1.1], a Web Services Description Language (WSDL) binding *style* can be **RPC** or **Document**, while the *use* can be **Encoded** or **Literal**.
A WSDL, as known, is a document that describes a Web service. In this document, the supported web service operations and messages are described abstractly and then bound to a concrete network protocol and message format.

As stated by [Rothaug, 2004], a WSDL binding describes how the service is bound to a messaging protocol, like HTTP GET/POST, MIME, or SOAP. It is SOAP that the *RPC/Document* distinction refers to.
The <*wsdl:binding*> element of the WSDL contains a pair of parameters that influence the form of the resulting SOAP messages: **binding style** ('RPC' or 'document') and **use** ('encoded' or 'literal').

Four permutations of style/use models are consequently available:

1. RPC/Encoded
2. RPC/Literal
3. Document/Encoded
4. Document/Literal

If we add to this collection a pattern which is commonly called as the D*ocument/Literal Wrapped* pattern, we have five binding styles to choose from, when creating a WSDL file.

7.5.1 The "Style" Attribute

WSDL specifies the style of the binding as either '*RPC'* or '*document'*; it has nothing to do with a programming model: this choice corresponds to how the SOAP payload - i.e., how the contents of the <*soap:Body*> element - can be structured.
Here are some details of how each style affects the contents of <*soap:Body*>:

- **Document**: the content of <*soap:Body*> is specified by XML Schema defined in the <*wsdl:type*> section. It does not need to follow specific SOAP conventions. In short, the SOAP message is sent as one "document" in the <*soap:Body*> element without additional formatting rules having to be considered. *Document* style is the default choice.
- **RPC**: the structure of an RPC style <*soap:Body*> element needs to comply with the rules specified in detail in Section 7 of the [SOAP/1.1]. According to these rules, <*soap:Body*> may contain only one element that is named after the operation (in this case the operation name is used) and all parameters must be represented as sub-elements of this wrapper element.

7.5.2 The "Use" Attribute

The *use* attribute specifies the encoding rules of the SOAP message and this is done within the <*wsdl:binding*> element. The value can be '*encoded'* or '*literal'*. It refers to the serialization rules followed by the SOAP client and the SOAP server to interpret the contents of the <*Body*> element in the SOAP payload.

- **Literal**: it means that the type definitions literally follow an XML schema definition.
- **Encoded**: it refers to the representation of application data in XML, usually according to the SOAP encoding rules of the SOAP specification. The rules to encode and interpret a SOAP body are in an URL specified by the *encodingStyle* attribute. *Encoded* is the appropriate choice where non-treelike structures are concerned, because all others can be perfectly described in XML Schema.

It's useful to remind that the terms *encoded* and *literal* are only meaningful for the WSDL-to-SOAP mapping.

7.5.3 OGC/ORCHESTRA situation

As reported in [EUR 23452 EN - 2008], there is no common choice on the WSDL data encoding style and use in all the OGC and ORCHESTRA specifications.
Following these proposals, there are only two possible choices to be undertaken:

- Document/literal, which has been proposed by ten documents.
  - Two specifications (ORCHESTRA WPS [PRO WSDL] and OWS Common Change Request [06-094r1]) propose the *wrapped* style;
  - one specification (ORCHESTRA WCTS [IS SMS]) proposed the *non-wrapped* style;
  - all the other specifications do not dip to this detail depth.
- Two services specifications (OGC WFS [04-094] and CSW Change Proposal [04-042]) leave the choice open to two different options RPC/literal or Document/literal

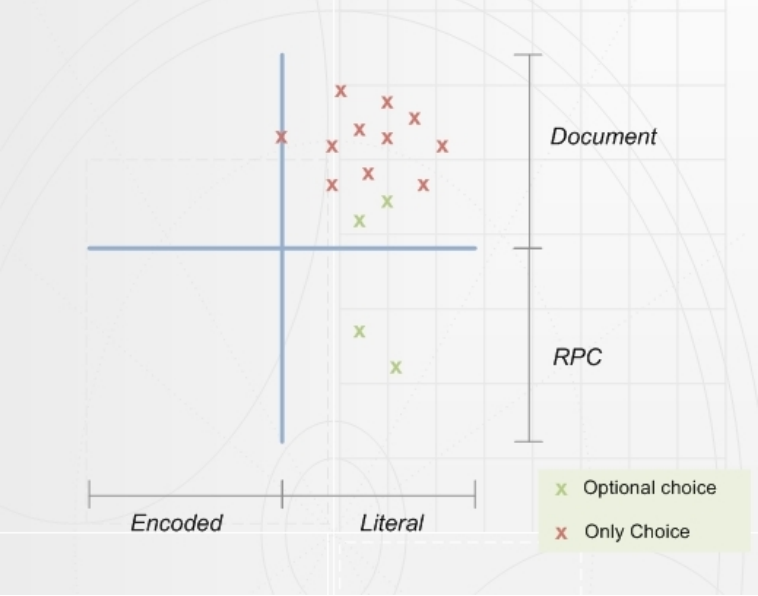

Figure 2 - OGC Data encoding choices space representation

7.5.4 Examples

It could be useful to show an example of the differences between the five permutations, underlying the strengths and the weaknesses of each one of them, to choose the most appropriate one for the INSPIRE SOAP framework. Such an example is reported from the Russel Butek article [Butek, 2005].
We start from a simple function written in JAVA called *myMethod*, that doesn't return any value and takes as input two parameters: an integer and a float value. Here comes the listing:

```
public void myMethod(int x, float y);
```

Listing 6 -The Java sample method

Now we can examine how this simple function is translated in a WSDL and which are its SOAP messages, depending on the style/use couple we choose.

7.5.4.1 RPC/Encoded

Here following the *RPC/encoded* WSDL for *myMethod*:

```
<message name="myMethodRequest">
  <part name="x" type="xsd:int"/>
  <part name="y" type="xsd:float"/>
</message>
<message name="empty"/>
<portType name="PT">
  <operation name="myMethod">
    <input message="myMethodRequest"/>
    <output message="empty"/>
  </operation>
</portType>
<binding .../>
<!-- I won't bother with the details, just assume it's RPC/encoded. -->
```

Listing 7 - RPC/encoded WSDL for myMethod

Now we assume to invoke this method with "5" as the value for parameter *x* and "5.0" for parameter *y*. That sends a SOAP message which looks something like the following listing:

```
<soap:envelope>
  <soap:body>
    <myMethod>
      <x xsi:type="xsd:int">5</x>
      <y xsi:type="xsd:float">5.0</y>
    </myMethod>
  </soap:body>
</soap:envelope>
```

Listing 8 - RPC/encoded SOAP request message for myMethod

Now we can look at the strengths and at the weaknesses of this choice:

Strengths

- The WSDL is about as straightforward as it's possible for WSDL to be.
- The operation name appears in the message, so the receiver has an easy time dispatching this message to the implementation of the operation.

Weaknesses

- The type encoding info (such as *xsi:type="xsd:int"*) is usually just overhead which degrades throughput performance.
- You cannot easily validate this message since only the *<x ...>5</x>* and *<y ...>5.0</y>* lines contain things defined in a schema; the rest of the *<soap:body>* contents comes from WSDL definitions.
- Although it is legal WSDL, *RPC/encoded* is not WS-I compliant.

7.5.4.2 RPC/Literal

The RPC/literal WSDL for this method looks almost the same as the *RPC/encoded* one (see next Listing). The use in the *binding* has changed from *encoded* to *literal*.

```
<message name="myMethodRequest">
  <part name="x" type="xsd:int"/>
  <part name="y" type="xsd:float"/>
</message>
<message name="empty"/>
<portType name="PT">
  <operation name="myMethod">
    <input message="myMethodRequest"/>
    <output message="empty"/>
  </operation>
</portType>
<binding .../>
<!-- I won't bother with the details, just assume it's RPC/literal. -->
```

Listing 9 - RPC/literal WSDL for myMethod

RPC/literal SOAP request message for *myMethod*:

```
<soap:envelope>
  <soap:body>
    <myMethod>
      <x>5</x>
      <y>5.0</y>
    </myMethod>
  </soap:body>
</soap:envelope>
```

Listing 10 - RPC/literal SOAP request message for myMethod

Let's examine now the strengths and the weak points of this choice:

Strengths

- The WSDL is still about as straightforward as it is possible for WSDL to be.
- The operation name still appears in the message.
- The type encoding info is eliminated.
- *RPC/Literal* is WS-I compliant.

Weaknesses

- You still cannot easily validate this message since only the *<x ...>5</x>* and *<y ...>5.0</y>* lines contain things defined in a schema; the rest of the *<soap:body>* contents comes from WSDL definitions.

7.5.4.3 Document/Encoded

This style is definitively not used on the Internet, it does not offer any peculiar benefit and **it is not WS-I compliant**.

7.5.4.4 Document/Literal

The WSDL for *Document/Literal* changes somewhat from the WSDL for *RPC/Literal:* the differences are highlighted in bold in the following WSDL Listing for *myMethod*.

```
<types>
```

```
  <schema>
    <element name="xElement" type="xsd:int"/>
    <element name="yElement" type="xsd:float"/>
  </schema>
</types>
<message name="myMethodRequest">
  <part name="x" element="xElement"/>
  <part name="y" element="yElement"/>
</message>
<message name="empty"/>
<portType name="PT">
  <operation name="myMethod">
    <input message="myMethodRequest"/>
    <output message="empty"/>
  </operation>
</portType>
<binding .../>
<!-- I won't bother with the details, just assume it's document/literal. -->
```

Listing 11 - Document/literal WSDL for myMethod

The new <types> section houses the definition of the XML schema that will be used in the WSDL: The needed XML schema can be explicitly expressed inside the <schema> element, or they can be included or imported from an external file like in the following examples:

```
<xsd:include schemaLocation="http://www.w3schools.com/cust.xsd"/>
```

```
<xsd:import namespace="http://www.test.com/xyz" schemaLocation="http://www.w3schools.com/cust.xsd"/>
```

Listing 12 – Include and import of an external XML schema sample

This is a Document/literal SOAP request message for *myMethod*:

```
<soap:envelope>
  <soap:body>
    <xElement>5</xElement>
    <yElement>5.0</yElement>
  </soap:body>
</soap:envelope>
```

Listing 13 - Document/literal SOAP request message for myMethod

Let's see which are the strengths and weaknesses for this choice:

Strengths

There is no type encoding info.

You can finally validate this message with any XML validator. Everything within the *<soap:body>* is defined in a schema.

***Document/Literal* is WS-I compliant**, but with restrictions (see weaknesses list).

Weaknesses

The WSDL is getting a bit more complicated. However this is a very minor weakness, since WSDL is not meant to be read by humans.

The operation name in the SOAP message is lost. Without the name, dispatching can be difficult and sometimes impossible.

WS-I only allows one child of the *<soap:body>* in a SOAP message. As you can see in the last Listing, this example's *<soap:body>* has two children.

7.5.4.5 Document/Literal wrapped

The *Document/Literal Wrapped* WSDL definition of *myMethod is reported in the following WSDL fragment.*

```
<types>
  <schema>
    <element name="myMethod">
      <complexType>
        <sequence>
          <element name="x" type="xsd:int"/>
          <element name="y" type="xsd:float"/>
        </sequence>
      </complexType>
    </element>
    <element name="myMethodResponse">
      <complexType/>
    </element>
  </schema>
</types>
<message name="myMethodRequest">
  <part name="parameters" element="myMethod"/>
</message>
<message name="empty">
```

```
   <part name="parameters" element="myMethodResponse"/>
</message>
<portType name="PT">
   <operation name="myMethod">
      <input message="myMethodRequest"/>
      <output message="empty"/>
   </operation>
</portType>
<binding .../>
<!-- I won't bother with the details, just assume it's document/literal. -->
```

Listing 14 - Document/literal wrapped WSDL for myMethod

Document/Literal Wrapped SOAP request message for *myMethod*:

```
<soap:envelope>
   <soap:body>
      <myMethod>
         <x>5</x>
         <y>5.0</y>
      </myMethod>
   </soap:body>
</soap:envelope>
```

Listing 15 - Document/literal wrapped SOAP request message for myMethod

As we can observe from Listing 14, these are the basic characteristics of the D*ocument/literal wrapped* pattern:

- the input message has just a single part;
- the part is an element;
- the element has the same name as the operation;
- the element's complex type has no attributes.

The SOAP message, instead, looks exactly like the *RPC/literal* SOAP message, but there's a subtle difference. In the *RPC/literal* SOAP message, the <*myMethod*> child of <*soap:body*> was the name of the operation. In the D*ocument/literal Wrapped* SOAP message; **the <*myMethod*> clause is the name of the wrapper element which the single input message's part refers to**. The name of the input element is the same as the name of the operation. This pattern is a sly way of putting the operation name back into the SOAP message.

Now give a look at pros and cons of this last permutation:

Strengths

- There is no type encoding info.
- Everything that appears in the <*soap:body*> is defined by the schema, so you can easily validate this message.
- Once again, you have the method name in the SOAP message.
- *Document/literal* is WS-I compliant, and the *wrapped* pattern meets the WS-I restriction that the SOAP message's <*soap:body*> has only one child.
- Each operation interface can be easily extended (i.e. by just adding a parameter).

Weaknesses

The WSDL is even more complicated.

From a WSDL point of view, there's no reason the *wrapped* pattern is tied only to *Document/literal* bindings: it could just as easily be applied to an *RPC/literal* binding. But this would be rather silly: the SOAP message would contain a <*myMethod*> element for the operation and a child <*myMethod*> element for the element name. Also, even though it's legal WSDL, an *RPC/literal* part should be a type; an element part is not WS-I compliant.

This *wrapped* style originates from *Microsoft*® and there is no official specification that defines this style.

7.5.5 SOAP response messages

Until now we have analysed the request messages, but what about response messages? By now it should be clear what the response message looks like for a D*ocument/literal* message. The contents of the <*soap:body*> are fully defined by a schema, so it's enough to look at the schema to know what the response message looks like.

```
<soap:envelope>
  <soap:body>
    <myMethodResponse/>
  </soap:body>
</soap:envelope>
```

Listing 16 - Document/literal wrapped response SOAP message for myMethod

But what is the child of the <*soap:body*> for the *RPC* style responses? The WSDL 1.1 specification [WSDL/1.1] is not clear about this point, but WS-I comes to the rescue. WS-I dictates that in the *RPC/literal* response message, the name of the child of <*soap:body*> is "... the corresponding *wsdl:operation* name suffixed with the string 'Response'." That's exactly what the conventional *wrapped* pattern's response element is called. So Listing 16 applies to the *RPC/literal* message as well as the *Document/literal wrapped* message. (Since *RPC/Encoded* is not WS-I compliant, the WS-I Basic Profile doesn't mention what an *RPC/Encoded* response looks like, but you can assume the same convention applied here applies everywhere else)

7.6 Binary data transport and representation

A SOAP message may need to be transmitted together with attachments of various sorts, ranging from images of maps to legal documents. Such data are often in some binary format, for example, most images on the Internet are transmitted using either GIF, or JPEG data formats.
The common INSPIRE SOAP framework can't avoid taking this important aspect into consideration, even more if we deem that services like the *Download Service* or the *Map Service* will provide images or large dataset as responses to client invocations.
Looking at what discovered in [EUR 23452 EN - 2008], there is no common approach about this topic in the different OGC and ORCHESTRA services specifications.
The most popular choice made by these two organisations in the single services specification documents is *SOAP with Attachments (SwA)*, or a simple URL reference of the binary resource to be attached with the response.
But if we take into consideration common discussion papers of OGC, like [06-094r1] or [08-009r1], a different approach is carried out: *MTOM*, *XOP* and potentially even *Fast Infoset*.

It could turn out to be useful to take into account all of these technical options, before choosing the most appropriate solution for the final INSPIRE SOAP framework.

The core of the problem is the following: because XML is a textual format, binary blobs must be represented using character sequences before they can be embedded in an XML document. A popular encoding that permits this embedding is known as *base64* encoding and it corresponds to the XML Schema data type *xsd:base64Binary*. A value of this type must be encoded before transmission and decoded before binding. **The encoding and decoding process is expensive and linear to the size of the binary object.**
All this process introduces an overhead that is a performance-concerning issue.
What we have just made clear is that the XML performance problems are not bound to the logical functionalities of the language, but rather to its lexical and syntactical structure.
Rationally the proposed improvement should be to find a better serialization to represent the XML logical functionalities.

7.6.1 Large XML data encoding

Taking under consideration the INSPIRE domain characteristics, it could be useful to give a particular focus to how it is possible to manage large XML data inside a SOAP message. We need to care about this issue, because INSPIRE services have often the need of handling huge GML data, that syntactically is just XML code.
For this issue, we can propose three different approaches:

- GML data can be normally encoded in the SOAP message as XML, because it doesn't need any encoding to string format. In this way there would be a linear dependence between the GML data length and the message size, and it's commonly known that XML is quite redundant as language (e.g. white spaces, similar node names and so on), so there would be no optimisation of data transport performances. It should be important that GML XML character encoding would be the same of the rest of the message;
- GML data can be normally encoded in the SOAP message as XML, like in the previous point, but a form of Binary XML (see §9.1.3) optimisation could be applied to enhance the transmission performances;
- GML data could be binary encoded before transmission (e.g. in a zip archive, or in a more specific encoding like Abstract Syntax Notation number One (*ASN1*), or similar) and sent as a binary attachment, allowing the optimisation of the transmission with MTOM+XOP. Under a performance point of view, binary encoding would be cost-effective after a GML data size threshold, which could/should be managed server-side. Anyway this choice would not be transparent to the services users, who would have to convert data before transmission and de-convert it after reception.

Each one of these options is possible and does not foreseen heavy side-effects. Probably the most natural approach would be the first one, even if it is the less performing choice. Binary XML is still not a standard and is not spread-widely used, as we will see in section 7.6.5, so, up to now, its use is not considered as

appropriate for this study. The last option could be the most balanced and average-performing, but it would imply a non transparent change for both services suppliers and consumers and this is the reason why it is not sponsored for this framework definition.

In case the INSPIRE servers would choose approaches different from the first one, the various INSPIRE services should describe which modality is supported and this information shall appear in the *getCapabilities* document of the service.

7.6.2 SOAP with Attachments

7.6.2.1 Compound SOAP Message

SOAP 1.2 part 1 ([SOAP/1.2, 1]) provides a flexible and extensible envelope for describing structured information intended for exchange between SOAP nodes.
Even though SOAP 1.2 is described in terms of XML Infoset, it is expected that [XML/1.0] can be a widely used representation for SOAP data.
The following problems can arise when using such an [XML/1.0] representation for SOAP data:

- encapsulation of binary data in the form of image files, etc. cannot be done without additional encoding/decoding of the data. The overhead of encoding binary data in a form acceptable to XML (for example using *base64*) is often significant both in terms of bytes added because of the encoding, as well as processor overhead performing the encoding/decoding.
- Encapsulation of other XML documents, as well as XML fragments, is cumbersome within a SOAP message, especially if the XML parts do not use the same character encoding.
- Although SOAP messages inherently are self-delimiting, the message delimiter can only be detected by parsing the complete message. This can imply a significant overhead in generic message processing as well as parsing and memory allocation.

In [SOAP/1.2 AF], W3C defines an abstract compound SOAP structure model to come greet with the just mentioned issues. The compound SOAP structure model is abstract in the sense that it does not define an actual means by which compound SOAP structures are represented or transmitted by a SOAP binding, but just describe its morphology and usefulness.

The components of a compound document structure are usually called *'parts'*. Accordingly, a compound SOAP structure consists of

- a primary SOAP message part;
- zero or more related secondary parts, distinct from the primary SOAP message but related to it in some manner.

Secondary parts are often informally referred to as *attachments*. While the attachment relationship is expected to be commonly used, the model makes no assumption about the nature of the semantic relationship between the primary SOAP message part and secondary parts, or between secondary parts.
It is important to note that the compound SOAP structure model does not modify or supersede the message envelope concept defined by SOAP.
Each part is identified by one or more URIs (typically one) that can be used to reference it from other parts. The URI(s) identifying a part can be of any URI scheme.

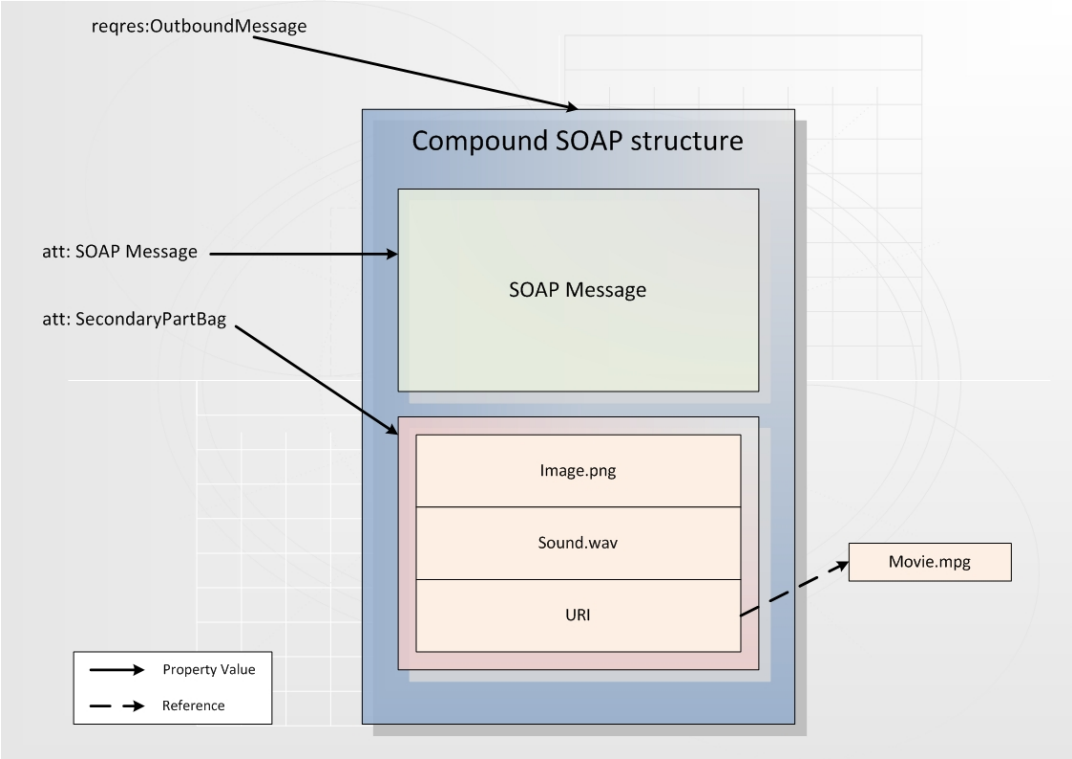

Figure 3 - Compound SOAP Message structure

The compound SOAP structure model does not require that a SOAP receiver process, dereference, or otherwise verify any secondary parts of the message. It is up to the SOAP receiver to determine, based on the processing context provided by the primary SOAP message part, which operations must be performed (if any) on the secondary part(s).

A binding that supports the transmission of compound SOAP structures MUST provide the following:

- a means by which the primary and secondary parts are made available to the receiving entity. Typically, this is achieved by transmitting all of the parts from the sender to the receiver, using binding-specified means. One option for achieving such transmission is to use an encapsulation mechanism (e.g. *DIME* or *MIME*, see §7.6.2.4 and §7.6.2.2) to prepare a single data stream containing all of the parts and to then transmit the encapsulation.
- A mechanism by which each part is identified using one (or more) URI(s). The URI scheme used MAY, but need not, to be the same for all parts.

The compound SOAP structure model is independent of the underlying protocol used for transmitting the primary SOAP message part and any of the secondary parts. That is, there is no requirement that all parts of a compound SOAP structure representation be transmitted within the same unit of the underlying protocol.
A binding that supports this feature MUST provide a means by which receivers can distinguish the primary SOAP part from the secondary parts. A SOAP receiver that supports this feature MUST process the primary SOAP message part according to the rules for processing SOAP messages (see [SOAP/1.2, 1]).
While a binding that supports this feature MAY provide mechanisms for verifying the integrity and enumerating the parts of a compound SOAP structure, this is not a requirement of this feature.

7.6.2.2 MIME Binding

[SOAP SwA] defines a binding for a SOAP 1.1 message to be carried within a MIME (Multipurpose Internet Mail Extensions) multipart/related message in such a way that the processing rules for the SOAP message are preserved. The MIME multipart mechanism for encapsulation of compound documents can be used to bundle entities related to the SOAP message such as attachments. Rules for the usage of URI references to refer to entities bundled within the MIME package are specified too.
The purpose of [SOAP SwA] is to show how to use existing facilities in SOAP and standard MIME mechanisms to carry and reference attachments: it is useful to remind that most Internet communication protocols are capable of transporting MIME encoded content.

A SOAP message package can be constructed using the multipart/related media type:

- the primary SOAP message must be carried in the root body part of the multipart/related structure. Consequently the *type* parameter of the multipart/related media header will always be equal to the *Content-Type* header for the primary SOAP message, i.e. *text/xml*.
- Referenced MIME parts must contain either a *Content-ID* MIME header, or a *Content-Location* MIME header.

It is strongly recommended that the root part contains a *Content-ID* MIME header and that in addition to the required parameters for the multipart/related media type, the start parameter (optional in [RFC2387]) is always present.
SOAP message itself is not aware that it is being encapsulated. That is, there is nothing in the primary SOAP message that indicates that the SOAP message is encapsulated.

```
MIME-Version: 1.0
Content-Type: Multipart/Related; boundary=MIME_boundary; type=text/xml;
      start="<claim061400a.xml@claiming-it.com>"
Content-Description: This is the optional message description.

--MIME_boundary
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: <claim061400a.xml@claiming-it.com>

<?xml version='1.0' ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
      ..
      <theSignedForm href="cid:claim061400a.tiff@claiming-it.com"/>
      ..
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

--MIME_boundary
Content-Type: image/tiff
Content-Transfer-Encoding: binary
Content-ID: <claim061400a.tiff@claiming-it.com>
...binary TIFF image...
--MIME_boundary--
```

Listing 17 - MIME Message sample

Both the header entries and body of the primary SOAP message may need to refer to other entities in the message package.
The data encoding rules given in SOAP specification allow the value of an accessor to be given by reference, i.e. as a resource referenced by a URI given as the value of a *href* attribute. We observe that the SOAP encoding schema allows the value of a *href* attribute to be any URI reference and the attribute may therefore be used to reference not just XML fragments within a SOAP message, but any kind of resource.

The resolution process operates in two steps:

1. to convert all URI references to absolute references;
2. to resolve the absolute references.

Every MIME part in the multipart/related structure that constitutes a SOAP message package has at least one absolute URI label.
[SOAP SwA] defines an extension to the transport binding mechanisms defined in SOAP 1.1. The packaging of a SOAP message in the root part of a multipart/related MIME structure along with other contents is to be viewed as a specific method for carrying SOAP messages in any protocol capable of transferring MIME-encoded content.
A SOAP processor must treat the SOAP message in the root part as the message to be processed.
The basic approach to carry multipart MIME structure in an HTTP message in this specification is to confine MIME-encoded content to the MIME parts and use the multipart media type header at the HTTP level as a native HTTP header.

- The rules for forming an HTTP message containing a SOAP message package are the following: the "*Content-Type: Multipart/Related*" MIME header must appear as an HTTP header.
- No other headers with semantics defined by MIME specifications (such as *Content-Transfer-Encoding*) are permitted to appear as HTTP headers. Specifically, the *"MIME-Version: 1.0"* header must not appear as an HTTP header. Note that HTTP itself uses many MIME-like headers with semantics defined by HTTP 1.1. These may, of course, appear freely.

The MIME parts containing the SOAP message and the attachments constitute the HTTP entity body.

Between the less positive aspect of this encoding-based approach, as stated in [Paranjpe, 2003], it is useful to remind that MIME based data attachments are stored in a linear fashion. To search for a given attachment, you have to traverse through the entire set of messages, because boundaries between different binary attachments are not defined in the header.

7.6.2.3 WS-I Attachment Profile 1.0

WS-I wrote a document ([WS-I AP/1.0]) adding a series of rules to try to standardise the Messages with Attachments (SwA) mechanism.
As stated before, SwA defines a MIME multipart/related structure for packaging attachments with SOAP messages. [WS-I AP/1.0] complements the WS-I Basic Profile 1.1 [WS-I BP/1.1] to add support for conveying interoperable SwA-based attachments with SOAP messages.
We report here after the most important indications that WS-I gives to add this new feature to WS-I Basis Profile 1.1.

SOAP Messages with Attachments defines a MIME multipart/related structure for packaging SOAP envelope with attachments. The Profile mandates the use of that structure, and places the following constraints on its use:

R2931 The entity body of the root part of multipart/related MESSAGE MUST be a *soap:Envelope*.

R2945 The Content-Type HTTP header field-value in a MESSAGE MUST be either *"multipart/related"* or *"text/xml"*.

R2932 If the Content-Type HTTP header field-value in a MESSAGE has a media-type of *"multipart/related"* then the *Content-Type* HTTP header field-value in that message MUST have the type parameter with a value of *"text/xml"*.

Encoding of Root Part

- **R2915** The entity body of the root part of a `multipart/related` message MUST be serialized using either *UTF-8* or *UTF-16* character encoding.

- **R2916** Non-root parts of a `multipart/related` message MAY use any character encoding.

Messages with No Attachments

If a receiver expects zero or more attachments in a message, the sender of that message can use the *text/xml* media type for a message that has no attachments.

Ordering of MIME Parts

It is possible that intermediaries might reorder the parts in a multipart/related message. Hence semantics should be neither given to, nor implied by the ordering of parts in a message.

R2921 a receiver MUST NOT infer any semantics from the ordering of non-root MIME parts in a message.

R2929 a message MAY have its MIME parts in any order provided that the identity of the root part is maintained.

A receiver must not assume that the order of *mime:part* elements specified in a WSDL description is the same as the order of MIME parts in the message. The order of MIME parts specified in a WSDL description must be considered independent of the order of MIME parts in the message.

Position of root part

If the start parameter is present, then the value of the start parameter is the *content-ID* of the root part of the message. In the absence of a start parameter, the root part is the first body part in the package, as defined by RFC 2387 Section 3.2.

- **R2922** If the *Content-Type* HTTP header field-value in a message does not have a start parameter; a RECEIVER MUST treat the first body part of the MIME package as the root part.

The Profile places the following constraints on its use:

Referencing Attachments from the SOAP Envelope

One of the advantages of having attachments is the ability to include data in a separate MIME part and refer to it from the SOAP envelope contained in the root part of the same MIME package.

This Profile defines a schema type *ref:swaRef* which can be used in a WSDL description to define a message part. When a message part is described using the *ref:swaRef* type, in its instance document, the URI points to an attachment in the same MIME package.

The XML Schema for the type used to refer to attachments from the SOAP envelope is:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema targetNamespace="http://ws-i.org/profiles/basic/1.1/xsd"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:simpleType name="swaRef">
      <xsd:restriction base="xsd:anyURI" />
   </xsd:simpleType>
</xsd:schema>
```

Please note that there is no way in a WSDL 1.1 description to correlate an attachment reference (defined using *swaRef*) with an attachment (defined using a *wsdl:part* bound to *mime:content*). As a best practice, the Profile recommends that when *ref:swaRef* is used, the corresponding attachment should not be described and vice versa:

R2940 In a DESCRIPTION, a *wsdl:part* defined with the *ref:swaRef* schema type SHOULD only be bound to a *soapbind:body*, or a *soapbind:header* in a MIME binding.

R2928 In an ENVELOPE, a URI reference that is typed using the *ref:swaRef* schema type MUST resolve to a MIME part in the same message as the envelope.

The *swaRef* type can be used to represent a reference to an attachment either as an element (as shown in the example below) or an attribute. There is no preferred approach.

7.6.2.4 DIME

SwA leans on some kind of standard that is able to house in the same message different kind of independent data.
As we have seen MIME is the most used format, but another standard was proposed for the same purpose: DIME.
Quoting Wikipedia, **Direct Internet Message Encapsulation (DIME) is a *Microsoft*-proposed internet standard for the transfer of binary and other encapsulated data over SOAP**.
According to the IETF web site, the standard has been withdrawn and never made RFC status. However, *Microsoft* currently does recommend DIME for transmitting files via Web services.
The first version was submitted to the IETF in November 2001; the last update was submitted in June 2002. By December 2003, DIME had lost out, in competition with Message Transmission Optimization Mechanism and SOAP with Attachments and Microsoft now describes it as "superseded by the SOAP Message Transmission Optimization Mechanism (MTOM) specification" [Messaging Specifications Index Page, Microsoft, retrieved on 31/10/2006].

The standard was supposed to be a simplified version of MIME: unlike MIME, DIME converts both text and binary portions into a single binary message. This storing of the binary portion in its raw binary form saves the DIME parser the overhead and complexity of reversing a text to binary encoding, as is required with MIME. Indeed, the simple length-delimited nature of the DIME parts, called records, allows the DIME parser to split the records without looking inside, past an initial binary header.

7.6.3 XML Optimisation Package (XOP)

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

When parsing a XOP package, the binary data can be made available directly to applications, or, if appropriate, the *base64* binary character representation can be computed from the binary data.

It is necessary to have a one to one correspondence between XML Infosets and XOP Packages; therefore, the conceptual representation of such binary data is as if it were *base64*-encoded.

```
<soap:Envelope
  xmlns:soap='http://www.w3.org/2003/05/soap-envelope'
  xmlns:xmlmime='http://www.w3.org/2004/11/xmlmime'>
   <soap:Body>
      <m:data xmlns:m='http://example.org/stuff'>
         <m:photo xmlmime:contentType='image/png'>/aWKKapGGyQ=</m:photo>
         <m:sig xmlmime:contentType='application/pkcs7-signature'>Faa7vROi2VQ=</m:sig>
      </m:data>
  </soap:Body>
</soap:Envelope>
```

Listing 18 - Example of XML Infoset prior to XOP processing

```
MIME-Version: 1.0
Content-Type: Multipart/Related;boundary=MIME_boundary;
  type="application/xop+xml";
  start="<mymessage.xml@example.org>";
  startinfo="application/soap+xml; action=\"ProcessData\""
Content-Description: A SOAP message with my pic and sig in it

--MIME_boundary
Content-Type: application/xop+xml;
  charset=UTF-8;
  type="application/soap+xml; action=\"ProcessData\""
Content-Transfer-Encoding: 8bit
Content-ID: <mymessage.xml@example.org>

<soap:Envelope
  xmlns:soap='http://www.w3.org/2003/05/soap-envelope'
  xmlns:xmlmime='http://www.w3.org/2004/11/xmlmime'>
  <soap:Body>
    <m:data xmlns:m='http://example.org/stuff'>
      <m:photo xmlmime:contentType='image/png'>
        <xop:Include
          xmlns:xop='http://www.w3.org/2004/08/xop/include'
          href='cid:http://example.org/me.png'/>
      </m:photo>
      <m:sig xmlmime:contentType='application/pkcs7-signature'>
```

```
          <xop:Include
            xmlns:xop='http://www.w3.org/2004/08/xop/include'
            href='cid:http://example.org/my.hsh'/>
        </m:sig>
      </m:data>
    </soap:Body>
</soap:Envelope>

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

// binary octets for png

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

// binary octets for signature

--MIME_boundary—
```

Listing 19 - XML Infoset serialized as a XOP package

The character children of elements to be optimized are removed and replaced with an item named *xop:Include*. The *xop:Include* item contains an attribute with a link to the part of the XOP Package that carries a binary representation of the data removed from the original item.
The Infoset used as input to XOP processing MUST NOT contain any element information item with a namespace name property of *"http://www.w3.org/2004/08/xop/include"* and a local name property of *Include*. Infosets containing such items cannot be serialized using XOP. This is because during Infoset reconstruction a processor is unable to differentiate between *xop:Include* element information items inserted during XOP package construction and those that were part of the original Infoset.
The *xop:Include* element information item has a mandatory *href* attribute, housing a normalized value which is a representation of a URI referencing the part of the package containing the data logically included by the owner element.

7.6.4 Message Transmission Optimisation Mechanism (MTOM)

Up to now in the previous two sections "SOAP with Attachments" and "XML Optimisation Package (XOP)" we focused on an optimisation of the SOAP message structure t transport binary data; now we are going to investigate an improvement focalised on the transmission of SOAP messages on the underlying protocol.

As stated by [MTOM], the **HTTP SOAP Transmission Optimization Feature** (*http://www.w3.org/2004/08/soap/features/abstract-optimization*) is intended to enhance the SOAP HTTP binding.

Optimization is available only for element content that is in a canonical lexical representation of the *xs:base64Binary* data type.

MTOM implementations typically optimize by transmitting a compact representation of the value in place of the less compact character sequence. At the receiver, the character form can be reconstructed if necessary.

7.6.4.1 Sending a message

When sending a SOAP Message, if the Abstract Transmission Optimization Feature is used in combination with the SOAP *Request-Response* Message Exchange Pattern, or the SOAP *Response* Message Exchange Pattern, the *http://www.w3.org/2003/05/soap/mep/OutboundMessage* property is the Infoset of the SOAP Message to be sent.

As told before, the purpose of the Abstract SOAP Transmission Optimization Feature is to optimize the transmission of *base64* encoded data. To be optimized, the characters comprising the children of an element MUST be in the canonical form of *xs:base64Binary* and MUST NOT contain any whitespace characters; non-canonical representations MUST NOT be optimized by implementations of this feature.

7.6.4.2 Receiving a message

When receiving a SOAP message optimized using an implementation of the Abstract SOAP Transmission Optimization Feature, a SOAP node SHOULD generate a fault if it does not support the implementation used.

The receiving node MUST reconstruct an Envelope Infoset from the optimized SOAP message and then the receiving node MUST perform SOAP processing on the reconstructed Infoset. In all cases, the received Infoset MUST be exactly the same as that transmitted by the sender.

Implementations are free to reconstruct only those portions actually needed for processing, or to present information from the message in a form convenient for efficient processing.

The Infoset contained in the "*http://www.w3.org/2003/05/soap/mep/InboundMessage*" property is the Infoset of the reconstructed SOAP Envelope.

The **Optimized MIME Multipart/Related Serialization** expands upon the Abstract SOAP Transmission Optimization Feature by describing parts of an implementation of this feature using the XML-binary Optimized Packaging format as its basis.

The SOAP envelope Infoset is transmitted as a MIME Multipart/Related XOP Package.

7.6.5 Binary XML formats

More and more companies are considering or making the move to XML as the format they rely on for transmitting data to and from applications and web services. XML is a flexible, cross-platform, robust, commonly accepted standard that has spawned a rich ecosystem of tools, utilities and applications that leverage its power and extensibility. Yet, even with all these strengths, XML adoption has been hampered because its sheer size and verbosity clogs networks overwhelm smaller devices and can slow data transmission to a crawl. Parsing XML can also consume a great deal of CPU time which reduces throughput and scalability, increasing the cost/performance ratio of computing resources. Because the cost of adding additional bandwidth or CPU cycles to accommodate XML is often prohibitively high, many companies are simply avoiding the move to XML and are missing out on all it has to offer.

For this set of reasons, different specifications have been made to overtake this hurdle, passing to a binary representation of XML data.

7.6.5.1 Fast Infoset

As wrote in [Yang, 2007], an XML Information Set (Infoset) is an abstract model of the information stored in an XML document. The XML Infoset is a W3C specification that specifies the result of parsing an XML document and identifies various Infoset components, called information items and properties. More and more technologies, such as SOAP 1.2 and MTOM/XOP, are defined in terms of the XML Infoset, because most common XML use cases are really concerned with the information stored within an XML document rather than its presentation with a specific encoding.

The **Fast Infoset (FI) specification provides a representation of an instance of Infoset using *ASN.1* binary encodings**. It can serve as an alternative to W3C's Infoset syntax. Fast Infoset documents retain the hierarchical structure described by the corresponding XML Infoset. Depending on which features you select, a Fast Infoset document can be self-contained or not. Self-contained Fast Infoset documents can be converted to and from traditional XML documents without loss of information, at least with respect to the information items and properties defined in the XML Information Set. You therefore can consider these representations as equivalent to XML documents for the corresponding XML Infosets.

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". 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. [08-009r1]

An optimization technique available in Fast Infoset is the use of the so-called "encoding algorithms". 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.
FI aims to optimize both document size and processing performance in a lossless fashion: while the original formatting is lost, no information is lost in the conversion from XML to FI and back to XML.
Fast Infoset is not the same thing as MTOM+XOP, but it can solve the same problem, i.e. not having to encode binary contents in *base64*.
Fast Infoset is a binary encoding of the XML information set, while MTOM and XOP is a transformation of an Infoset into another Infoset where *base64 clobs* are instead represented as *blobs*, stored as binary attachments.

Fast Infoset is more efficient than MTOM at handling *blobs* when the binary content is small and the overhead of the MIME part headers is large. Fast Infoset can encode *blobs* for attribute values and also support the binary encoding of integer and real numbers.

7.6.5.1.1 Fast Infoset Documents as Attachments

A SOAP message package with attachments is usually constructed using the MIME `multipart/related` type. This is typically accomplished by employing a MIME binding in the WSDL file. For example the following snippet, taken from [Pericas, 2005], shows how to bind a hypothetical *addPhoto* operation so that it returns the *status* part as a MIME attachment.

```
<wsdl:operation name="addPhoto">
  <wsdl:input>
    ...
  </wsdl:input>

  <wsdl:output>
    <mime:multipartRelated>
      <mime:part>
        <soap:body use="literal"/>
      </mime:part>
      <mime:part>
        <!-- Use application/fastInfoset to indicate an FI attachment -->
        <mime:content part="status" type="application/fastInfoset"/>
      </mime:part>
    </mime:multipartRelated>
  </wsdl:output>
</wsdl:operation>
```

Listing 20 – Sample WSDL for an operation returning data as FastInfoset MIME attachment

This binding indicates not only that the *status* part is bound to a MIME part but also that the type of this attachment is *application/fastInfoset*, that is, a Fast Infoset document.

7.6.5.2 Efficient XML

AgileDelta's *Efficient XML* tries to specifically solve the XML transmission and processing problem. *Efficient XML* aims to reduce the size of the XML and to increase the processing and transmission speed of XML across existing networks using a compact **binary XML format**.

Efficient XML is a general purpose interchange format that was designed to optimize performance while reducing bandwidth, battery life, processing power and memory requirements. **It is the only format currently being tested that supports all of the features specified by the minimum binary XML requirements defined by the W3C XBC group** *XML Binary Characterization* [XBC Charact].

The encoding is schema "informed", meaning that it can leverage available schema information to improve compactness and performance, but does not depend on accurate, complete or current schemas to work. It works very effectively with partial schemas or no schemas at all. It also supports arbitrary schema extensions and deviations and allows dynamic schema negotiation, discovery and acquisition.

Efficient XML achieves broad generality, flexibility and performance by unifying concepts from formal language theory and information theory into a single algorithm. The algorithm uses a grammar to determine what is likely to occur in an XML document and encodes the most likely alternatives in fewer bits. The fully generalized algorithm works for any language that can be described by a grammar (e.g., XML, Java, HTTP, etc.); however, *Efficient XML* is optimized specifically for XML languages. The built-in *Efficient XML* grammar accepts any XML document or XML fragment and may be augmented with productions derived from XML Schemas, RelaxNG schemas, DTDs, or other sources of information about what is likely to occur in a set of XML documents. The *Efficient XML* encoder uses the grammar to map a stream of XML information items onto a smaller lower entropy stream of tokens. The encoder then encodes the stream of tokens using a Huffman tree derived from the grammar or, if additional compression is desired, passes the stream of tokens to a more sophisticated XML compression algorithm that replaces frequently occurring token patterns to further reduce size. When schemas are used, *Efficient XML* also supports a user-customizable set of data-type CODECs for efficiently encoding typed values and provides typed streaming APIs for efficiently accessing typed values.

The binary form of *Efficient XML* is very compact. It is competitive with hand-optimized formats and is consistently smaller than both *ASN.1 PER* and *gzipped XML*. Even on very large, repetitive documents where *gzip* works best, it is not uncommon for *Efficient XML* to be 2-5 times smaller than *gzipped XML*.

Production implementations of *Efficient XML* have been integrated into a broad range of platforms, including mass market mobile phones, PDAs, application servers, web servers, high-volume message routers, pub-sub systems, vehicles, aircraft, and satellite broadcast systems. High quality, commercial implementations are available for UNIX, MS-Windows and a wide variety of mobile devices running both Java and Microsoft.NET.

7.6.6 Comparative analysis

In this paragraph we are going to report a set of tests to compare the performance of the different technical choices for binary data encoding, beginning with a test cited in [Faster M/X] to compare performances offered by normal XML encoding, MTOM+XOP and Fast Infoset, then we will pass to a compare between Fast Infoset, gzipped XML and Efficient XML, taken from [AGILE P&F].

7.6.6.1 XML vs. MTOM+XOP vs. Fast Infoset

Trying to summarise the previous paragraphs, MTOM (SOAP Message Transmission Optimization Mechanism) is a W3C Recommendation for packaging binary data within SOAP messages in a way that avoids *base64* encoding. It depends on XOP (XML-binary Optimized Packaging), which provides a mechanism for embedding binary data in XML Infosets. XOP supports only the embedding of binary

data for chunks of character information items. Fast Infoset, on the other hand, describes a binary encoding of the XML Information Set and allows for the direct embedding of binary data (as either chunks of character information items, or the property of attribute information items). In this sense, Fast Infoset does what MTOM/XOP can do plus something more.
So which technology is better in terms of their impact on web service performance?

For each of the following settings, Figure 4 and Figure 5 present bar charts that summarize the average response time (ART) of a sample web service, developed with JAX-WS RI, for three different sets of data:

1. A standard JAX-WS web service (Standard)
2. Both client and service MTOM-enabled (MTOM/XOP)
3. Fast Infoset enabled (FI)

Figure 4 shows the results of invoking an upload method. The data size format (X + Y) indicates the size of the two binary data to be uploaded, one smaller and one larger, sent with the request. The standard deviation is marked on the bars.

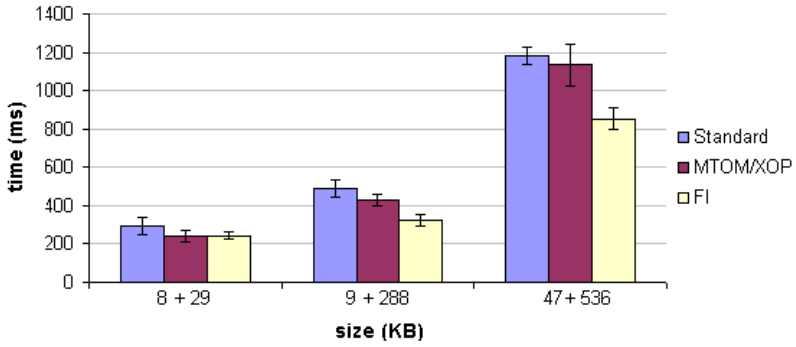

Figure 4 - First performance test results (two files upload)

Figure 5, instead, is for a download method; on the *x* axis there's the size of the binary data to download and on the bars the standard deviation is marked.

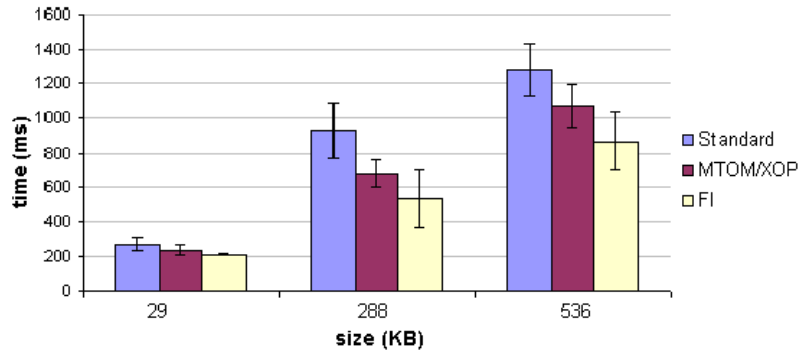

Figure 5 - Second performance test results (one file download)

As the empirical data clearly demonstrates, web services—especially those that process and transport binary data—clearly benefit from both MTOM and Fast Infoset. However, the reference implementation of JAX-WS 2.0 Fast Infoset-enabled web services consistently perform better than MTOM-enabled ones.

Fast Infoset is on its way to being widely supported in various platforms and frameworks such as Microsoft .NET and .NET CF, Sun GlassFish, BEA WebLogic, IBM SDK for Java 6.0, and TMax Soft JEUS 6, as well as in the Linux, Solaris, and Win32 operating systems.

7.6.6.2 XML vs. Fast Infoset vs. gzipped XML vs. Efficient XML

The World Wide Web Consortium (W3C) conducted extensive benchmarking of binary XML technologies and found Efficient XML consistently achieved the best compression for every test group, every use case and every class of application. At the same time, they found that AgileDelta's Efficient XML was one of the fastest binary XML implementations of any kind. In the best real-world cases reported by customers, Efficient XML was over 450 times smaller than standard XML and 42 times smaller than GZip.

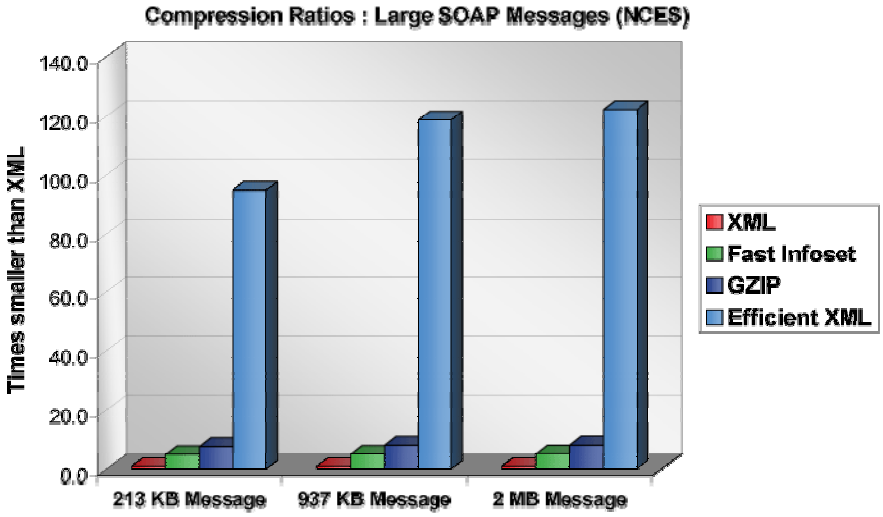

Figure 6 - Compression ratios large XML messages comparative analysis

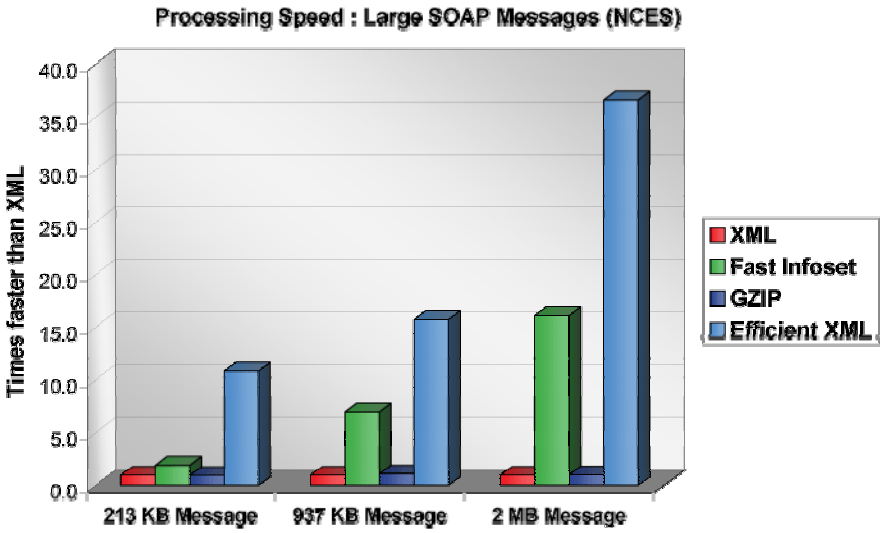

Figure 7 – Processing speed large messages comparative analysis

The graphs above show how many times smaller and faster Efficient XML makes a set of large, real-world SOAP web-service messages compared side-by-side with XML, GZIP and Fast Infoset. Taller bars represent smaller messages and faster processing. So in these cases, Efficient XML is up to 122 times smaller than XML, 14 times smaller than GZIP and 23 times smaller than Fast Infoset. At the same time, Efficient XML processes these messages up to 35 times faster than XML, 36 times faster than GZIP and 5 times faster than Fast Infoset. Efficient XML is the only technology of any kind that

achieves this level of bandwidth and processing efficiency and if processing speed is your primary goal, Efficient XML can operate at over twice these speeds when optimized for speed over size.

As you can see, Efficient XML is considerably smaller and faster than XML. In addition, applications can read and write Efficient XML directly using popular XML APIs instead of serializing, compressing, decompressing and parsing XML.
Efficient XML also works for any size messages, including very small messages typical of web services, mobile devices, financial data and location-based services. Traditional data compression techniques have very little effect on these types of messages and can actually increase their size while reducing processing speeds. Efficient XML makes them both smaller and faster.

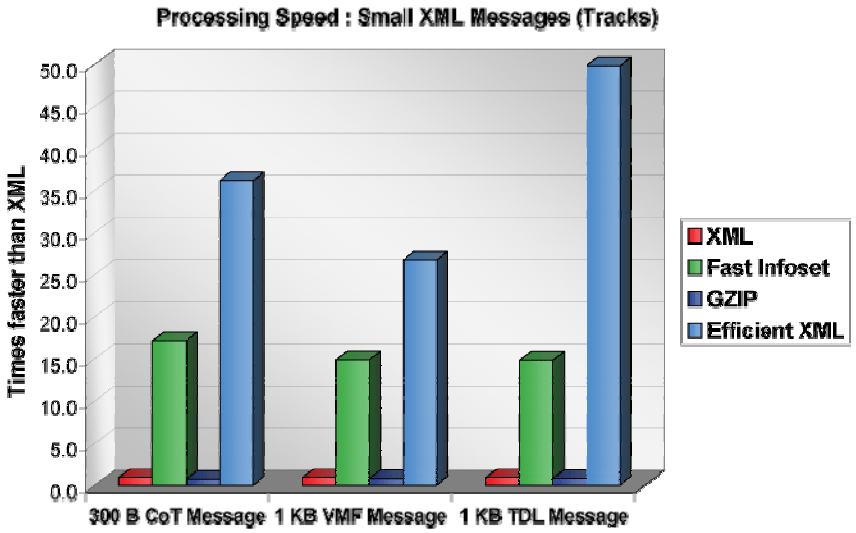

Figure 8 - Processing speed small XML messages comparative analysis

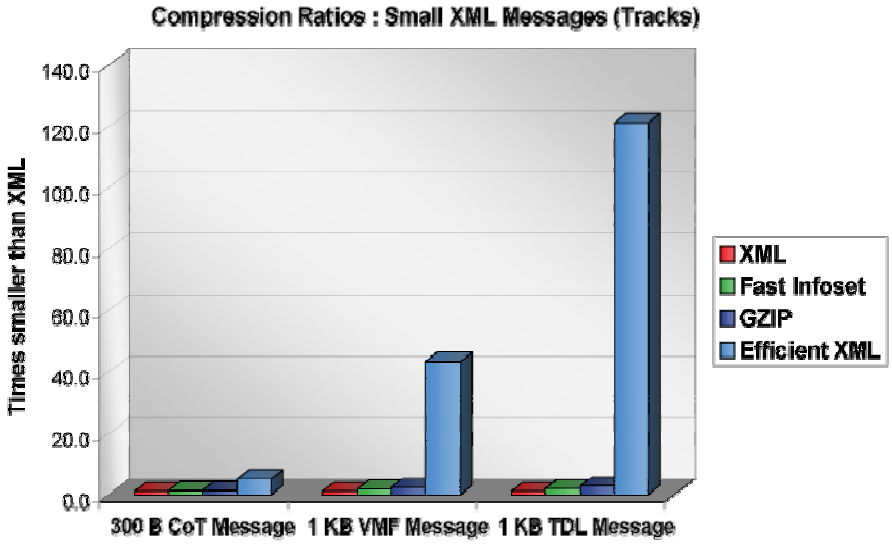

Figure 9 - Compression ratios small XML messages comparative analysis

The graphs above show how many times faster and smaller Efficient XML makes small XML documents used for various geo-location applications compared side-by-side with XML, Fast Infoset and GZIP. In these cases, Efficient XML is up to 121 times smaller than XML, 42 times smaller than GZIP and 64 times smaller than Fast Infoset. At the same time, Efficient XML processing speeds are up to 50 times faster than XML, 60 times faster than GZIP and 3 times faster than Fast Infoset.

7.6.6.3 Considerations

After these brief tests and the previous theoretical explanations, it is out of any doubt that assuming any kind of binary data management optimisation is useful to improve the network services performances. The big choice to be done is between a textual XML representation and a binary one.

In an absolute context, we can cite the study performed by W3C in [Eff XML – IMN], while for the whole analysis we recommend to read in-depth the full article.
"Considering XML as an encoding of the XML Information Set, the results demonstrate that it is possible for an alternate format to achieve significant improvements in both Processing Efficiency and Compactness simultaneously. Therefore, even though sometimes one property can be traded for the other, as illustrated by comparing the Neither and Document classes, this does not mean that XML itself is optimal in either property. […] Based on examining the format specifications and the results of these measurements, the group selected Efficient XML as the basis for development of the Efficient XML Interchange format."

8. Annex B - Comparative analysis with OGC services

If we wish to make a comparative analysis between the framework proposed in chapter 6 and the one proposed by OGC, the first consideration to be done is that the INSPIRE framework was born with the idea to be based on SOAP, while OGC web services infrastructure has been thought when SOAP was not a standard yet and so it layers mostly on HTTP GET and POST actions.
This substantial divergence brings to a different choice also about the interoperability standard: OGC is not WS-I compliant, while the just proposed framework follows WS-I Basic Profile 1.2 and could evolve in time, in order to follow WS-I profile 2.0 dictates.
This choice should enhance the interoperability of INSPIRE services, making them available for the largest number of users and applications.

Under a more technical point of view, as demonstrated in [EUR 23452 EN - 2008], OGC does not take an univocal decision under the main aspects of a hypothetical SOAP framework. There is not a common choice for the data encoding topic, neither for the binary data transport and representation, while INSPIRE, with this document, clearly defines the ways to be followed. Document/literal wrapped style and MTOM+XOP on MIME will be the used standards. Anyway these choices do not differ too much from the several ones proposed by OGC: OGC in some specifications left a free choice between *Document/literal* and *RPC/literal*, but the former is the most popular suggestion. Even for the binary data representation and transport mechanism MTOM+XOP solution was between the proposed choices by OGC.

OGC does not take over the SOAP header theme at all, while in this document the possibility to exploit this kind of extensions has been analysed and has been proposed as a solution for different kind of problems of the INSPIRE domain, like security, artefact signatures, download checksums and multilingualism.

If we consider the exception reporting topic, instead, INSPIRE SOAP framework followed precisely the OGC specification about how to represent the raised exceptions in SOAP messages, approving the whole proposed XML message structure.

To conclude this comparative analysis, it could be useful to summarise in a table all the protocol and standard used by the two different frameworks, together with the specific version.

INSPIRE SOAP FRAMEWORK	OGC WEB-SERVICES
XML 1.0	XML 1.0
HTTP 1.1	HTTP 1.1
SOAP 1.1	SOAP 1.2
WSDL 1.1	WSDL 1.1
MTOM 1.0 + XOP 1.0 + MIME 1.0	SwA or MTOM+XOP+MIME or FastInfoset
Document/literal wrapped	Mainly Document/literal
WS-I 1.2 *(next 2.0)* compliance	No WS-I compliance

Table 4 - Protocol and standard versions comparison

9. Annex C - Critical analysis

The proposed INSPIRE SOAP framework has been designed starting from the vision of the whole INSPIRE domain, based on the service description and the geo-spatial realm general knowledge. Beside these preliminary inputs, all the OGC and ORCHESTRA relevant documentation about studied and implemented web services and about known technical issues have been analysed and investigated to move towards the best direction.
The proposed technical solutions have all been evaluated taking into consideration their standard relevancy, because a high level of interoperability is one of the required key points for INSPIRE.

9.1 Future versions of existing standards

9.1.1 WS-I Profiles

Under the theme of interoperability, INSPIRE will keep an eye open to the fore-coming WS-I Basic Profile 2.0 [WS-I BP/2.0] that is going to be refined and approved by the Drafting Team in the next months and that will bring technological enhancements like the use of SOAP 1.2.
In fact, this next version of SOAP is going to be the commonly used version and will refine the previous specification as much as to become the referenced edition by the just cited next to come WS-I Basic Profile 2.0. Anyway, up to this first release of the INSPIRE SOAP framework, SOAP 1.1 will be used, even if, as already reported in paragraph 5.1, OGC explicitly recommends SOAP 1.2.

9.1.2 WSDL

A similar analysis to the one performed for SOAP version can be applied to WSDL: the latest version of this language (*2.0*) has to be considered more evolved and with more prospective in the next future, but up to now the commonly used version and the one supported by WS-I profiles is the previous one: *1.1* and will be adopted at least for these first releases of the framework.

It could be useful in this section of the document to try to briefly summarise the main changes between the two versions of the language.
Drawing on [Dhesiaseelan, 2004], we need to remember that WSDL 1.2 was renamed WSDL 2.0 because of its substantial differences from WSDL 1.1. Some of these changes include:

- Adding further semantics to the description language. This is one of the reasons for making *targetNamespace* a required attribute of the definitions element in WSDL 2.0.
- Removal of message constructs. These are specified using the XML schema type system in the *types* element.
- No support for operator overloading.
- *PortTypes* renamed to interfaces. Support for interface inheritance is achieved by using the *extends* attribute in the interface element.
- Ports renamed to endpoints.

9.1.3 Binary XML

Moreover, taking into account the analysis performed in section 7.6.6 the choice of using MTOM+XOP is not going to be the most performing one, but to move to binary XML domain could be a risk looking at the actual market and technological landscape, as this new kind of XML encoding has not become a standard yet and we cannot be sure that this eventuality will become a reality.
Potentially, if during the time the INSPIRE SOAP framework is getting defined, the market will follow new directions, or Binary XML will become a standard, these options will be taken into account for the next framework releases.

9.2 Data encoding issues

Under the data encoding point of view, the way to be followed has been to chase the most popular standards and to be WS-I compliant.
The preference for *Document/literal wrapped* encoding follows this logic, but, as it will be explained in the following technical paragraphs, this encoding cannot be used if overloaded methods are present.

9.2.1 Problem with Document/literal wrapped

Even if the Document/literal wrapped choice looks like the best one, there are still cases where it's better to use another style, like if you have overloaded operations: in such a case the use of the *Document/literal wrapped* style is forbidden. Here after follows a simple demonstration.
We can imagine that, along with the method we have been using along paragraphs 7.5.4.x, we have the additional method specified in the following Listing:

```
public void myMethod(int x, float y);
public void myMethod(int x);
```

Listing 22 - Problem methods for Document/literal wrapped style

W3C states that WSDL 1.1 allows overloaded operations. But when we add the *wrapped* pattern to WSDL, we require an element to have the same name as the operation, and we cannot have two elements with the same name in XML. So we must use the *Document/literal non-wrapped* style or one of the *RPC* styles.
Although the WSDL 1.1 specification allows overloaded operations, the **WS-I Basic Profile does not permit it** and **WSDL 2.0 neither**.
According to XML schema each type (and ultimately document) can only have one definition - so overloading makes no sense for D*ocument/literal - wrapped or not.*

9.2.2 Reasons to use RPC/Encoded

The primary reason to prefer the *RPC/Encoded* style is for data graphs (non tree-like structures). Imagine that we have a binary tree whose nodes are defined in the following Listing:

```
<complexType name="Node">
  <sequence>
    <element name="name" type="xsd:string"/>
    <element name="left" type="Node" xsd:nillable="true"/>
    <element name="right" type="Node" xsd:nillable="true"/>
  </sequence>
</complexType>
```

Listing 23 - Binary tree node schema

With this node definition, we could construct a tree whose root node -- *A* -- points to node *B* through both its left and right links (see Figure 10).

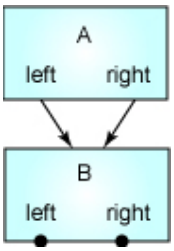

Figure 10 - Encoded tree

The standard way to send data graphs is to use the *href* tag, which is part of the *Encoded* style:

```
<A>
  <name>A</name>
  <left href="12345"/>
  <right href="12345"/>
</A>
<B id="12345">
  <name>B</name>
  <left xsi:nil="true"/>
  <right xsi:nil="true"/>
</B>
```

Listing 24 - The RPC/encoded binary tree

Using SOAP 1.1, under any *Literal* style, the *href* attribute can raise problems due to the serialization based on XML 1.0, so its use is not recommended and without *href*, the graph linkage would be lost (see following Listing and Figure).
We would still have a root node, *A*, which points to a node *B* to the left and another node *B* to the right. These *B* nodes are equal, but they are not the same node. **Data would have been duplicated instead of being referenced twice**.

```
<A>
  <name>A</name>
  <left>
    <name>B</name>
    <left xsi:nil="true"/>
```

```
    <right xsi:nil="true"/>
  </left>
  <right>
    <name>B</name>
    <left xsi:nil="true"/>
    <right xsi:nil="true"/>
  </right>
</A>
```

Listing 25 - The literal binary tree

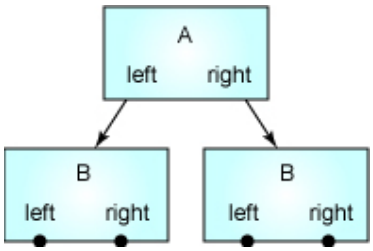

Figure 11 - Literal tree

There are various ways we can do graphs in *literal* styles, but **there are no standard ways**; so anything we might do would probably take the risk not to interoperate with the service on the other end of the wire.

The reason for the just exposed issue is only related with SOAP 1.1 because the serialization process is based on XML 1.0 serialization (which uses *href* and *id* to encode links), while SOAP 1.2 (not allowed by actual WS-I profile as seen in previous paragraphs) is based on serialization of information sets (main changes listed at http://www.w3.org/2003/06/soap11-soap12.html). In particular the corresponding relevant change between the two versions is the namespace of the *href* used for the serialization, SOAP 1.2 separates it from the default one as stated by SOAP 1.2 primer http://www.w3.org/2003/06/soap11-soap12.html: "The *href* attribute in SOAP 1.1 (of type *xs:anyURI*) is called *enc:ref* in SOAP 1.2 and is of type *IDREF*.".
This change removes the potential issues related with SOAP messages serialization (e.g. validation) of document/literal messages using data graphs.

9.2.3 Impact on the INSPIRE Domain

Both problems described in the previous sections need to be taken into account before taking a decision for the INSPIRE SOAP framework.
Regarding the method overloading issue, that could affect the *Document/Literal wrapped* choice, it should be noticed that in none of the specification examined in [EUR 23452 EN - 2008] the possibility of overloading methods is treated or discussed. In any case, naive solution is always possible, based on the possibility to change the name of the overloaded methods.
Moreover, the WSDL 2.0 specification, which could be taken under consideration for the next versions of the INSPIRE SOAP framework, does not allow overloaded methods in a service declaration. In conclusion, the problem highlighted in §9.2.1 does not affect the Document/Literal wrapped choice for the Inspire SOAP framework.

Regarding the use of RPC/Encoded style for solving datagraph-like issues, it should be noticed that the INSPIRE SOAP framework selects the SOAP version in accordance with WS-I, which is currently SOAP 1.1 (and not SOAP 1.2 which would solve the issue). Fortunately all the circumstances where the *href* attribute is used by OGC services (e.g. GML streams using polygons or metadata records) are using an *href* with a specialized namespace and not the default one as used in Listing 13 for exemplifying the encoding problem; therefore the data graph problems does not occur in the INSPIRE infrastructure, even in the case it relies on SOAP 1.1.

So the only side effect deriving from this choice is the complexity of the final services WSDL, but this is definitely a minor issue.

10. Annex E – Terms, Definitions and Abbreviations

ACRONYM		*EXPLANATION*
BLOBS		Binary Large OBjectS
CLOBS		Character Large OBjectS
DT		Draft Team
GML		Geography Mark-up Language
IEC		International Electro-technical Commission
NSDT		Network Services Drafting Team
QoS		Quality Of Service
SwA		SOAP With Attachments
WG		Working Group

Table 5 – Document Abbreviations Table

11. Annex F – Inspire IR Reference

The listed documents are publicly available in the INSPIRE web site
http://inspire.jrc.ec.europa.eu/reports.cfm

REF.	*AUTHOR(S)*	*TITLE*	*DATE*
INSPIRE IR Disc	INSPIRE DT	Draft Implementing Rules for Discovery Service (IR3)	2008
INSPIRE IR View	INSPIRE DT	Draft Implementing Rules for View Service 2.3	2008

Table 6 - Inspire IR References

12. Annex G – Bibliography

The following table lists the sources referenced in the present document.

All OGC change requests are publicly available at http://www.opengeospatial.org/standards/cr

All ORCHESTRA documents are publicly available at http://www.eu-orchestra.org/documents.shtml

All INSPIRE documents are publicly available at http://inspire.jrc.ec.europa.eu/reports.cfm

All WS-I documents are publicly available at http://www.ws-i.org

REF.	AUTHOR(S)	TITLE	DATE
04-042	Martell R.	OGC CSW 2.x change proposal: WSDL and SOAP usage	2004
04-094	Vretanos P.A.	Web Feature Service Implementation Specification	2005
06-094r1	OWS DT	OWS Common change request: Add SOAP encoding (OGC)	2008
08-009r1	Schäffer B.	SOAP/WSDL Common Engineering Report (OGC)	2008
AGILE P&F	AgileDelta	Performance and Features	//
Butek, 2005	Butek R.	Which style of WSDL should I use?	2005
BXSA	Lu W., Chiu K. et al.	Building a Generic SOAP Framework over Binary XML	2006
Cohen, 2003	Cohen F.	Understanding SOAP Encoding Impact on Web Service Performance in WebLogic Workshop	2003
Corera, 2007	Corera A.	Using ASP.NET SOAP Headers in Web Services for Orthogonal Interception Services	2007
Dhesiaseelan, 2004	Dhesiaseelan A.	What's New in WSDL 2.0	2004
Eff XML - IMN	//	Efficient XML Interchange Measurements Note	//
eIDM art	//	A question of identity (*http://ec.europa.eu/information_society/activities/egovernment/policy/key_enablers/eid/index_en.htm*)	//
Faster M/X	Yang Y.	Faster Data Transport Means Faster Web Services with MTOM/XOP	2007
IS SMS	Friis-Christensen et al.	ORCHESTRA Implementation Specification of the Schema Mapping Service	2007
Paranjpe, 2003	Paranjpe	How to handle binary data with SOAP	2003
Pericas, 2005	Pericas S., Sandoz P.	Fast Infoset in Java Web Services Developer Pack	2005
PRO WSDL	//	Processing Service WSDL	//
RFC2045	Freed N., Borenstein N.	Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies	1996
RFC2387			
Rothaug, 2004	Rothaug S.	The Difference Between RPC and Document Style WSDL	2004
SOAP SwA	Barton J. J., Thatte S. et al.	SOAP Messages with Attachments	2000
SOAP/1.2 AF	Nielsen H. F., Ruellan H.	SOAP 1.2 Attachment Feature	2004
SOAP/1.2, 1	Gudgin M., Hadley M.	SOAP version 1.2 Part 1 – Messaging Framework	2003

	et al.		
WS-Addr	Box D., Christensen E. et al.	Web Services Addressing (WS-Addressing)	2004
WSDL/2.0	Chinnici R., Moreau J. J. et al.	Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language	2007
WS-I AP/1.0	Ferris C., Karmarkar A., Liu C. K.	Attachments Profile Version 1.0	2006
WS-I BP/1.1	Ballinger K., Ehnebuske D. et al.	Basic Profile Version 1.1	2004
WS-I BP/2.0	Ballinger K., Ehnebuske D. et al.	Basic Profile Version 2.0	2004
WS-RM	Bilorusets R., Box D. et al.	Web Services Reliable Messaging Protocol (WS-ReliableMessaging)	2005
WSS/1.0	A. Nadalin, C. Kaler et al.	Web Services Security: 3 SOAP Message Security 1.0 (WS4 Security 2004)	2004
XBC Charact	Goldman O., Lenkov D.	XML Binary Characterization	2005
XML SOAP BD	Bosworth A., Box D. et al.	XML, SOAP and Binary Data	2003
XML SSP	Bartel M., Boyer J. et al.	XML Signature Syntax and Processing (Second Edition)	2008
Yang, 2007	Yang Y.	Boost Web Service Performance in JAX-WS with Fast Infoset	2007
INSPIRE NSA/3.0	*NSDT*	*INSPIRE Network Services Architecture v3.0*	*2008*

Table 7 - Bibliography

REF.	AUTHOR(S)	TITLE	DATE
HTTP/1.1	Fielding R., Gettys J. et al.	Hypertext Transfer Protocol -- HTTP/1.1	1999
MTOM	Gudgin M., Mendelsohn N. et al.	SOAP Message Transmission Optimization Mechanism	2005
SOAP/1.1	Box D., Ehnebuske D. et al.	Simple Object Access Protocol (SOAP) 1.1	2000
SOAP/1.1-MTOM	Angelov D., Karmarkar A. et al.	SOAP 1.1 Binding for MTOM 1.0	2006
WSDL/1.1	Christensen E., Curbera F. et al.	Web Services Description Language (WSDL) 1.1	2001
WS-I BP/1.2	Ferris C., Karmarkar A. et al	Basic Profile Version 1.2	2007
XML/1.0	Bray T., J. Paoli et al.	Extensible Markup Language (XML) 1.0	2006
XOP	Gudgin M., Mendelsohn N. et al.	XML-binary Optimized Packaging	2005

Table 8 - INSPIRE SOAP framework used standard

European Commission

EUR 23635 EN – Joint Research Centre – Institute for Environment and Sustainability
Title: INSPIRE NETWORK SERVICES SOAP Framework
Author(s): Matteo Villa, Giovanni Di Matteo, Roberto Lucchi, Michel Millot, Ioannis Kanellopoulos
Luxembourg: Office for Official Publications of the European Communities
2008 – 61pp. – 21 x 29.7 cm
EUR – Scientific and Technical Research series – ISSN 1018-5593
ISBN 978-92-79-10966-9
DOI 10.2788/36426

Abstract
The goal of this document is to provide a definition and rationale for a proposed INSPIRE SOAP framework (SOAP nodes policy, RPC, attachments, WS-I, WSDL…) and description of issues and solutions for the specific geospatial domain.

How to obtain EU publications

Our priced publications are available from EU Bookshop (http://bookshop.europa.eu), where you can place an order with the sales agent of your choice.

The Publications Office has a worldwide network of sales agents. You can obtain their contact details by sending a fax to (352) 29 29-42758.

The mission of the JRC is to provide customer-driven scientific and technical support for the conception, development, implementation and monitoring of EU policies. As a service of the European Commission, the JRC functions as a reference centre of science and technology for the Union. Close to the policy-making process, it serves the common interest of the Member States, while being independent of special interests, whether private or national.

LB-NA-23635-EN-C