# Portable and Efficient Replacement for Web Services Messaging Protocol

[*]Institute of Computer Science
Masaryk University Brno
Botanická 68a, Brno 602 00
Czech Republic

**Ondřej Krajíček**[*]
krajicek@ics.muni.cz

**Petr Holub**[+*]
hopet@ics.muni.cz

[+]CESNET z. s. p. o.
Zikova 4, 160 00 Praha 6
Czech Republic

## Motivation

Uses for distributed applications emerge in all areas of computing today. However, complexity of existing distributed systems has been preventing their wide adoption. Web Services technology aims to offer simple and portable solution for building distributed applications.

### XML Web Services Technology

XML Web Services technology comprises standards, which cover the main aspects of service oriented application architecture and service invocation:

- description of service interface and bindings: WSDL
- data encoding and serialisation: XML Schema
- encoding data into messages and message transmission: SOAP
- service discovery and integration: UDDI

Due to the XML background of these technologies, several drawbacks are imposed, which may not be obvious for the first sight and yet they may be an issue in certain environments:

- **problem 1** (data presentation problem): with textual protocol, the data must be converted from their internal representation (not necessarily arbitrary) into its textual representation.
- **problem 2** (messaging overhead problem): with XML-based protocol, the message must be constructed as valid XML document imposing unnecessary overhead.
- **problem 3** (transmission overhead problem): the message must be transmitted as a whole (i.e. fully constructed in the memory) due to the nature of the transport protocol.
- **problem 4** (stateless communication model): the communication model of XML Web Services is stateless. For applications requiring stateful communication, the state is encoded in every message transmitted or stored explicitly by both communicating partners.
- **problem 5** (one-way communication model): the communication stemming from web service invocation is always client initiated.

## Near-Real World Example

Thermometer service represents a simple agent which implements temperature monitoring in a temperature-sensitive environment. A method implementing the service could have the following prototype:

```
float[] GetTemperature (int id, DateTime from, DateTime to);
```

Invocation of our example service may result in this SOAP response (truncated):

```
<?xml version="1.0" encoding="utf-8" ?>
<ArrayOfFloat xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://tempuri.org/">
  <float>16.4567432</float>
  <float>36.21009</float>
  ...
  <float>33.59396</float>
</ArrayOfFloat>
```

Each representing one temperature measurement. Binary encoded, the example response containing one thousand elements could be approx. 4 kB in size (each float taking 4 bytes). SOAP message that contains essentially the same data is in this case more than ten times larger.

### Problem Overview

Bandwidth consumption is one of the most painful problems of XML Web Services in environments where bandwidth is limited or expensive. However, in processing-intensive applications (GRID computing, High-Performance Computing) processing resources (CPU, memory) consumption may be even more significant.

Due to the XML nature of the SOAP protocol, unnecessary data serialisation and messaging overhead are imposed. Data must be serialised into its textual representation, the message must be constructed as a whole in memory and transmitted at once. With SOAP, these drawbacks seem unavoidable, at least without introducing binary extensions and breaking some of the fundamental principles of XML and related technologies.

## GSIP – Generic Service Invocation Protocol

Instead of building performance-enhancing extensions for the SOAP protocol, GSIP protocol takes completely different approach. It is new, alternative messaging protocol for XML Web Services, which is, unlike SOAP, binary in its nature.

### GSIP Architecture

Following figure shows layered Web Service architecture comparing SOAP and GSIP protocols.

| |
|---|
| Web Service Application |
| WSDL bindings |
| SOAP |
| HTTP, HTTP[SG],... |
| TCP |
| IP |

| | | |
|---|---|---|
| Web Service Application | | |
| WSDL bindings | | |
| GSIP | | |
| HTTP[SG]*,... | TCP | UDP |
| TCP | | |
| IP | | |

**WSDL Bindings** layer provides transparent proxy interface and binding and serialisation stub which implements remote invocation of Web Service. Binding and serialisation stubs are generated from WSDL service specification (contract) by a **stub generator**. Bindings layer introduces a notion of **data units** – atomic chunks of serialised data, which are packaged and transmitted across the wire to the other communication endpoint.

**GSIP Protocol** layer consists of two parts: **messaging** and **channels**. Messaging part encapsulates data units into messages for transmission and decodes data units from received messages. Channels provide transparent interface to the transport protocols with notion of transport channels. Transport channels are characterised by certain properties (reliability, blocking operation, quality of service) and are provided by the protocol implementation based on policies specified by the WSDL contract (policy definition extensions for WSDL need to be created).

### GSIP Stubs and Stub Generators

GSIP invocation stub (or just "stub") consists of three parts: the binding stub, the serialisation stub (the two are generated automagically using the WSDL contract) and the skeleton, which implements functionality common to all stubs.

#### Binding stub

Binding stub publishes call-level client/server interface and implements the remote invocation functionality. Binding stub serialises data and packages them into data units using serialisation stub, and passes them to messaging implementation for further processing.

Binding stub implements:

- **Data serialisation** using the serialisation stub
- **Policy creation and handling** used to configure the underlying messaging and channels. Policies are declared in WSDL contract.
- **Data mapping** which maps data structures into data units.

#### Serialisation stub

Serialisation stub implements data serialisation. Data mapping is a process which maps serialised data into data units. Data mapping is controlled by policies, which may be specified (instantiated) by the binding stub.

Serialisation stub is fundamentally ASN.1 parser. Type notation for the parser is automatically created by the stub generator from the WSDL contract. This process is essentially mapping XML Schema to ASN.1 schema (XML Schema mapping is defined in ITU-T X.694). The stub generator also creates data structure declaration for the target programming language.

### Problems addressed by GSIP

GSIP addresses the abovementionted problems with XML Web Services. Short overview of solutions for problems mentioned above is given hereby:

- **solution 1** (data presentation problem): GSIP is a binary protocol, data are encoded using ASN.1 Encoding Rules which impose significantly smaller overhead than textual encoding (for performance comparison of ASN.1 binary and textual encoding see `http://www.asn1.org/benchmark/`).
- **solution 2** (messaging overhead problem): with GSIP, message is constructed using data units. Message itself contains no explicit header while data and meta-data are all carried within the data units.
- **solution 3** (transmission overhead problem): the stub may begin message transmission when at least one data unit is available. Data that has to be transmitted may be serialised as more than one data unit, i. e. multiple messages may (but need not) be used for data transmission. The messaging behavior depends heavily on protocol implementation and involved policies.
- **solution 4** (stateless communication model): the channels layer, which encapsulates messaging transport and communication, may provide means for keep-alive communication. The communication state may be embedded within channel instance and transparently bound to each message as data unit.
- **solution 5** (one-way communication model): more sophisticated messaging models may be easily implemented (message push model, publisher-subscriber model).

### References

- Kenneth Chiu, Madhusudhan Govindaraju, Randall Bramley, *Investigating the Limits of SOAP Performance for Scientific Computing*, Proceedings of The Eleventh International Symposium on High Performance Distributed Computing, IEEE Computer Society Press, pp. 246-254, Edinburgh, Scotland, 23-26 July, 2002.
- Paul Sandoz, Santiago Pericas-Geertsen, Kohuske Kawaguchi, Marc Hadley, and Eduardo Pelegri-Llopart, *Fast Web Services*, Sun Microsystems, August 2003 `http://developer.java.sun.com/developer/technicalArticles/WebServices/fastWS/`

**Motto**:
*Web Services have layers.*
*Ogres have layers.*
*Ogres are like Web Services.*
*Everybody likes Web Services.*
*I don't care what everybody likes...*

typeset with ConTEXt