Efficiency of compression techniques in SOAP¹

TOMASZ PODLASEK, JOANNA ŚLIWA, MAREK AMANOWICZ
Military Communication Institute, 05-130 Zegrze, Poland

Abstract: The article describes the comparison of compression techniques, which increase SOAP efficiency in a disadvantaged environment. The authors present the functionality of three mechanisms – Gzip compression algorithm, Fast Infoset and Efficient XML Interchange encodings. Particular attention is paid to the EXI standard that gives satisfactory results when comparing to the other two. The main part of the article presents results of experiments that were carried out in experimental environment with test implementations. In this part the authors provide a short description of services used for testing, as well as the architecture and configuration of the test bed. The results of experiments are shown in graphical charts, grouped by message sizes. These charts show the compression ratio of the SOAP messages and the response time for service call. Collected results were useful for determining what is the most efficient way to send SOAP messages in disadvantaged networks.

Keywords: SOA, tactical networks, adaptation, dynamic proxy, ontology, reasoning

1. Introduction

SOAP is the crucial transport protocol used by Web Services (WS). Contemporary Web Services provide a wide spectrum of interoperability, platform and language independence, are self-contained and easy to develop. What is important, they are well standardized by e.g. W3C (Word Wide Web Consortium) and OASIS (Organization for the Advancement of Structured Information Standards) – organizations that take care of their fast and advanced development. These advantages make web services more useful in business processes, company’s management and multimedia distribution.

Web services are usually realized by the exchange of SOAP messages [12]. SOAP is based on the eXtensible Markup Language (XML) [28] and has been designed to operate in high bandwidth links. XML has become very popular because it solves many interoperability problems, even though it adds significant overhead related to the size of messages and the XML processing and parsing time. Unfortunately, such solutions need a high capacity telecommunications channel in order to efficiently exchange SOAP messages.

¹ Within the project supported by Polish R&D funds allocated for the years 2007-2010
Originally Web Services were designed to be used in overprovisioned, high bandwidth enterprise networks. They do work very well in military environment, on the operational and strategic levels, however there are some constraints and limitations in their utilization in disadvantaged networks. In order to be able to use SOAP web service calls in such networks, first of all we must diminish the size of the stream between the service and the client.

The goal of our work was to test how web services may be integrated with compression mechanisms and what is the gain of using them in a disadvantaged environment. Based on results of the experiments we compared selected tools, used to make the SOAP lighter and transport it in the most efficient way by wireless communications links. These tests are to show, to what extent SOAP can be used in the wireless network that is characterized by low bandwidth, high delays and packet error rate.

2. Compression and binary coding

In the experiments we concentrated on using Gzip, Fast Infoset (FI) and Efficient XML Interchange (EXI) implementations to compress (Gzip) [10] or binarily encode (FI, EXI) the SOAP messages.

2.1. Gzip compression

In general, compression techniques can be divided into "lossy" and lossless ones. In the case of data transmission it is necessary to use only the lossless ones. These compression algorithms usually exploit statistical redundancy in such way as to represent the sender's data more concisely without error. Lossless compression uses the fact that most real-world data has statistical redundancy. However, lossless data compression algorithms will always fail to compress data that has high entropy (high disorder). This applies to already compressed data, random data or encrypted data. In such cases attempts to compress data will usually result in an expansion.

Gzip is a lossless compression technique which was created as a fusion of two algorithms: Lempe-Ziv (LZ77) and Huffman Coding [10]. It is based on the DEFLATE algorithm, which was designed to replace LZW and other patented data compression. It finds duplicated strings in the data to be compressed. The second occurrence of a particular string is replaced by a pointer to the previous one, in the form of a pair represented by the distance and length. Distances are limited to 32K bytes and lengths are limited to 258 bytes. When a string does not occur anywhere in the previous 32K bytes, it is represented as a sequence of accurate bytes.
This characteristic of Gzip indicates that it can be satisfactory only with data that has some order. This makes it inappropriate e.g. for already compressed files and encrypted ones.

Gzip in fact stands for GNU zip [10]. This name derives from the software development world since Gzip has been adopted by the GNU project and published to be used on the Internet. It has been integrated e.g. with widely used commercial application servers and can be turned on easily for the return web services traffic.

In terms of Java programming language support, standard Gzip library from Java Virtual Machine (JVM) cannot be used on mobile devices, because Java 2 Mobile Edition (J2ME) specification is very constrained in comparison to Java 2 Standard Edition (J2SE). Currently there are a few implementations of light Gzip such as Tiny Line [17] and Gzip [11]. These implementations theoretically may be used on mobile devices, where J2ME [16,21] specification is supported. Nevertheless in mobile devices contemporary light implementations of Gzip may only be used for data compression over HTTP protocol.

2.2. Fast Infoset

Fast Infoset specifies a standardized binary encoding for the XML Information Set [26]. It uses the existing and proven ASN.1 standards. The specification is standardized as an ITU-T Recommendation within ITU-T SG 17 and ISO/IEC JTC1 SC6.

The ASN.1 types and components in the ASN.1 module describe information items and properties of those items. According to [6,19] a fast infoset document is an encoding of a fast infoset value (an ASN.1 value), whose ASN.1 type, defined in the ASN.1 module, corresponds to the document information item. The default encoding of a fast infoset document uses the Packed Encoding Rules with extensions.

There is a Fast Infoset implementation for J2ME (FIME) [7] – a light implementation [7,16,19] which is generally only relevant to J2SE (FISE). This project is still under development which is to improve the implementation and provide a friendly built environment.

2.3. Efficient XML Interchange

Efficient XML Interchange (EXI) [2,3] is a specification of binary coding of the XML data. EXI is a very compact representation for the XML Information Set that is intended to simultaneously optimize performance and utilization of computational resources. The EXI format uses a hybrid approach drawn from information and formal language theories, plus practical techniques verified by measurements, for entropy encoding of XML Information. To efficiently encode XML event streams, the EXI format is using a relatively simple algorithm, which is amenable to fast and compact implementation, and a small set of data type rep-
representations. The EXI specification consists of the grammar production system and the format definition. EXI is compatible with XML at the XML Information Set level, rather than the XML syntax level. This permits it to encapsulate an efficient alternative syntax and grammar for XML, while facilitating at least the potential for minimizing the impact on XML applications' interoperability. EXI is schema-"informed", which allows utilizing the available schema information to improve compactness and performance. It also uses a grammar-driven approach that achieves very efficient encodings.

Figure 1. The concept of encoding XML to EXI format

Figure 1 depicts an algorithm of processing the XML encoding to EXI format. The parsing result is encoded according to the grammar onto a smaller, lower entropy stream of events. Additionally, the user may set any options to customize additional information such as schema less document, data block size, compression, etc. These options make EXI more flexible and useful for the user.
EXI has many options reflected in the EXI header [3]. They are represented as an EXI Options document, which is an XML document encoded using the EXI format. EXI Processor observes the specified options to process the EXI stream accordingly. The EXI options are not supported by the open source EXIfficient implementation [4].

There is a commercial implementation of EXI [2,3] standard, the most extended one which offers native EXI support to popular commercial and open source Web Services platforms and clients. Furthermore, it provides support to a wide group of existing XML standards such as: DOM, SAX, StAX, XML Schema, etc. Another EXI implementation is EXIfficient [4], an open source implementation of W3C Efficient XML Interchange (EXI) format specification, written in the Java programming language that was used for our tests.

3. Experiments

The three aforementioned mechanisms were evaluated in terms of their performance. The mediation service [22] which processes information according to NATO Friendly Forces Information (NFFI) XML Schema [21] was used for integration of compression techniques. That service is also consistent with SOAP 1.2 [14], WSDL 2.0 [24] and XML Schema [26] and was implemented in Java programming language by the use of JAX-WS [15] API. Some of these extensions were also coded in Axis2 [1]. Services were hosted on Glassfish and Tomcat application servers [11,12,19]. The implementation of the mobile application was done in Java programming language according to J2ME [17] restrictions and JAX-RPC [16] API to create the skeleton of the mobile client.

In order to test the efficiency of techniques in a disadvantaged environment we used WANem [23] network emulator, which was configured to degrade delay and packet error rate (PER).

The server device was a PC with a Core2 Duo E7400 2.8 GHz processor, 2 GB RAM, Windows XP SP3 with JEE SDK provided from Oracle.

As the web service client we used the laptop – Lenovo X300, Core2 Duo L7100 1.20 GHz processor, 2 GB RAM, Windows XP SP3 with standard JDK runtime version 1.6.16 provided from Oracle. For the purpose of creating a web service client the JAX-WS API 2.1 [13] was used.

The test was performed for 9.6 kbps bandwidth, for small and medium NFFI messages. The number of tracks was equal 1 for a small message and 30 tracks – for medium SOAP messages.

In our tests we assumed that the NFFI message with 1 track/object may be treated as a small SOAP message, and the message with 30 tracks may be treated as a medium SOAP message.
The delay of the link was equal to 0, 100, 500 ms and the packet error rate was equal to 0, and 25% for each step. In those measurements a symmetric link between the client application and the server was set up.

3.1. The scenario

The experiments were to measure the compression ratio for different kinds of SOAP messages (small, medium, big) and the response time when transmitted through the disadvantaged network. The tests were divided into two parts measuring:

- the message size after compression,
- the response time for each service call.

Wireshark Network Analyzer was used for measurements.

3.2. Implementation details

Efficient XML used for tests was a Siemens open source EXI implementation. During tests, before the EXI [4] encoding of the SOAP envelope, we needed to retrieve the output SOAP message. There are several ways to do it, e.g. AXIS2 API and JAX-WS API. Otherwise, the EXI encoding engine should be informed about the XML Schema of the SOAP message. After this step, on the server side, we must use the following libraries to parse and transform the SOAP message into a set of SAX events.

For parsing the SOAP message into a set of SAX events, the Java API for XML Processing (JAXP) 1.3: xml-apis.jar, and xercesImpl.jar was used. For transformation of the SOAP messages into a set of SAX events we used the Xalan [25] – xalan.jar, serialize.jar, and xercesImpl.jar libraries. These libraries are needed to support SAX, which was used in internal EXI engine. Because the Xalan implements the javax.xml.transform interface from JAXP 1.3, it should be used to perform SOAP message transformations. Both the Xalan [25] and JAXP libraries are a part of the JAX-WS API.

In JAX-WS a good choice to integrate EXI is to apply the mechanism for inbound and outbound SOAP messages or to use the handler mechanism.

In order to use EXI on mobile devices, from the implementation point of view, the main challenge is a multiplicity of software solutions. There are some operating systems which support to some extent the core libraries to handle SOAP. In order to integrate EXI [4] with web service client on mobile devices, the developer must install the needed libraries, like XSLTC [29]. The XSLTC is a part of Xalan and provides a compiler and a runtime processor for XML-kind data.

In our proof of concept for mobile devices we used the Android. In order to create a web service client which can consume the EXI encoded messages on the device supporting J2ME and Android, we must make sure that Xalan and JAXP libraries were installed. The next step is to install the EXI libraries on the mobile device and integrate them with runtime libraries mentioned above.
From the service side it is only necessary to make small modifications to XML Schema description file. Such files define the structure of messages exchanged with the client. In each file of that type declarations of namespaces (such as e.g. xmlns:S=""http://schemas.xmlsoap.org/soap/envelope/"") must be added in the root declaration section.

Gzip integration was implemented in two ways. The first used the filter web service mechanism to integrate Gzip with the test bed, and the second used Gzip compression supported directly by application servers like Glassfish and Tomcat [9,18]. The two solutions gave very similar results.

On the mobile devices the integration of Gzip was done with Tiny Line [17] library which perfectly accompanies the J2ME and Android devices.

Fast Infoset implementation on applications servers and standalone units is integrated with JAX-WS, and AXIS2 APIs [1,6,15] by the vendors, but implementation on mobile devices called FIME [7] could only be installed on devices that use the J2ME libraries [14,16].

3.3. Results

The results of experiments compare the efficiency of standard SOAP message exchange with adaptation techniques, Gzip, FI and EXI with deflate compression turned on.

The results are shown in graphical charts, grouped by message sizes. These charts show the compression ratio of the SOAP messages and the response time for service call. Collected results will be useful for determining what is the most efficient way to send SOAP messages.

The first scenario was to measure the compression ratio after encoding it with EXI, FI, or compressed with Gzip algorithms.
Figure 2 and 3 present the level of compression of the small and medium SOAP messages. For such messages, EXI schema-oriented and deflate compressed messages have the smallest size among all messages (78%).

![Figure 3. EXI vs. Gzip vs. FI in reference to medium SOAP/XML message](image)

The second scenario was to measure response time for invoking operations on the test service from the client application with different values of network degradation. For every value of response time we had taken 15 measurements and calculated their average.

Figure 4 presents the response times for invoking operations with small SOAP message size in the network with 10 kbps and 0% PER. Figure 5 presents the response times for the same SOAP message, but larger PER (10 kbps and 25% PER).
Figure 4. EXI vs. Gzip vs. FI response times for web service operation
(1 object in the message, 10 kbps, 0% PER)

Figure 5. EXI vs. Gzip vs. FI response times for web service operation
(1 object in the message, 10 kbps, 25% PER)

In the next scenario we measured the response times for invoking operations
with medium-size messages.
According to results depicted in Figure 6, the advantage of FI over SOAP is over 50%. More detailed comparison of the EXI and Gzip was shown in Figure 7. It shows that the EXI is slightly better than Gzip, given that the EXI message sizes are significantly lower than Gzip compressed messages.

Tests performed with packet error rate equal to 25% (see Fig. 8) show that there was no possibility to successfully call operations on service using SOAP or FI. We can
present only the results for compression and EXI measurements, because in these network conditions only these methods were successful (response was returned).

![Graph showing response time for 30 objects in message, 10 kbps bandwidth, 25% packet error rate](image)

**Figure 8.** EXI vs. Gzip response time for invoked web service operation

## 4. Summary and conclusions

The article describes the comparison of compression techniques, which increase the SOAP efficiency in a disadvantaged environment and presents the functionality of three mechanisms – Gzip compression algorithm, Fast Infoset and Efficient XML Interchange encodings.

The measured web service response time for EXI was shorter than for Gzip, but we expected greater difference for EXI solution. It may be caused by some deficiency in the open source EXI implementation we used [4]. The compression ratio we measured proved that EXI can significantly limit the size of SOAP messages (95% for medium messages), however Gzip has also very good results (94% for medium messages). FI binary encoded messages gave worse results (53% for medium messages).

In order to prepare the test bed the EXI open source implementation had to be integrated with web services created in Axis2, JAX-WS. We integrated EXI with web service client applications. In order to do so, we used the Xalan, JAXP API, also on mobile devices.

When the message is EXI encoded and compressed with deflate compression algorithm, it should not be compressed with Gzip algorithm, because the size of that message is obviously bigger then the initial size (the increase was about 20% for small, and 25% for medium messages).

For the integration of Gzip and FI with web services we used the existing open source solutions. Achieved results prove that the efficiency of Fast Infoset is the worst as compared to the other 2 techniques.
The results proved that the Efficient XML standard is very promising, however its efficiency strongly depends on implementation. It is clearly better than FI in a disadvantaged environment, where the delay and packet error rate are high. Our tests indicate that its advantage over Gzip is not so big. Small messages have significantly bigger compression gain for EXI than for Gzip. With the medium messages the difference was not so big. In order to make conclusions in terms of using investigated mechanisms, data processing time must be also taken into account. According to our results, the open source EXI processing time is a little bit smaller that Gzip, however it increases when the messages are getting bigger. This can result in very similar results for Gzip and EXI in disadvantaged networks.

In order to fully evaluate the EXI standard it is necessary to make comparison tests of open source implementation against the commercial one, provided by Agile Delta [5].

REFERENCES


