3GPP TR 32.809 V7.0.0 (2006-12)

Technical Report

3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Telecommunication management; Feasibility Study of XML-based (SOAP/HTTP) IRP Solution Sets (Release 7)

The present document has been developed within the 3rd Generation Partnership Project (3GPP ™) and may be further elaborated for the purposes of 3GPP.
The present document has not been subject to any approval process by the 3GPP Organizational Partners and shall not be implemented.
This Specification is provided for future development work within 3GPP only. The Organizational Partners accept no liability for any use of this Specification.
Specifications and reports for implementation of the 3GPP ™ system should be obtained via the 3GPP Organizational Partners' Publications Offices.

Keywords
UMTS, XML, SOAP, HTTP, management

3GPP

Postal address

3GPP support office address
650 Route des Lucioles - Sophia Antipolis
Valbonne - FRANCE
Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Internet
http://www.3gpp.org

Copyright Notification

No part may be reproduced except as authorized by written permission.
The copyright and the foregoing restriction extend to reproduction in all media.

© 2006, 3GPP Organizational Partners (ARIB, ATIS, CCSA, ETSI, TTA, TTC).
All rights reserved.

Contents

Foreword

This Technical Report has been produced by the 3rd Generation Partnership Project (3GPP).

The contents of the present document are subject to continuing work within the TSG and may change following formal TSG approval. Should the TSG modify the contents of the present document, it will be re-released by the TSG with an identifying change of release date and an increase in version number as follows:

Version x.y.z

where:

- x the first digit:
  - 1 presented to TSG for information;
  - 2 presented to TSG for approval;
  - 3 or greater indicates TSG approved document under change control.
- y the second digit is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc.
- z the third digit is incremented when editorial only changes have been incorporated in the document.

1 Scope

This present document is for the requested 3GPP Release 7 Work Item "XML-based (SOAP/HTTP) IRP Solution Set".

The purpose of the present document is to give TSG SA5 some arguments, through a survey work, about the need to specify a new Solution Set for all Interfaces, Network Resource Models and Data Definition IRPs based on XML-based (SOAP/HTTP) technology, as additional choice to CORBA and CMIP Solution Sets.

This document is intended to gather all information in order to provide a survey work on XML-based (SOAP/HTTP) Solution Set, based on the following sections:

(1) XML-based Solution Set technical introduction

(2) Business Status: Reasons

(3) Standardization Status

(4) Vendor Status

(4) Performance status:

(5) Economic Status

2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, the latest version applies. In the case of a reference to a 3GPP document (including a GSM document), a non-specific reference implicitly refers to the latest version of that document *in the same Release as the present document*.

[1] SOAP Specification definition part 1: http://www.w3.org/TR/soap12-part1

[2] SOAP specification definition part 2: http://www.w3.org/TR/soap12-part2/

[3] MTOSI XML implementation user guide

[4] MTOSI communication styles

[5] OSS through Java Web Services Integration Profile,

[6] SOAP 1.2 test collection : http://www.w3.org/TR/soap12-testcollection/

[7] XML Information Set : http://www.w3.org/TR/xml-infoset/

[8] Comparison of Web Services, Java-RMI and CORBA service implementations, S5-056555,

3 Definitions and abbreviations

3.1 Definitions

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

SOAP: formal set of conventions governing the format and processing rules of a SOAP message. The convention includes the interaction between SOAP nodes generating/accepting SOAP messages in order to exchange information along the SOAP message path.

SOAP application: An entity, typically software, that produces, consumes or otherwise acts upon SOAP messages in a way conforming to the SOAP processing model.

SOAP body: A collection of 0 or more element information items targeted at an ultimate SOAP receiver within the SOAP message path.

SOAP binding: formal set of rules for carrying a SOAP message within or on top of another protocol for the purpose of exchange.

SOAP envelope: Outermost element information item of a SOAP message.

SOAP feature: An extension of the SOAP messaging framework.

SOAP header: A collection of 0 or more SOAP header blocks, each of which might be targeted at any SOAP receiver within the SOAP message path.

SOAP header block: An element information item used to delimit data that logically constitutes a single computational unit within the SOAP header. The type of a SOAP header block is identified by the XML expanded name of the header block element information item.

SOAP intermediary: A SOAP intermediary is both a SOAP receiver and a SOAP sender.

SOAP message: Basic unit of communication between SOAP nodes.

SOAP message exchange pattern: Template for exchanging SOAP messages between SOAP nodes, enabled by one or more underlying SOAP protocol bindings

SOAP message path: Set of SOAP nodes through which a single SOAP message passes.

SOAP module: A specification that contains the combined syntax and semantics of SOAP header blocks. A SOAP module realizes 0 or more SOAP features.

SOAP node: embodiment of the processing logic necessary to transmit, receive, process or relay a SOAP message. A SOAP node is responsible for enforcing the rules that govern the exchange of SOAP messages. It accesses the services provided by the underlying protocols through one or more SOAP bindings

SOAP receiver: A SOAP node that accepts a SOAP message.

SOAP role: A SOAP receiver's expected function in processing a message. A SOAP receiver can act in multiple roles.

SOAP sender: SOAP node that transmits a SOAP message

Ultimate SOAP receiver: SOAP receiver that is the final destination of a SOAP message. It is responsible for processing the content of the SOAP body and any SOAP header blocks targeted at him.

3.2 Abbreviations

For the purposes of the present document, the following abbreviations apply:

CORBA	Common Object Request Broker Architecture
CORBA/IDL	Common Object Request Broker Architecture/Interface Definition Language
EM	Element Manager
EMS	Element Management System
IDL	Interface Definition Language
IRP	Integration Reference Point
IS	Information Service
MEPs	Message Exchange Patterns
NM	Network Manager
NMS	Network Management System
OAM&P	Operations, Administration, Maintenance and Provisioning
OS	Operations System

OSS	Operations Support System
SOAP	Simple Object Access Protocol
SS	Solution Set
TCP/IP	Transmission Control Protocol/ Internet Protocol
TMF	TeleManagement Forum
UDDI	Universal Description Discovery and Integration
WSDL	Web Services Description Language

4 XML-based (SOAP/HTTP) technical introduction

This section is a quick technical overview of XML-based Solution Set, SOAP and SOAP/HTTP that can be used as a reference to better understand the context of this document. This is not exhaustive but provides the main principles and gives enough information to be able to make a judgement concerning this candidate technology.

4.1 Foreword and vocabulary

The following section will introduce XML-based technologies that uses, as a vehicle, SOAP technologies as well as its add-ons such as WSDL and UDDI.

It has to be clear to the reader that SOAP is the protocol that is used to introduce XML-based IRP Solution Set.

As well, if SOAP is the XML-based protocol used, it comes with a number of add-ons such as WSDL, UDDI, and that this set of "tools" are an entire part of the XML-based IRP Solution Set that is being defined in this study.

4.2 What is SOAP?

SOAP is a lightweight protocol defined by the w3c organisation (see [1] and [2]). SOAP is intended to exchange information in decentralized and distributed environment. It uses XML technologies that allow defining an extensible messaging framework providing a message construct that can be exchanged over a variety of underlying protocols. As well, the framework has been designed to be independent of any particular programming model and other implementations specific semantics.

SOAP stands for Simple Object Access Protocol and is a XML messaging and invocation protocol. It does not specify a transport event though it is very commonly used with HTTP.

SOAP was designed with the two main goals that are simplicity and extensibility. Later on in this technical introduction there will be a detailed description about the way they have been implemented.

Globally SOAP is said to be agnostic in the sense that:

- Independent of transport protocol used to carry the messages (HTTP, SMTP, etc, etc...)
- Independent from other web services such as WSDL or UDDI
- Independent from programming languages

4.3 How does SOAP work?

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 a number of SOAP intermediaries.

4.3.1 SOAP message construct

A SOAP message is specified as an XML infoset that consists in a SOAP envelope, a SOAP header and a SOAP body.

The SOAP envelope element information item is embedded within an XML infoset and specifies that this infoset is a SOAP message. It contains a local name (envelope) as well as a namespace and a number of namespace qualified attribute information items (for example encoding Style). Finally it has to contain (Mandatory) a Body element information item and optionally a Header element information item.

The SOAP header provides a mechanism for extending a SOAP message in a decentralized and modular way. The header can be inspected, inserted or deleted by SOAP nodes. It allows SOAP intermediary nodes to add value added services such as correlation, audit, security etc, etc… Basically, the intermediaries SOAP nodes can potentially check through the header to get info if necessary or to know which SOAP node is to be the ultimate SOAP receiver. The SOAP header is defined by a local name (header), a namespace, a number of namespace qualified attribute information items and a number of namespace element information items (header blocks). The header blocks attributes can be, for example, EncodingStyle, role, mustUnderstand or relay.

The SOAP body provides a mechanism to transmit information to an ultimate SOAP receiver. It is a mandatory field since it contains the message or payload that is to be processed by the ultimate SOAP receiver.

An example of the SOAP flexibility is that the decision to determine which data is to be placed in the header block or body is decided at the application design stage.

4.3.2 SOAP processing model

The SOAP processing model specifies the way a SOAP receiver processes a SOAP message.

While processing a SOAP message, a SOAP node (identified by an URI) is said to act in one or more SOAP role, each of which is identified by a URI known as the SOAP role name. For example, there are three main role names in the context of SOAP:

- Next: Each SOAP intermediary and the ultimate SOAP receiver must act in this role
- None: SOAP nodes must NOT act in this role
- UltimateReceiver: Only the ultimate receiver must act in this role

For example, a SOAP header block may carry a role attribute information item that is used to target the header block at SOAP nodes operating in the specified role. In the same way, a SOAP header block may carry a mustUnderstand attribute information item set to true, in which case the header block is said to be mandatory because a targeted node must process the header block and act according to its content.

Finally, the ultimate SOAP receiver must process correctly the SOAP body.

4.3.3 SOAP extensibility model

As said earlier, SOAP provides a simple messaging framework whose core functionality is concerned with providing extensibility. Mainly, the SOAP extensibility is provided by the following three “possibilities”:

- SOAP Message Exchange Patterns (MEPs)
- SOAP modules
- SOAP protocol binding framework

4.3.3.1 SOAP features

A SOAP feature is an extension of the SOAP messaging framework. Although SOAP poses no constraints on the potential scope of such features, those can be reliability, security, correlation, routing or MEPs such as request/response, one-way and peer to peer conversation.

There are two mechanisms through which features can be expressed: SOAP processing model (See Section 4.3.2) or SOAP protocol binding framework (See sub-clause 4.3.3.4). The first mechanism is already known. The second mechanism mediates the act of sending/receiving SOAP messages by a SOAP node via an underlying protocol.

The SOAP processing model enables SOAP nodes that include the necessary mechanisms to implement one or more features to express such features within the SOAP envelope as SOAP header blocks. Such header blocks can be intended for any SOAP node or nodes along the SOAP message path. The combined syntax and semantics of SOAP header blocks are known as a SOAP module (See sub-clause 4.3.3.3).

On the other hand, the SOAP protocol binding operates between two adjacent SOAP nodes along a SOAP message path. There are no requirements that the same underlying protocol is used for all hops along a SOAP message path. In

some cases, underlying protocols are equipped, either directly or through extension, with mechanisms for providing certain features. The SOAP protocol binding framework provides a scheme for describing these features and how they relate to SOAP nodes through a binding specification.

Globally, in the case where the feature requires an end to end processing, it is probably recommended to use the SOAP header blocks to reuse the SOAP processing model rules. Otherwise, SOAP protocol binding can be used.

A SOAP feature is fully defined by the following:

- A URI used to name the feature. This enables the feature to be unambiguously referenced in description language or during negotiation
- The information state required at each node to implement the feature
- The processing required at each node in order to fulfil the obligations of the feature, including handling of communication failures that might occur in the underlying protocol
- The information to be provided from node to node

4.3.3.2 SOAP Message Exchange Patterns (MEPs)

A Message Exchange Pattern (MEP) is a template that establishes a pattern for the exchange of the messages between SOAP nodes. In a way, MEPs are a kind of feature.

A MEP is defined by the following:

- A URI used to name the MEP
- Description of life cycle of a message exchange conforming to the pattern
- Description the temporal/causal relationships, if any, of multiple messages exchanged in conformance with the pattern
- Description of the normal and abnormal termination of a message exchange conforming to the pattern

4.3.3.3 SOAP modules

SOAP module refers to the specification of the syntax and semantics of one or more header blocks.

A SOAP module:

- Must identify itself with a URI. This enables the module to be unambiguously referenced in description languages or during negotiation
- Must declare the features provided by a module
- Must clearly and completely specify the content and semantics of the SOAP header blocks used to implement the behaviour in question, including if necessary any modifications to the SOAP processing model

4.3.3.4 SOAP protocol binding framework

SOAP enables exchange of SOAP messages using a variety of underlying protocols. The formal set of rules for carrying a SOAP message within or on top of another protocol for the purpose of exchange is called a binding.

A SOAP binding specification:

- Declares the features provided by a binding
- Describes how the services of the underlying protocol are used to transmit SOAP messages infosets
- Describes how the services of the underlying protocol are used to honour the contract formed by the features supported by that binding
- Describes the handling of all potential failures that can be anticipated within the binding

- Defines the requirements for building a conformant implementation of the binding being specified

Globally, it can be considered that a binding does not really provide a separate processing model and does not constitute a SOAP node by itself. Rather, a SOAP binding is an integral part of a SOAP node.

Two or more bindings can offer a given optional feature using different means. One binding might exploit an underlying protocol that directly facilitates the feature and the other binding might provide the necessary logic itself. In such cases, the feature can be made available to applications in a consistent manner, regardless of which binding is being used.

Finally, since the SOAP processing model describes the processing that is common to all SOAP nodes when receiving a message, the purpose of a binding is to augment the core SOAP rules with additional processing that is particular to the binding and to specify the manner in which the underlying protocol is used to transmit information between adjacent nodes in the message path.

4.3.4 SOAP 1.1 Versus SOAP 1.2

The following section is to introduce the delta between SOAP 1.1 and SOAP 1.2. The XML working group evaluated the technical solutions proposed by SOAP 1.1. The result of this study work is the elaboration and proposal of a SOAP 1.2 version which is a more extensible and protocol-independent XML-based messaging framework.

The key point to keep in mind is that SOAP 1.2 can do everything that SOAP 1.1 does, and more.

4.3.4.1 SOAP 1.2 offers a clear processing model and better interoperability

Many interoperability issues in SOAP 1.1 were caused by ambiguities in the processing model. For example the scope of the “mustUnderstand” attribute in the processing of a message as well as the processing being done by intermediary impacted interoperability.

The XML protocol working group study raised more than 400 issues and built a clean, robust and unambiguous SOAP 1.2 processing model. W3C actually ran a collection of tests that proved the better interoperability of SOAP 1.2 (Ref [6]).

4.3.4.2 SOAP 1.2 based on XML Information Set

SOAP version 1.2 is based on the XML Information Set (See [7]). This is a significant change since with SOAP 1.2 a SOAP message is specified as an infoset which is carried from one SOAP node to another. In SOAP 1.1, which was based in XML 1.0 serialization, SOAP 1.2 places no restriction about how the infoset is transported. For example, it could be done using HTTP and XML 1.0 or a completely different mean of transportation. SOAP 1.2 is agnostic to this which allows for compression, optimization and other performance gains.

4.3.4.3 SOAP 1.2 gives developers protocol independence by providing a binding framework.

Building on top of the XML infoset, the SOAP 1.2 specification defines a binding framework explaining the responsibility of the mechanism carrying a SOAP message from one SOAP node to another. This makes SOAP processors underlying protocol agnostics and SOAP version 1.2 protocol independent. SOAP 1.2 messages can be carried over HTTP, SMTP or any other protocol for which a binding conforms the binding framework.

As a conclusion, SOAP 1.2 is fully protocol independent.

4.3.4.4 SOAP 1.2 includes HTTP binding for better integration to the WWW

Since SOAP 1.2 is likely to interoperate with the World Wide Wed, it needs to conform to the web architectural principles. SOAP 1.2 defines a web method feature and the SOAP HTTP binding provides support for both HTTP GET and POST operations. This implies that SOAP gets the benefits of the web infrastructure such as HTTP caches.

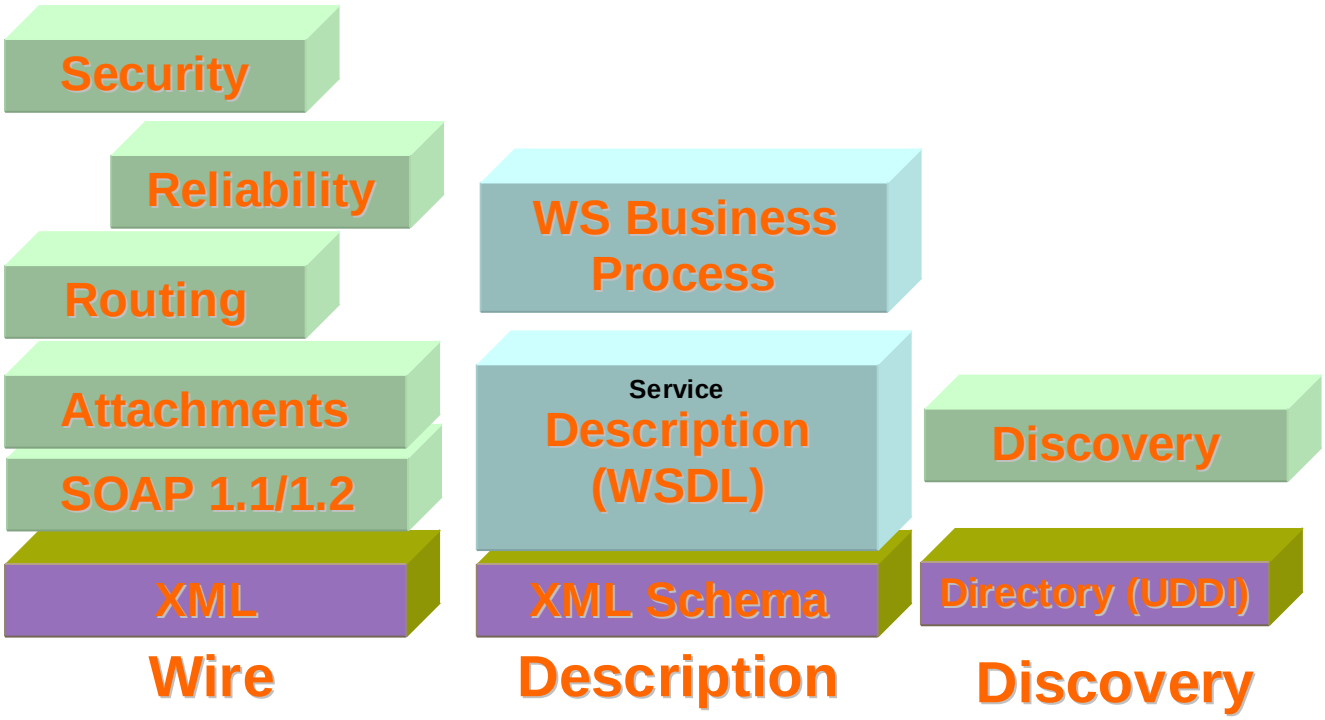
Basically, SOAP 1.2 uses existing, established web technologies and it allows achieving better performance.

4.3.4.5 SOAP 1.2 delivers a well defined extensibility model

The biggest strength of SOAP 1.2 is coming from its extensibility model. SOAP 1.1 model has been reworked and formalized as features and properties that can be expressed either in the SOAP envelope or via the underlying protocol binding, making SOAP1.2 very flexible and making it take advantage of any feature that the underlying protocol would be providing.

4.4 Heading towards a XML-based IRP Solution Set

This study's goal is to assess whether or not SA5 can head towards a XML Solution Set based on SOAP as a protocol, WSDL as a description language and UDDI as a discovery "layer". WSDL and UDDI will be described in the following section.



4.4.1 Web Service Description Language (WSDL)

WSDL is a Standard to define Web Services.

WSDL is an XML format that allows Web Services description as a set of endpoints operating on messages containing either a document oriented or procedure oriented information. Operations and messages are described at an abstract level and then bound to a concrete network protocol and message format to define an endpoint.

Basically WSDL describes the format and protocols of a web service in a standardized way. For example, the requestor and the Service Provider have a standard way to describe:

- Service to be provided
- Messages needed to be sent by both parties in order to provide the service:
  - Data contained in each message
  - Binding of the messages to a communication protocol
- What the service does

- How the service is accessed
- Where the service is located

The Service provider publishes service WSDL description to a UDDI registry. The service requestor discovers the service by searching the UDDI registry and then binds the WSDL description to a specific message it uses to access the service. Finally, the service requestor can use the service.

4.4.2 Universal Description Discovery and Integration (UDDI)

UDDI allows service providers to publish information about their services. The requestor can then find those information when looking for a service.

This is done using the UDDI functionalities providing a registry in which information such as business entity, service provided by this entity and the WSDL description of those services is stored. UDDI provides as well an interface to insert information of the registry and possibly search for a service.

4.5 More about SOAP/HTTP XML-based Solution Set

The SOAP HTTP binding provides a binding of SOAP to HTTP. The binding conforms to the SOAP protocol binding framework defined in Section 4.3.3.4.

The SOAP HTTP binding is optional and SOAP nodes are not required to implement it. However, this binding does not preclude the development of other bindings to HTTP or to other protocols, but communication with nodes using such bindings is not a goal even if those nodes can use HTTP features or status codes.

4.5.1 Correct use of HTTP

The SOAP HTTP binding defines a base URI which is the HTTP Request-URI or the value of the HTTP Content-Location header field.

This binding of SOAP to HTTP is intended to make appropriate use of HTTP as an application protocol. In fact, this binding is not intended to fully exploit the HTTP features but rather to use HTTP specifically for the purpose of communicating with other SOAP nodes implementing the same binding.

4.5.2 Interoperability with non SOAP HTTP implementations

When used with MEPs, the HTTP messages produced by the SOAP HTTP binding are likely to be indistinguishable from those produced by non SOAP implementations performing similar operations. Accordingly, some degree of interoperation can be made possible between SOAP nodes and other HTTP implementations when using this binding.

4.5.3 MEP operation

A SOAP node instantiated at an HTTP client may assume the following two role properties:

- Requesting SOAP node
- Responding SOAP node

4.5.3.1 Behaviour of the requesting SOAP node

This binding supports streaming and, as a result, requesting SOAP nodes must avoid deadlock by accepting and, if necessary, processing SOAP response information while the SOAP request is being transmitted.

A number of state help for that purpose:

- *Init*: In this state a HTTP request is formulated and the transmission of the request is initiated
- *Requesting*: the sending of the request continues while waiting for the start of the response message. A number of statuses can be allocated at this stage (successful, Ok, redirection, client error, bad request etc, etc…). Depending on the status, the node acts in consequence. It has to be noticed that the SOAP HTTP binding follows

the rules of any HTTP application which means that an implementation of the SOAP HTTP binding must understand the class of any status code

- *Sending + Receiving*: the transmission of the request message and receiving of the response message is completed. The response message is assumed to contain a SOAP envelope serialized according to the rules for carrying SOAP messages in the media type given and the content-type header field

- *Receiving*: in the receiving state, receiving of the response is completed

- *Success and Fail*: Success and Fail are the terminal states of the Request-Response and SOAP response MEPs. Control over the message exchange context returns to the local SOAP node

4.5.3.2 Behaviour of responding SOAP node

In the same way, the Responding SOAP node has a number of state that are driving the node behaviour

- *Init*: the binding waits for the start of an inbound request message. In this state, no SOAP message has been received

- *Receiving*: the binding receives the request and any associated message and waits for the start of a response message to be available

- *Receiving + Sending*: the binding completes receiving of the request message and transmission of the response message

- *Sending*: the binding completes transmission of the response message

- *Success and Fail*: they are the terminal states for the Request- Response and SOAP Response MEPs. From the point of view of the local node, this message exchange has completed

4.5.4 Security considerations

The SOAP HTTP binding can be considered as an extension of the HTTP application protocol. As such, all the security considerations described in the HTTP specification apply to the SOAP HTTP binding. In particular, HTTPS is supported.

On top of that, SOAP security items apply as well (see [2]).

4.5.5 Request-Response RPC

One of the goals of the SOAP to HTTP binding is to encapsulate remote procedure call functionality, using XML's flexibility and extensibility.

In this context, RPC is used to model a certain programmatic behaviour. To invoke a SOAP RPC, the following information is necessary:

- The address of the target SOAP node. URI can be included in header blocks or outside of SOAP message, in bindings

- The procedure or method name

- The identities and values of any arguments to be passed to the procedure or method together with any output parameters and return value

- A clear separation of the arguments used to identify the Web resource which is the actual target for the RPC, as contrasted with those that convey data or control information used for processing the call by the target resource

- The message exchange pattern which will be employed to convey the RPC

- Optionally, data which may be carried as a part of SOAP header blocks

5 Business status

This section is a rough business assessment of the proposed XML-based IRP solution set. The goal of this section is to assess and estimate whether or not the proposed solution can be an alternative to existing Solution Sets, notably CORBA/IDL.

It has to be clear as well that the following is not a plea against CORBA but just a comparison with an existing solution set. Globally, it will be demonstrated that everything (or nearly) that can be achieved using CORBA is do-able using the XML-based Solution Set.

5.1 Comparative analysis

The comparison will be made against two different aspects which are the computing model and the features supported by each technology.

The comparison will be based on a number of criteria developed in the following.

Finally, this is obviously not an exhaustive comparison but that should be enough information to get a good enough assessment of the proposed XML-based solution set.

The following table provides a first level of comparison between the two technologies:

CORBA stack	XML-based stack
IDL	WSDL
CORBA Services (naming service, trader service, IR)	UDDI
CORBA Stubs/Skeletons	SOAP Message
CDR (common data representation) binary encoding	XML Unicode encoding
GIOP/IIOP	HTTP
TCP/IP	TCP/IP

5.1.1 Computing model

5.1.1.1 Data model

CORBA is a true Object Oriented component framework, whereas the XML-based solution set are focused around a message passing paradigm with no notions of objects.

As well, with CORBA, there is a tight coupling between the client and the server. First of all, both of them must share the same interface and must run an ORB (Object Request Broker) at both ends. Then, the interaction between the client and the server may be achieved directly with no need to further intermediation. In the case of the XML-based solution, everything is decoupled. The client sends or receives a message.

5.1.1.2 Scalability and reliability

In CORBA, the Portable Object Adapter (POA) policies combined with the Fault-tolerant CORBA features and the Load-balancing CORBA service provide the desired scalability to CORBA applications

In the context of XML-based solution, standards do not address these issues. However, this is done on purpose and left to the responsibility of the components.

5.1.1.3 Static and runtime checks

CORBA IDL is strongly typed and provides some static guarantees, DII (Dynamic Invocation Interface) addresses runtime checks

In the XML-based solution, there is no standardized infrastructure support to offer static checking. At runtime, only the structure of the SOAP message is checked, and the payload is only required to be a piece of well-formed XML. It is the responsibility of the application to verify the payload further (i.e., validate against a schema). In the future, WSDL could be used to generate a mapping to a programming language to offer some static guarantees.

XML Schemas are more expressive than IDL and define some syntactic checks (e.g., regular expression describing a date format, etc.), as well as some semantic checks (e.g., range constraints), which cannot be captured in IDL.

5.1.2 Features supported

5.1.2.1 Location transparency

CORBA client applications obtain references to objects and invoke operations without concern for the location of the objects

XML-based solution client applications (using SOAP) refers to services using URLs which implicitly encode the location (IP address) of the service

5.1.2.2 Registry

CORBA defines an Interface Repository (IR) that provides run-time information about IDL interfaces and invokes via DII

UDDI provides searchable central repositories using a publish/subscribe mechanism to store service definitions. UDDI registries can store structural information about the service in the form of a WSDL specification

5.1.2.3 Service discovery

CORBA Naming service provides lookup via service name. The Trading Object Service supports an advanced lookup, registration and discovery of services based on service type.

In the XML-based solution, UDDI registries are allowing to discover services matching a given interface.

5.1.2.4 Firewall Traversal

The upcoming CORBA Firewall Traversal specification allows CORBA requests to be routed through firewalls. Explicit routing information is added to the interoperable object references (IOR).

For XML-based solution, since the preferred transport protocol is HTTP, firewalls are usually configured to allow inbound and outbound HTTP traffic

5.1.2.5 Security

Security features include authentication, authorization, encryption, data integrity, delegation, non-repudiation and auditing

Security features are supported by the CORBA Security service

The XML-based solution stack does not define security services. However, security can be addressed by transport network and internet technologies like SSL and XML-Signature

5.1.2.6 Platform independence

CORBA has been designed to be platform independent. This includes hardware, operating systems, and programming and scripting languages independency.

On the XML-based solution, since all messages communication is via SOAP, and that no other restrictions (e.g., platform similarities) are imposed on the client or the server, the only requirement is to be able to read and write SOAP messages (i.e. XML documents)

5.1.3 Summary

Aspect	CORBA	XML-based SS
Data model	Object model	SOAP message exchange model
Client-Server coupling	Tight	Loose
Location transparency	Object references	URL
Type system	IDL	XML schemas
	static + runtime checks	runtime checks only
Error handling	IDL exception	SOAP fault messages
Serialization	built into the ORB	can be chosen by the user
Parameter passing	by reference	by value (no notion of objects)
	by value (*valuetype)*	
Transfer syntax	CDR used on the wire	XML used on the wire
	binary format	Unicode
State	stateful	stateless
Request semantics	at-most-once	defined by SOAP
Runtime composition	DII	UDDI/WSDL
Registry	Interface Repository	UDDI/WSDL
	Implementation repository	
Service discovery	CORBA naming/trading service	UDDI
Language support	any language with an IDL binding	any language
Security	CORBA security service	HTTP/SSL, XML signature
Firewall Traversal	work in progress	uses HTTP port 80

5.2 Conclusion

Basically, it seems that most of the things that can be achieved using CORBA are achievable as well using the proposed XML-based solution.

There are a number of points that are not actually defined at the XML-based solution specification level but this is on purpose and those are left to the discretion of the design team when implementing the application.

As a result of that, the flexibility of XML-based solution is clearly demonstrated.

XML-based solution seems as well to be easy to use with possibly less constraints than CORBA based solution set. There is no need to compile each time a change or a new implementation is done and the fact that there is a binding to HTTP seems interesting as well.

However there are few concerns that would need probably further studies. For example, there is a question mark towards the performance of the system (See clause 8). As well, the interoperability might not be yet up to speed.

Finally, it looks like the notification are not defined in the XML-based Standards and are therefore open to the four winds which potentially means that every standardisation organism is setting its own standard on those. SA5 should look into that as well for further studies.

6 Standardisation status

The XML-based IRP Solution Set discussed in this document is built on SOAP protocol, WSDL and UDDI. All three are today standardised by the W3C organization. The details of those specifications can be found by the link given in Reference [2] and [3].

Since this solution is transport independent, it can easily be based on a standardised transport layer.

Today SOAP version is v1.2 and is not expected to change. However, the extensibility features are open and linked to bindings to protocol.

7 Usage status

There is a real growing demand for the XML-based complete set of solution, which are today very widely used and implemented in the industry. Especially XML-based solutions over HTTP or HTTPS since most of the web applications are choosing this kind of implementation. It seems to be a well known and accepted fact, and is driving the industry at the moment. For example, travel booking companies, music download sites and banking activities are using XML-based solution to perform their activities.

This is in that regard, really, that this feasibility study has been set up. The question asked really is: "can we use, or re-use" those XML-based technologies in the context of the telecom management and this is the answer that has to be provided as an output to this feasibility study.

As a matter of fact, a number of Telecom Standard organisations have already made a positive assessment regarding this XML-based solution set.

For example, the TMF MTOSI industry group has already progressed on that topic and presented their work to the SA5. In the same way OSS/J study group have produced an assessment document that concludes that this the way going forward. Both those documents are available and given as references [3] and [4]. Other groups from Telecommunications and other industries are actively working on that topic.

8 Performance status

8.1 State of the art

It seems that the performances offered by an XML-based solution would not be as good as the ones provided by a CORBA solution set.

Globally, it seems to be an agreed and accepted point that SOAP, being XML-based, is too verbose as compared to CORBA IIOP's binary format. Therefore, it is likely to introduce a performance hit.

However, that may be true for certain kinds of applications, but in most complex business applications, the cost of message passing is dwarfed by the cost of processing the business logic

Anyway, this should be studied further to get maybe a clearer overview of the real performances impacts.

8.2 Way forward

There is clearly an argument to be discussed concerning performances. However, the industry makers, including in the Telecommunication world seem to privilege the wider use of an XML-based solution rather than what seems to be a small enough and acceptable impact on performances.

Unfortunately, at this stage, there is not a sufficient set of references to give definite arguments on that purpose. But a discussion on that topic should be set up.

The discussion could stand, for example, over the use of an XML-based solution set against all our IRPs. Or is it only applicable to PM, CM basic and Bulk and maybe not for FM, even though there are no existing studies demonstrating the inefficiency of those type of solution in this sector.

On top of that, it would be very interesting, in the context of SA5, and even beyond, to set up a work task group that would study the performances in an XML-based environment versus existing solutions to find out if there is an impact on performances and if any, how big it is.

8.3 Examples of Performances studies

In the following section, we'll go through two performance studies which should allow us to get a better understanding on the potential impacts that an XML-based solution set is introducing.

8.3.1 Comparison of XML, Java-RMI and CORBA service implementation

This study is the analysis of a white paper presented by China Mobile during SA5 # 43 and is based on the document given in Ref [8].

8.3.1.1 Introduction and context of this study

Previous studies ran upon previous SOAP and HTTP versions had identified significant issues in terms of performances for a system implemented using XML-based solution sets. The intent of this very study is to confirm whether or not XML-based solution set have overcome this important issue. In fact, in prior studies, when comparing Java, CORBA and XML-based solutions, a degradation factor of up to 400 in disfavour of XML-based solution was demonstrated.

This study is based on three cases that will be tested against each of the technologies. The testing environment is exactly the same for each technology. The three cases that will be tested against are:

- A "calculator" which defines a simple four-function calculator with operations that each required an integer input argument and generated an integer response. This is the light case scenario that should involve minimal overheads,
- A case called "data", which involves a string input argument and a returned structure with integer, string array, and double array fields. At this stage, the XML encoding has to include some more significant structural meta-data,
- Finally, the last case is called "large data" and it models a real world data retrieval application. For example, intranet use of Java-RMI and CORBA frequently has a client running in one departmental system that uses a middle-layer server to access a database in some other departmental system. This is the kind of system/application that an XML solution sets will have to take on if they are to establish themselves as a preferred technology, or at least as an alternative to Java or CORBA.

More details about the implementation of each of those study cases can be found in the document Ref [8].

8.3.1.2 Cases study results

8.3.1.2.1 Costs associated directly with communications protocols

Typically, an XML-based implementation has a client connecting to a service, submitting a single request for data and terminate. Such applications are unlikely to put any great demands on either the server host's CPU, or a communications network. Therefore, at this level XML-based solution perform pretty well in comparison with the distributed objects systems.

Globally, any remote connection incurs the establishment costs of a TCP/IP connection. Typically, an XML-based solution will only incur the cost of one connection. The subsequent exchanges using HTTP may be non optimal. For example a request can be sent in two parts (HTTP headers in the first message, SOAP envelope with request in the second part). The same goes for a response. Despite all that, XML-based solution performs best.

CORBA has to take into account the fact that the client contacts a name service to obtain a reference to its server. This costs a TCP/IP connection that implies a decrease of performances.

Finally, Java-RMI has the most complex solution. Establishment of a connection towards the rmiRegistry and submission of a lookup request, establishment of a connection to a HTTP server, posting a request in order to download a class file and finally, exchanges with the current server. This costs dearly and implies additional data traffic. However, Java still performs better than CORBA on that topic.

Technology	Total Latency	Total Packets	Total data transferred in bytes
XML-Based	0.11s	16	3338
CORBA	0.86	24	3340
Java-RMI	0.32	48	7670

As a conclusion, the issues raised previously against XML-based solutions are now fixed. The traffic between the client and the server is now more efficient, even though there is still a bigger amount of data to be transmitted. It has to be noted that the introduction of HTTP 1.1 (versus HTTP 1.0) helped a lot in that matter, especially due to the fact that

now the HTTP connections are now persistent (keep alive) and that the "chunked" response format is now incorporated with HTTP 1.1.

8.3.1.2.2 Costs of document transfer

The image of large XML documents replete with textual data, mark up tags, and data type specifiers had lead to an expectation of even poorer performance from an XML-based solution. In fact, what we see is that the relative performance is poor with the simpler data transfers ("calculator") but the difference is less marked where the data volume is greater ("large data"). We see that the XML-based solution requires from three to five times as many data packets as the other solutions, and from three to ten times as many bytes but this is without using any kind of compression.

The CORBA implementation is the one that performs the best while java performs poorly.

As a conclusion, it seems that the five fold or so increase in data transfer costs for XML-based type of implementations may be tolerable for an intranet application for example. Local bandwidths should be high and local switched Ethernet systems should reduce contention. However, if the application requires encryption of data traffic then the larger amount of data will incur additional processing costs for encryption.

Example	Technology	Packets	Total data transferred
"Calculator"	XML-Based	48.931	10.360.814
	CORBA	10.007	1.400.851
	Java	10.098	1.017.477
"Data"	XML-Based	55.617	16.053.312
	CORBA	10.007	2.236.661
	Java	10.050	2.451.790
"Large data"	XML-Based	1.143.608	1.134.974.047
	CORBA	475.235	344.363.683
	Java	1.354.377	449.330.931

8.3.1.2.3 Costs of XML generation and parsing

There is clearly time and space costs associated with the use of XML encoding. In terms of CPU usage, Java performs best even though the "large data" scenario proved to show poor enough performances on the client size. The XML-based solution set required up to six times as much CPU power on the server and the client-side parsing of the large data as even a higher cost. CORBA is performing OK but not as well as Java.

On the memory space aspect, once again XML-Based solution Sets are consuming much more than the other two alternatives solutions.

Globally, the cost of XML-Based solutions in that domain is significant but can easily be corrected by setting up more CPU and Memory. It then becomes a financial costs discussion.

Application	"Calculator"		"Data"		"Large Data"	
Technology	**Client CPU**	**Server CPU**	**Client CPU**	**Server CPU**	**Client CPU**	**Server CPU**
XML-Based	15.0s	6s	22.8s	8.4s	1087s	551s
Java	2.3s	0.8s	4s	1.1s	148s	212s
CORBA	3.2s	1.9s	3.6s	2.1s	54.2s	250s

8.3.1.3 Conclusion

The performance restrictions linked to the usage of an XML-Based solution are significantly reduced, if not solved, compared to the previous studies performed on that topic. Even though it is still a slower implementation than Java or CORBA, the gap has been reduced.

It can even be reduced further by increasing the CPU and Memory configurations.

8.3.2 XML-Based Solution Set Real-Time Applications capabilities

This study is the analysis of a white paper, written by Steve Orobec from British Telecom, on behalf of the MTOSI working group, see Ref [9].

8.3.2.1 Introduction and context of this study

8.3.2.1.1 Scope

This white paper is investigating the ability of an XML-Based Solution Set to handle Real-Time applications and in particular Fault Management.

The goal is not to sell or promote such or such toolkit that are used in this study but to provide a purely technical study from which a clear conclusion can be issued regarding the Real-Time capabilities of XML-Based Solution Set.

8.3.2.1.2 Testing environment

This study was done using open source software, freely available. For this purpose,

One single CPU 2.4 GHz single core 32 bit Pentium laptop with 1GB RAM and Windows XP SP-1, and one single CPU 2.5 GHz single core 32 bit Athlon XP Desktop with 1 GB RAM and Windows XP SP-2 were used as Client and Server respectively.

Glassfish V2b03 nightly (Glassfish V1.0 is the open source ESB for JEE5 and is the java reference implementation covering JAX-WS, JAX-B, JAX-P, EJB 3.0 and supporting HTTP,

Derby Database,

54Mbit/s WiFi Router LAN.

The alarms flowing the system in the context of those tests are 3.75Kb which is realistic when talking about alarms.

It is very important to notice that none of the development and the tools (database) were tuned to optimize the results of this study. As well, the tools used are potentially not the one offering the best performances aspects. Once again, the goal was to set up a test environment neutral enough to allow the reader to make his own mind concerning the Real-Time capabilities of an XML-Based SS.

8.3.2.2 Method and Testing Results

The testing took place in three phases:

The first step was to measure the sustainable alarm flow, with no persistency for the alarms. The client (EMS side) was then set up to perform a pre-defined number of iteration (2000). The client (EMS) posts alarm events and the glassfish decodes the SOAP message to provide two java objects, as parameters, at the notification service method (MTOSI header and body respectively). These are manipulated in the same way as any java object can be.

Using default settings (no tuning), it was found that over 3 or 4 batches of 2000 iterations, performance increases from 110 transactions per seconds (after 2000) to a peak of 143 transactions when sending up to 8000 iterations. It has to be reminded that no tuning whatsoever was done!

The second run of tests was to add persistency of the data. Other than that, the same test methodology was used.

However, saving the alarm in a persistent database cost dearly to alarm flow that can be sustained. The average, in this configuration is from 25 to 36 transactions per seconds. Mainly, it is writing in the database that has a dear cost since it is estimated to 21 ms which is about three times the time for decoding the actual SOAP/XML messages. Once again, no tuning was done, in particular on the database.

Finally, it was decided to use a third PC as a dedicated Application Server.

This significantly increased the performances. Over 11000 transactions, we reached a rate of 52.6 transactions per seconds while the CPU load on the server was constantly between 2 and 4%. Concerning the LAN utilization, it averaged to 0.08%.

8.3.2.3 Questions relating to the Performance studies

Question 1: Does the alarm rates and times to process include the parsing of the alarms ?

Answer 1: Steve Orobec which performs the study confirmed that the times quoted are to take the soap wrapper and open it, de-serialise the XML to java and then to write to a database on the same pc as the application server. There is

obviously going to be a significant overhead in doing an i/o write to disk but the actual soap/xml de-serialisation seems quite quick after which the servlet has access to a concrete java representation of an alarm (i.e. you can get/set it).

Question 2: Concerning the batch of iterations that were used to hit the system, are they always the same alarms or did you have a pool of different alarms that were thrown randomly across the system?

Answer 2: Steve built the application in 2 modes, single shot (to support the MTOSI fault interface in which we can populate the database with different types of alarm for retrieval. However, there is still a need to do a little bit of work on the fault interface) and batch for performance tests for which the same alarms were used. The reason for using the same alarm is as follows: Steve was examining the performance of the application server receiving the alarm to de-serialise and persist rather than the ability of a client to instantiate and send the message. It was quicker/easier client software wise to send the same alarm in a loop. There would not be any difference if the same or different alarms would be used since the app server container generates a new 'entity manager' for each http request i.e. there is no caching (This is what the java community claim and Steve verified as such by applying a caching service locator pattern).

Question 3: Did you run your testing aver a significant amount of time? For example, did you consider the case where you would run/send a batch of alarms for a 24 hours period?

Answer 3: Steve just selected what he thought was a reasonable batch size. He could do more but the issue being that he didn't have any special equipment other than his work laptop, home pc and wifi router. The database he used wasn't very fast (it's main purpose is prototyping). Anecdotal comments say that Postgress sql is twice as fast. Steve may be able to get his hands on an old sun platform with multiple cpu if he can find office space for it. Because of the way java works performance increases over time since the jvm optimizes its byte code (JIT/Hotspot - also I used no performance tweaks)

8.3.2.4 Conclusion

The testing session was undertaken with non optimal development environment and, really, represents a worse case scenario.

The aim of this test was not to prove that an XML-Based Solution Set was the fastest technology but rather to verify that they could support reasonable real life performance requirements. This has been smoothly proven by the figures obtained by the testing.

Finally, further performance increases are likely to be met by the use of a faster customized database, the use of sc-si disks rather than PC-ATI disks and the use of multiple dual core 64 kbit CPUs.

Globally, initial findings indicate that SOAP/XML can easily meet the performance requirements asked by any networks.

9 Market status

Using a flexible and extensible feature, based on XML schema is easy to set up and therefore necessitates a shorter development cycle. As well, due to the fact that XML-based solutions do not need to be compiled, once again, it is very convenient to use and making changes don’t have the same cost than a solution for which compiling is necessary each time a change is to be made.

There is not really such things as companies providing services to the XML-based technologies unlike in the CORBA world where you can have companies such as Orbix providing those services. In the XML-based technologies, there are no tools per say, excepted the parsers. The services at developed independently.

10 Conclusion and recommendations

This XML-Based Solution Set Feasibility Study is going through all the main aspects that a technology has to comply with if it wants to be chosen and used. Some issues were raised during the life cycle of this study and today, most of them have been answered and closed in the core of the document.

In particular, in terms of performances a big effort was deployed to answer positively to the existing doubts surrounding XML-Based Solutions.

However, today, one more key aspect to be considered concerning performances of such systems is still under study. It concerns the ability for XML-Based technologies to handle in an acceptable way, real-time high volume notifications, such as "alarm storm". This is under study and the scenario to be tested is 40 notifications per seconds, random alarms over a ten minutes period of time. The outputs of this study are expected very shortly.

It appears clearly, through this study, that XML-Based IRP Solution Set, based on SOAP, at least as a wrapper, is the solution picked going forward by the IT/Telecom industry since it is recognized by a number of industrial sectors including many Telecommunication operators, vendors and organisations.

As a result of this feasibility study, the following points are agreed and accepted if 3GPP SA5 were to go ahead with an XML-Based Solution Set type:

- XML-Based Solution Set would not introduce any significant restrictions when compared to existing and used Solution Sets. However, the ability to handle large volume real-time notifications is still to be demonstrated,

- XML-based Solution Sets introduce fundamental and easy implementation concepts such as flexibility and extensibility, and independence on the transport layer. However, the study has reservation about extensibility in terms of backward compatibility and forward compatibility. Currently, W3C has recognized the importance of such capability and is studying this aspect pending conclusion,

- Moving to XML-Based Solution Set is clearly a vehicle to use towards harmonisation in the Telecom Industry,

This Feasibility Study indicates that SA5 goes ahead with the use of XML-Based Solution Sets. The introduction of such technologies is beyond 3GPP Release 7. For a start a particular focus should be set on Service Management but this shouldn't be restrictive. Starting with Service Management implementation should allow a smooth transition and to gain a know-how experience.

Annex A:
Example of SOAP message

SOAP Message Header:

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">  env is the namespace name
    <env:Header>                                                                    optional element
        <m:reservation xmlns:m="…"  env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
                    env:mustUnderstand="true">                                          child
element, header block, with
                    attribute: role,
                                                        mustUnderstand,encodingStyle, relay
            <m:priority>1</m:priority>                                                  character or
element infor element item
            <m:expires>2001-11-29T13:20:00.000-05:00</n:expires>
        </m:reservation>
    </env:Header>
```

SOAP Message Body:

```
    <env:Body>                                                                      mandatory
element
        <m:alert xmlns:m="http://example.org/alert">                            character infor item
child element
            <m:msg>Pick up Mary at school at 2pm</m:msg>                    character or element
infor element item
        </m:alert>
        <env:Fault>
        </env:Fault>
    </env:Body>
</env:Envelope>
```

Annex B:
Links to more literature concerning XML-based technologies

General:

http://www.bitpipe.com/rlist/term/Web-Services-(Software).html

http://www.globusworld.org/program/abstract.php?id=80

http://soapclient.com/papers.html

http://www.developers.net/tsearch

http://www.webservicesarchitect.com/

Performances:

http://www.orchestranetworks.com/fr/soa/contraintesws.cfm

http://www.orchestranetworks.com/fr/soa/extraitws.cfm

Annex C:
Change history

Change history								
Date	**TSG #**	**TSG Doc.**	**CR**	**R**	**Subject/Comment**	**Cat**	**Old**	**New**
Sep 2006	SA_33	SP-060566	--	--	Submitted to TSG SA#33 for Information	--	--	1.0.0
Dec 2006	SA_34	SP-060755	--	--	Submitted to TSG SA#34 for Approval	--		7.0.0