USING BIT-EFFICIENT XML TO OPTIMIZE DATA TRANSFER OF XFORMS-BASED MOBILE SERVICES

Jaakko Kangasharju
Helsinki University of Technology, PO Box 5400, 02015 TKK, Finland

Oskari Koskimies
Nokia Research Center, Itämerenkatu 11–13, 00180 Helsinki, Finland

Keywords: Mobile XForms Services, Binary XML Serialization.

Abstract: We consider here the case of XForms applications on small mobile devices. The aim is to find out whether a schema-aware binary XML format is better suited to this area than generic compression applied to regular XML. We begin by limiting the potential areas of improvement through considering the features of the binary format, and then proceed to measure effects in the identified areas to determine whether a binary format would be effective.

1 INTRODUCTION

The ServiceSphere project at Nokia Research Center has focused on researching mobile service solutions for small and medium enterprises (SMEs) in different business domains. One of the goals has been to study the benefits of software as a service paradigm (Traudt and Konary, 2005) that has gained a big momentum in the fixed Internet-based enterprise solutions.

Increasing capabilities of cellular mobile devices open possibilities for developing mobile applications and services in many different business domains. Better UI, larger memory, integrated peripherals like camera and voice recorder, and wirelessly connected peripherals like GPS and bar code readers make it possible to use generic mobile phones for applications that used to require custom-made, integrated systems.

Development of mobile service implementations in such an environment has requirements for extreme agility, low overhead, rapid prototyping capability, and high enough quality that the services could be deployed in a commercial environment for trial use. One of the technologies we selected based on these requirements was XForms (W3C, 2006). We have implemented a prototype mobile-optimized XForms processor that allows us to easily reconfigure the user interface of the mobile device. As a consequence of using XForms, all client-server communication is done in XML, which also makes it easier to design flexible server-side components. However, XML is a relatively verbose format and in some domains the amount of data transmitted from client to server each day can be considerable. When operators charge for data transfer on a per-kilobyte basis, the question of efficient XML interchange becomes important for the commercial viability of a mobile service.

In this paper, we analyze data from two different domains, both based on concrete trials we have performed. The first domain is customer relationship management (CRM), where a company representative would regularly visit customers and use a mobile device to enter detailed data about their current operations. In this case there were two separate data sets, one which was sent on the first visit and contained basic customer information, and another which was sent on subsequent visits and contained information about the current operations of the customer. The second domain is road maintenance, where a mobile device was used to track road maintenance tasks, collect history data, create reports, and monitor the cost of activities. A single data set was used. It contained information about the task at hand, the route the maintenance crew was taking, and consumed materials. Our experiences from the road maintenance domain have been documented in detail in (Karhinen et al., 2007).

The data sets from the two domains are representative of two different content structure types, repeating and non-repeating. As such, they capture well the

Kangasharju J. and Koskimies O. (2008).
USING BIT-EFFICIENT XML TO OPTIMIZE DATA TRANSFER OF XFORMS-BASED MOBILE SERVICES.
In *Proceedings of the Tenth International Conference on Enterprise Information Systems - SAIC*, pages 5-11
DOI: 10.5220/0001683800050011

variance in data sets that most affects compression. In the CRM domain, the structure is non-repeating, meaning that the data set consists of many different elements without significant repetition. In the road maintenance domain, the structure is repeating, meaning that repetition of the same elements takes up a significant portion of the data set. The reason for the repeating structure in the road maintenance domain is that the route of the maintenance crew is recorded as a long list of coordinates.

Sample forms and data sets (purged of confidential information) were given for analysis to the Fuego Core research team at the Helsinki Institute for Information Technology, who specialize in efficient XML processing for mobile devices. They concluded that by taking advantage of the known document structure, advanced bit-efficient XML representations can provide significant advantages when compared to generic compression algorithms. The rest of this paper covers the Fuego Core analysis in more detail.

We begin this paper with a more detailed problem statement in Section 2. A brief overview of the main features of the Xebu binary XML format that we used is given in Section 3. The purpose of the section is to introduce Xebu sufficiently well to allow following the rest of this document, so no technical details are provided. Section 4 considers the scenario on a high level, looking at the features of Xebu to determine how to best use it in the scenario and in what specific ways. Section 5 provides sample measurements to determine the effectiveness of Xebu in the problem domain. Section 6 lists some conclusions on the feasibility based on the previous sections. Finally, Section 7 outlines some future work that could be done on Xebu to make it a better fit for a variety of applications, especially this one.

2 PROBLEM STATEMENT

XForms (W3C, 2006) is a useful language for specifying interactive XML-based applications for distributed computing. However, when considering small mobile devices that could definitely benefit from a standardized, truly user-interface-agnostic language, the use of XML raises some questions. The chief among these is XML's verbosity, which causes a large amount of network bandwidth to be used in communication.

In an XForms application, there are two kinds of documents. The form document follows the XForms schema, and contains the data model and presentation logic of the application. The data model also includes the specification of what data the user is expected to provide, and a document template for submitting the data.

After a user has filled the requisite information on the form, the application then needs to send the information to the server. This is done by filling the user-provided information into the template provided in the form, and sending the resulting XML document over whatever protocol the application uses to communicate.

To mitigate the effect of XML's verbosity, the obvious solution is to apply some form of compression to it before sending it over the network. Common existing protocols such as HTTP (Fielding et al., 1999) already support indicating the use of generic compression like gzip (Deutsch, 1996). Since XML is text, and highly-redundant text at that, generic compression algorithms usually perform acceptably well.

There are, however, two potential issues with generic compression over XML when applied to XForms applications. One is that compression takes time, and when this gets added to the already significant time needed to process XML, the amount of required processing may become prohibitive. The larger problem is that the amount of data may in many cases be quite small, so generic compression that is based on redundancy in the data will not perform very well.

Because of these, and other, reasons, there have been several proposals for *binary XML* formats (Pericas-Geertsen, 2003; W3C, 2003; W3C, 2005). Such a format is a replacement for XML that is usually intended to be a more compact representation as well as more efficiently processable. Two requirements for a general binary XML format are that it be able to represent any XML and that it be able to use available schema information (usually in the form of XML Schema (W3C, 2001a; W3C, 2001b)) to improve its compression ratio.

There are a large number of binary XML formats already in use. Well-known general-purpose formats include Fast Infoset (Sandoz et al., 2004) and XBIS (Sosnoski, 2003), but these are capable of using only a limited amount of schema information, i.e., they cannot take advantage of the structure information present in a schema. Better use of schema is provided by formats such as ASN.1 (ITU, 2004), BiM (Niedermeier et al., 2002), and Xenia (Werner et al., 2006), but these have the drawback that all documents must be schema-valid and usually good results are achieved only when the schema describes everything very precisely. The EXI format currently being developed at the W3C (W3C, 2007) has the ability to serialize any XML but also to use all the information available in a schema to improve its compression

ratio. EXI is also capable of serializing any document even when a schema is in use, but schema-valid documents will result in much smaller final documents.

3 XEBU OVERVIEW

Xebu (Kangasharju et al., 2005) is a binary XML format developed by one of the authors in the Fuego Core research project[1]. It has been designed to be usable on mobile phones as a part of a general XML-based middleware system. It consists of a basic format applicable to any XML and a few techniques that decrease document size further when a schema is available. The main reason why Xebu is attractive for this research is that it has a publicly-available, Open Source implementation[2] written for mobile phones that support the Java Mobile Information Device Profile (MIDP) 1.0, which includes most Java-enabled phones. This eliminates the effort needed for implementing a format processor.

The basic Xebu format, like Fast Infoset and XBIS, is based on *tokenization*, i.e., replacement of repeatedly-occurring pieces of content by small binary tokens. This is similar to generic compression methods, but in Xebu this tokenization happens at the XML level. The only candidates for tokenization are complete XML namespace URIs, XML names, attribute values, etc. By concentrating on the XML level, Xebu's speed is much faster than that of a byte-oriented generic compression algorithm.

In principle, Xebu has three different schema-based techniques that are each individually applicable:

1. Pre-tokenization establishes a set of token mappings beforehand based on the names and potential values that appear in a schema.
2. Typed content encoding uses a binary encoding for typed data values, such as integers, instead of encoding everything as strings like in XML.
3. Omission automata process XML event streams by leaving out completely events that are deducible from context, e.g., when the schema specifies a sequence of elements, the start tags of the elements are deducible from the previous element's end tag and can be omitted.

In the current implementation, typed content encoding is based on an extended XML API and the presence of `xs:type` attributes instead of determining the proper data type from the schema.

[1] `http://www.hiit.fi/fi/fc/`

[2] `http://hoslab.cs.helsinki.fi/homepages/xebu/`

The omission automaton technique, as it is defined, has one nice benefit. Namely, when the automata encounter an event that they do not recognize, it gets passed unchanged through the automata. This permits extending the schema by, e.g., adding new elements at certain places without hurting the compression ratio of the rest of the document. Also, deletions are possible, but with the current system the rest of the document will then potentially not benefit from the schema use.

However, the omission automata are not usefully applicable to some cases. They perform best for linear schemas, i.e., where elements follow each other in a predetermined sequence. In case of alternatives, i.e., where there are even two choices for the next event, they do not provide any additional compression due to the either-or nature of event omission. This is in contrast with techniques that have the choices built into the processing and have an indicator of which choice to take (Werner et al., 2006; W3C, 2007).

A specific, and potentially unexpected, instance of this issue is mixed content, since there each position has the option of containing either text or an element, so the omission automata do not handle mixed content very well. This mixed content also extends to the whitespace often used to indent XML document for easier readability and editability. This choice reflects Xebu's primary application area, since Xebu-using applications typically write XML through an API, where putting in additional whitespace is normally not expected.

4 INITIAL ASSESSMENT

In XForms applications, two kinds of XML data are being sent over the network. The client receives the form in a document that includes both the form data model and the form presentation logic. The data model also contains a document template for submitting the data. After filling in the information in the template, the client submits it to the server. This latter process can happen multiple times for the same form.

Looking at the server-to-client sending of the form, real-world applications usually require relatively large forms to capture the necessary presentation logic. The resulting documents are typically in the 10–50 kB size range, for which gzip probably performs quite well, and a binary format might not provide much additional compaction.

Furthermore, application of schema-based techniques to this document is not straightforward. The XForms schema is very free-form, allowing many different elements at each place. Therefore the omission

automata of Xebu would probably not provide much benefit in this case.

The situation is different for the submitted data. In this case the document is typically much smaller than in the other case, in the 1–10 kB size range, and therefore gzip does not achieve as high a compression ratio. But what is better, the schema in this case is often very linear, as the template provided in the form will have the elements in a specific order, which is then used when sending the filled-in form as well. Therefore the omission automata of Xebu should be well suited for this case.

There still remain the questions of how to best apply Xebu in the given situation. For one, there are several options for how to construct the omission automata and pre-tokenization tables from the schema. The current Xebu implementation can either create a description of both in a reasonably simple text format, or it can create Java classes that directly implement the tables and automata in a form understood by the Xebu implementation. Second, there is the choice of constructing these on the server side and transmitting to the client or having the client construct them.

In the specific application scenario, having the client construct the tables and automata from a schema seems like the less reasonable option. The client is not going to use the schema for anything other than these Xebu techniques, so having the schema available at the client seems like a waste. Furthermore, doing the generation on the client side just uses the client's resources. Finally, the option of having the server generate the tables and automata for both sides ensures that the client and server are interoperable, as there is no need to specify precisely the generation process.

There are drawbacks to this approach, though. The first one is that, in essence, it is creating a new schema language that would need to be supported by all implementations of Xebu. This might not be a large hurdle if Xebu is treated as an essentially proprietary format that is specified precisely by its implementation. The second concern is the size of the transmitted descriptions compared to the schema. While in most cases encountered so far the generated files are approximately the same size as the schemas, it could be possible that some schemas would generate much larger automata (Section 7 outlines ways to eliminate this even as a potential issue). However, for a simple linear schema this should not be a concern.

The current implementation of Xebu supports both the generation of individual Java classes for each schema as well as generic implementations of both the tokenization tables and omission automata. These generic implementations read a simple text-based format file that gives the contents of the tables and the transitions of both automata.

The choice between these approaches is probably dictated by the environment. The client will reside on a mobile phone, and MIDP permits loading only the system classes and classes contained in the application's suite. Therefore, when generating classes, it would not be possible to dynamically include new recognized schemas at runtime, but rather all used schemas would have to be known at the build time of the application. As XForms-based applications are often generic, instead of being specific to a particular form, it seems therefore best to adopt the use of the generic tables and automata.

Acquiring the schema to use in generation is another question. The algorithm that is used in generating the omission automata is mostly generic, but currently implemented only for RELAX NG (OASIS, 2001). For this initial assessment, the automata generation was also ported to understand XML Schema, to the extent required to process the sample documents[3]. This latter work should be useful, as XForms already supports XML Schema, and when no schema is specified, it should be possible to generate a schema from the template in the form. This generation would seem to be preferable to the alternative of implementing the automata generation algorithm specifically for the XForms language.

When using XML with schemas in the real world, it often occurs that the documents provided are not actually valid according to the schema. As noted, Xebu's omission automata provide some resilience against invalid documents (or, equivalently, schema changes), but the precise limits of this resilience are not known at present. Some cases are straightforward to observe, e.g., adding an element inside a sequence of elements is supported, and deleting a mandatory element is often not supported well.

It is not expected that the automata would be fully resilient to changes. However, what the serialization side automaton should be able to do is to determine in all cases whether the serialization succeeded or not. Here success is defined so that the parser side automaton will produce an equivalent XML document to the one that was transmitted. The current implementation can check that the automata have properly round-tripped back to the start state at document end, but this check might not be sufficient. Further evaluation with real documents that are schema-invalid would be needed.

[3]Note that this already includes a substantial amount of what is used in practice.

5 SAMPLE MEASUREMENTS

To evaluate the feasibility of using Xebu instead of XML we conducted a few simple measurements on a small set of real-world XForms data. According to the analysis of Section 4 this measurement only included the client-submitted data files and not the more complex server-provided forms. A total of three different files were used, called `CRM_basic`, `CRM_visit`, and `road`.

The main comparison point in these measurements should be against gzipped XML, as that has been deemed acceptable in the specific use cases. It is also assumed that schema information is present, so Xebu's schema-based techniques can be used. Comparisons should be made with both plain and gzipped Xebu, to determine whether Xebu's schema-based techniques are sufficient to eliminate the need for gzip completely.

The schemas for the files were hand-written based on the XForms data models for the applications. For further use, a system for automatically generating a schema from an XForms data model would need to be built, which does not appear infeasible, as the hand translation was sufficiently mechanical to automate. The provided example files have been verified to validate against the hand-written schemas.

The most interesting measurement will be the size of the resulting documents, as this is often the most significant concern in mobile messaging. Another interesting measurement will be the size of the compiled class files, as current mobile phones can have severe limitations on total application size.

The measurement application would first read in the XML file and convert it to an abstract representation. This representation is then serialized as Xebu into a file, the size of which is measured. Finally, the application parses the written file as Xebu into the abstract representation and verifies that it produced the same, or equivalent, result as parsing the original XML.

The sizes of the compiled class files are shown in Table 1. The files were obfuscated with Proguard[4] as would be done in real deployment. The table shows the size of the popular kXML[5] parser and serializer, the size of the Xebu parser and serializer, and finally the size of the generic code for using the text-based tokenization table and automata file, marked COA in the table. The size of an XForms implementation is measured in hundreds of kilobytes, so the overhead of including Xebu is not prohibitive on devices capable of supporting XForms.

[4] `http://proguard.sourceforge.net/`

[5] `http://kxml.org/`

Table 1: Compiled class file sizes in bytes for kXML and Xebu.

Code	kXML	Xebu	COA
Size	18056	22883	11990

Document sizes for the three example documents are shown in Figure 1. Sizes are shown for the original XML, gzipped XML, and for two kinds of Xebu: XebuT uses only the pre-tokenization tables and XebuS also uses the omission automata. Both Xebu kinds are given in plain and gzipped versions. To get these numbers, extraneous whitespace was removed from the XML files, as it is only used for indentation, and the current omission automata are not built to handle it.

The results of this experiment have one noteworthy point, namely that Xebu does much better on the `CRM` files than on the `road` file. The reason for this is that, while all of the files are approximately the same size, the `road` file consists mostly of repetitions of a certain element, which helps gzip in compressing the file. This reason is also evident when we note that gzip on top of Xebu does much better on the `road` file as well.

We also note that with schema optimizations even uncompressed Xebu is competitive with gzipped XML. This indicates that it might not be necessary to include compression support in the application to get the needed size reduction. Since a compression library is not included by default in current mobile phones, leaving it out could have a beneficial effect on application footprint, even though the footprint of Xebu is larger than that of kXML.

6 CONCLUSIONS

Based on the experiments above, we can conclude that replacing gzipped XML with gzipped bit-efficient XML yields significant benefits in the examined domains. In particular, even if the COA implementation is considered too fragile, the pre-tokenizations alone seem to provide improved compactness compared with gzip. We note that the Xebu implementation with the COA is approximately twice the size of kXML, so the code size poses little hindrance except in the most resource-constrained of environments. More complex bit-efficient XML formats, such as the upcoming EXI standard (W3C, 2007), may yield even more savings. On the other hand, in the resource-constrained mobile environment, simplicity of format and implementation are also important.

Specifically in the XForms case, we recommend

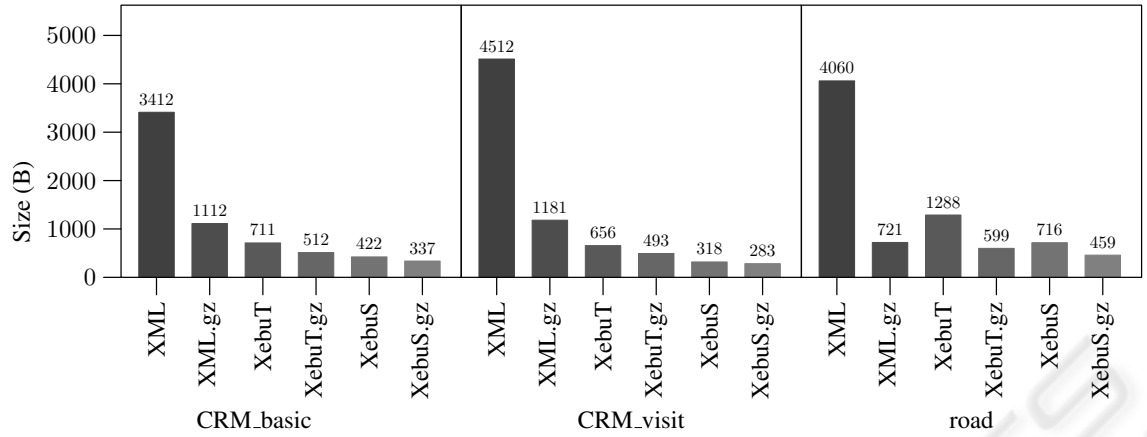

Figure 1: Document sizes in bytes for gzipped XML and Xebu.

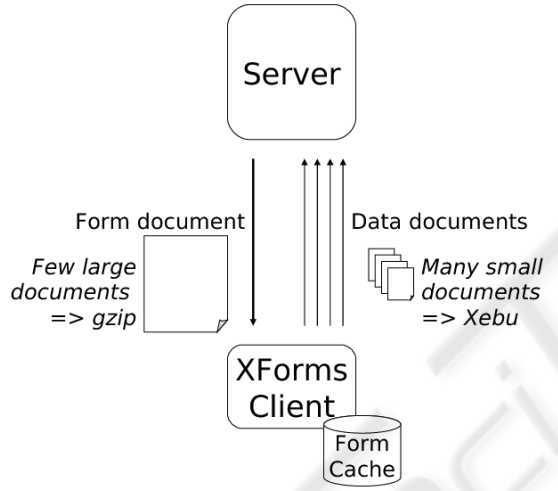

Figure 2: Recommended compression use in XForms applications.

the approach shown in Figure 2. Namely, gzip should be used for the actual form documents, since they are sufficiently large for gzip to do well and the XForms schema is complex so Xebu's schema optimizations have less effect. On the other hand, the situation is reversed for the actual data documents, which are small and have rigid schemas, so Xebu, or another equally efficient binary format, should be used there.

7 FUTURE WORK

The design and implementation of Xebu has been performed in the context of a research project, and its main purpose has been to test a variety of ideas related to binary XML. Therefore the implementation choices that have been made may not be the best possible for wide application, and we have some concerns that would be useful to address to make Xebu applicable more widely.

The implementation of typed content fits well into Xebu's initial application area, and to take proper advantage of type support in the format does require an extended API of some form. However, requiring `xs:type` attributes is not a very good solution, as they are not often present in real data apart from some SOAP applications, and they require additional processing. A better solution could be to build a filter based on the schema that would decode bytes from the input directly into objects based on the schema state.

The omission automata are currently specified as normal finite automata. This means that they, e.g., cannot support recursive content in elements. Another problem caused by this is that, for each kind of element, the part of the automata constructed for that element needs to be duplicated everywhere that the element could appear. This does not cause any problems for so-called "Russian doll" schemas (van der Vlist, 2001), but with a flat schema this could lead to a blowup in the size of the automaton compared to the schema (theoretically even exponential, but this should not be a concern in the XForms case). This could be fixed by splitting each element into its own automaton and extending the automata to use an auxiliary stack to keep track of the automata corresponding to the parent elements and the states they are in.

To consider the transmission format for the automata and tokenization tables, the current text-based format is designed for readability to aid in debugging the automata. In particular, there is much repetition and readable names. It would be straightforward to design a much more compact format by tokenizing all information and packing them tightly into bytes. However, it is not yet clear what kinds of size savings

such a format would have over gzipping the current text-based format. Of the files used here, the sizes of such gzipped files were between 2 and 4 kilobytes.

However, all of these considerations on extending Xebu need to be contrasted with how work at the W3C on a standard binary XML format is progressing (W3C, 2007). Assuming that this process manages to produce something stable soon, it might be better to invest in implementing the standard on mobile phones instead of developing what is essentially a proprietary format. Xebu does have the advantages, though, that an implementation already exists, its properties are mostly well-understood, and the Xebu format is much simpler than the current EXI draft format.

REFERENCES

Deutsch, L. P. (1996). *RFC 1952: GZIP File Format Specification Version 4.3*. Internet Engineering Task Force.

Fielding, R., Gettys, J., Mogul, J., Nielsen, H. F., Masinter, L., Leach, P., and Berners-Lee, T. (1999). *RFC 2616: Hypertext Transfer Protocol — HTTP/1.1*. Internet Engineering Task Force.

ITU (2004). *Mapping W3C XML Schema Definitions into ASN.1*. International Telecommunication Union, Telecommunication Standardization Sector, Geneva, Switzerland. ITU-T Rec. X.694.

Kangasharju, J., Tarkoma, S., and Lindholm, T. (2005). Xebu: A binary format with schema-based optimizations for XML data. In Ngu, A. H. H., Kitsuregawa, M., Neuhold, E., Chung, J.-Y., and Sheng, Q. Z., editors, *The 6th International Conference on Web Information Systems Engineering*, volume 3806 of *Lecture Notes in Computer Science*, pages 528–535, Heidelberg, Germany. Springer-Verlag.

Karhinen, A., Koskimies, O., and Nurminen, J. K. (2007). Experiences in applying a mobile service platform across different business domains. In *The 4th International Workshop on Ubiquitous Computing*, pages 33–42.

Niedermeier, U., Heuer, J., Hutter, A., Stechele, W., and Kaup, A. (2002). An MPEG-7 tool for compression and streaming of XML data. In *IEEE International Conference on Multimedia and Expo*, pages 521–524.

OASIS (2001). *RELAX NG Specification*. Organization for the Advancement of Structured Information Standards, Billerica, Massachusetts, USA.

Pericas-Geertsen, S. (2003). Binary interchange of XML Infosets. In *XML Conference and Exposition*.

Sandoz, P., Triglia, A., and Pericas-Geertsen, S. (2004). Fast Infoset. On Sun Developer Network.

Sosnoski, D. M. (2003). XBIS XML Infoset encoding. In (W3C, 2003).

Traudt, E. and Konary, A. (2005). 2005 software as a service taxonomy and research guide. Research report, IDC.

van der Vlist, E. (2001). Using W3C XML schema. On XML.com.

W3C (2001a). *XML Schema Part 1: Structures*. World Wide Web Consortium, Cambridge, Massachusetts, USA. W3C Recommendation.

W3C (2001b). *XML Schema Part 2: Datatypes*. World Wide Web Consortium, Cambridge, Massachusetts, USA. W3C Recommendation.

W3C (2003). *W3C Workshop on Binary Interchange of XML Information Item Sets*. World Wide Web Consortium.

W3C (2005). *XML Binary Characterization*. World Wide Web Consortium, Cambridge, Massachusetts, USA. W3C Note.

W3C (2006). *XForms 1.0*. World Wide Web Consortium, Cambridge, Massachusetts, USA, 2nd edition. W3C Recommendation.

W3C (2007). *Efficient XML Interchange (EXI) Format 1.0*. World Wide Web Consortium, Cambridge, Massachusetts, USA. W3C Working Draft.

Werner, C., Buschmann, C., Brandt, Y., and Fischer, S. (2006). Compressing SOAP messages by using pushdown automata. In *IEEE International Conference on Web Services*, pages 19–26, Piscataway, New Jersey, USA. Institute of Electrical and Electronic Engineers.