Implementation and Evaluation of a Binary Interchange System for XML-Applications in a Cellar Phone

Kazunori Matsumoto Arei Kobayashi Naomi Inoue
{matsu,kobayasi,inoue}@kddilabs.jp

KDDI R&D Laboratories
2-1-15 Ohara Kamifukuoka-shi Saitama 356-8502 Japan

Table of Contents

Introduction

- Promising XML-applications and technical obstacles -

Applications in a cellar phone are prevailing owing to middleware.

- Java VM
- BREW (Binary Runtime Environment for Wireless)

Following XML-applications in a cellar phone are promising.

- SVG map
  Merits: zoom-in , zoom-out, rotation, standard format
- PIM data synchronization
  Merits: interoperation with PCs, standard format
- etc.

XML-base applications in cellar phones are promising because of interoperable format, if the following two technical obstacles are avoided.

- Limited bandwidth for transmission
  Raw XML documents are inefficient to transmit.
- Limited CPU power
  Raw XML document requires much computation cost for parsing.

Introduction

- Problems of conventional compression methods -

1. Generic compression method (e.g. gzip)
   Compressed contents are uncompressed by an application in a handset. Thus, the computation cost of uncompressing and parsing is needed. It is shown later that the parsing cost is fatal to applications.

2. XML-aware compression methods (e.g. XMill, xmlppm)
   XMill [1] divides a document into three tables (1) element/attribute names, (2) strings such as element/attribute value, (3) tree structure, and compress each table with a generic compressor. xmlppm [2] compress tree structures with a special technique called as “Prediction by Partial Match”. These methods may reduce data size better than “gzip”. But fatal parsing cost is also needed.

3. Namespace-dependent compression method (e.g. WBXML: WAP Binary XML)
   WBXML [3] is specified by WAP (Wireless Application Forum [4]), and is a pre-parsed binary format for a specific application. WBXML may reduces the parsing cost, but the encoding and decoding program is hard to re-use because the format is designed for the specific application.

Proposal for a binary interchange system for cellar phone applications

- The requirements (design goal) are as follows:
  - To deal with a non-specific XML documents (not restricted to a specific schema)
  - Pre-parsed stream to reduce decoder's computation cost

The proposal is ``schema-aware compression''.

- Encode/decode process is based on schema information.
  - Schema information includes:
    - Syntax (vocabulary, tree structure, data type)
    - Code dictionary (binary representation of name)
- Schema information is known to both a sender and a receiver.

The general XML document encoding/decoding method "XEUS" is a complete example of the proposal. I will explain XEUS in the following sections.

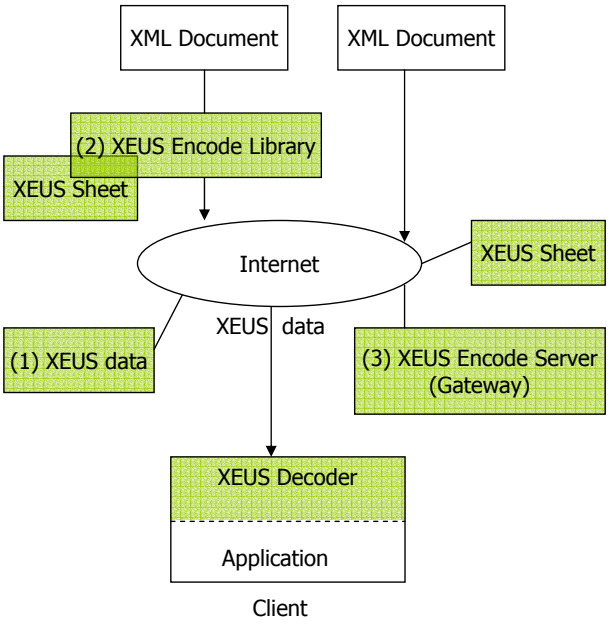
Overview of XEUS

Xml document Encoding with Universal Sheet

- XEUS is a coding system for a general (non-specific) XML document.
- It encodes/decodes with the coding table called "XEUS Sheet", which depends on the namespace of a target document.

If the optimal coding table is prepared for every namespace of an XML document (XHTML, SVG, etc.), the rate of compression can be improved.

Contents delivery models for XEUS



There are three contents delivery models as follows:

(1) XML documents are encoded in advance. Encoded binaries are stored in a server.

(2) XEUS encoder library is embedded in the server which generates XML documents dynamically. The server generates encoded data dynamically.

(3) XML document server has no encoding function. XML documents are encoded in "XEUS Encode Server", which works as a gateway.

Overview of XEUS Sheet

XEUS sheet defines schema information and strategies of compression (byte aligned or not, what compression method is applied after the first encoding stage)

XEUS sheet sample:

```
<xeus version="2.0" xmlns="#xeus-sample" >
 <head>
  <root name="sample_elem0" bit="8"/>
code="00000000"/>
 </head>
 <body alignment-level="all" compress="none" >
  <element name="sample_elem0">
   <attlist>
    <attr name="elem0_attr0" type="implied">
     <value>
      <char encoding="Shift_JIS" length="implied"/>
     </value>
    </attr>
   </attlist>
   <children bit="8">
    <child_element name="sample_elem1"
code="00000001"
     type="required"/>
   </children>
```

data type

schema

```
<element name="sample_elem1">
  <attlist>
   <attr name="elem1_attr0" type="implied">
    <value>
     <number bit="8" data="UI" qt="1"/>
    </value>
   </attr>
   <attr name="elem1_attr1" type="required">
    <value>
     <choice bit="8" qt="1">
<list code="00000000">sample_value0</list>
<list code="00000001">sample_value1</list>
     </choice>
    </value>
   </attr>
  </attlist>
 </element>
 </body>
</xeus>
```

code dictionary

Overview of Coding Model

XEUS encoder serializes a XML document to the following six parts recursively:

1. **Element start code**: defines start of an element.
2. **Length**: specifies the code-length of a target element.
3. **Element symbol**: specifies the code of a target element. The code value is defined in a XEUS sheet.
4. **Attribute existence code**: specifies the existence of attributes in a target element. Existence of each attribute is assigned to 1 bit.
5. **Attribute value**: specifies the value of an attribute. The data representation is defined in a XEUS sheet. Multiple values in a single attribute are accepted.
6. **Element value**: specifies the value of an element. The data representation is defined in a XEUS sheet. Multiple values are accepted.

Decoder application can re-construct a tree structure from 1,2,3,6.

Implementation of XEUS system

- **XEUS Encoder Server** [5]
  - This server receives a user's http request and relays it to a XML-document server. Downloaded document from the document server is encoded with the XEUS sheet specified by user's request.
- **XEUS Decoder on Java** [5]
  - The decoder is described in Java (J2ME/CLDC), which is a middleware of many cellar phone.
  - Decoder's API is an extension of XMLPull API. Application can receive values of elements/attributes directly as raw data type, not as a string. (This reduces data binding efforts of an application.)
- **XEUS Decoder on BREW** [6]
  - The Namespace-dependent decoder program described in C language is generated from a given XEUS sheet. The program is executed on BREW phone.
  - The decoder's API can be decided in a program generation stage. Either SAX-like API or DOM-like API is available. Data binding of elements/attributes values is also supported.

Evaluation of XEUS Encoder
- Experiment Environment -

Configuration of XEUS Encode Server

CPU	Pentium III 1GHz
Memory	640MB
Network	Ethernet (10Mbps)
OS	RedHat Linux 7.2
httpd	apache2.0
XML parser	Xerces C++ 2.2.0 [7]
Target Data	SVG
	XMark [8]

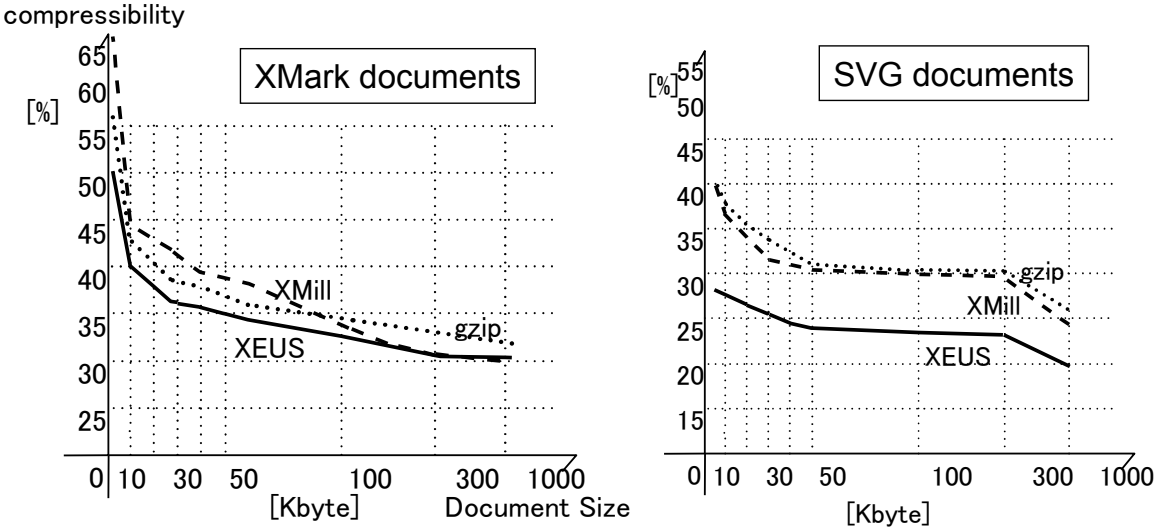
The rate of markup tag is higher in SVG documents than in XMark documents. XMark documents have many values whose data type is a string.

Compression strategy describe in XEUS sheet is as follows:

- Byte alignment
- Second level compression is “gzip”

Size of a document is 10K to 100K bytes.

Evaluation of XEUS Encoder
- Comparison of compressibility -



Compressibility = Size of encoded data / Size of original document

When the markup rate of a document, such as a XMark documet, is low, compressibility of XEUS is almost same as that of other conventional methods. In case of SVG documents which have high markup rate, XEUS is obviously better than other conventional compression methods.

Evaluation of XEUS Encoder
- Encoding Time-

Document size and encoding time

Document Size [Byte] (**SVG**)	Encoding Time [msec]	Document Size [Byte] (**XMark**)	Encoding Time [msec]
10,385	37.705	5,832	18.938
20,550	72.276	8,947	25.266
30,664	100.873	27,233	67.243
40,805	134.787	39,021	93.267
50,479	167.476	47,811	119.317
102,111	323.294	118,274	276.644

When the first occurrence of new namespace, this XEUS encoder has to parse a XEUS sheet. Time to parse the XEUS sheet of SVG is 78 milliseconds, and time to parse that of XMark is 103 milliseconds. Please note that the process parsing a XEUS sheet is needed at the first occurrence.

From this table, the encoding time is almost in proportion to the document size.

XEUS encoder can encode a 100KB document in about 300 milliseconds. This result shows the encoding time of this encoder is enough feasible.

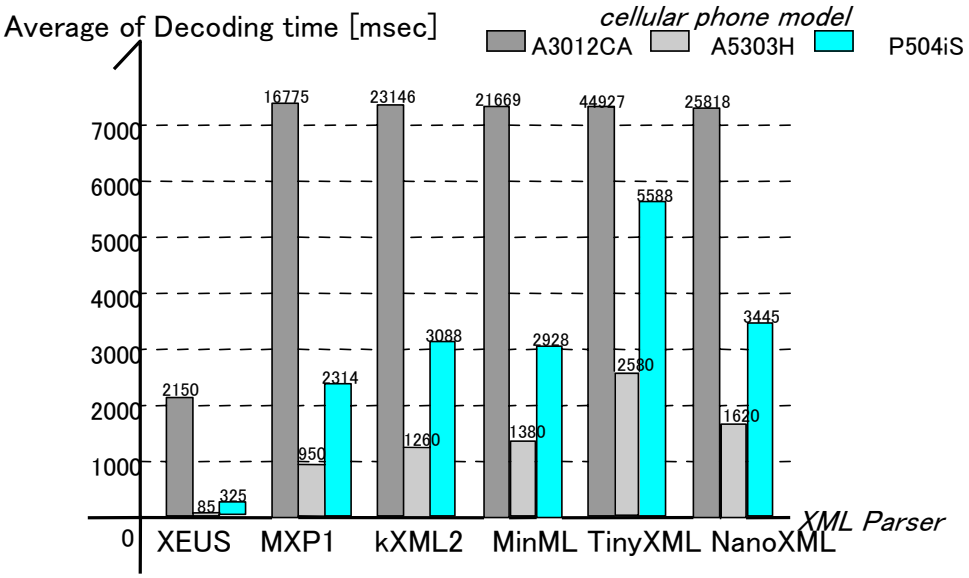
Evaluation of XEUS Decoder on Java
- Experiment Environment -

Conventional SAX-like parsers to be measured	MXP1 [8] kXML2[9] MinML [10] NanoXML/Lite [11] TinyXML [12]
Test document	SVG (11,502 byte) XEUS sheet (3,300 byte)
Cellar phones Model Name	A3012CA (KDDI) A5303H (KDDI) P504is (DoComo)
Performance Test specification	To count the occurrence of element/attribute values.

Known XML-compressor such as XMill can not be executed on a cellar phone because the object size of their program is too large for a handset resource. Only SAX-like parsers can be executed on it. Therefore, KDDI measured the time such a light-weight parsers parses a raw text, and compared XEUS's decoding time with them.

Evaluation of XEUS Decoder on Java

- Decoding Time -



XEUS decoder is about 9 times faster than MXP1 which is supposed to be the fastest in SAX-like XML parsers.

Evaluation of XEUS Decoder on Java

- Object Size (jar size) -

Comparison of object size (jar size)

XEUS	MXP1	kXML2	MinML	NanoXML /Lite	TinyXML
8.9KB	17.8KB	11.3KB	13.6KB	6.8KB	9.8KB

Object size of XEUS decoder is almost even in these light-weight parses. From the point of object size, XEUS decoder written in Java is a feasible solution for cellar phones.

Evaluation of XEUS Decoder on BREW

- Experiment Environment -

Measured item	•gunzip + SAX2 •Generated XEUS decoder (SAX API) •Generated XEUS decoder (DOM API) •Man-made XEUS decoder (SAX API)
Test document	SVG (11,469 to 115986 byte) XEUS sheet (3,300 byte)
Cellar phones Model Name	A5304T (KDDI)
Performance Test specification	To count the occurrence of element/attribute values.

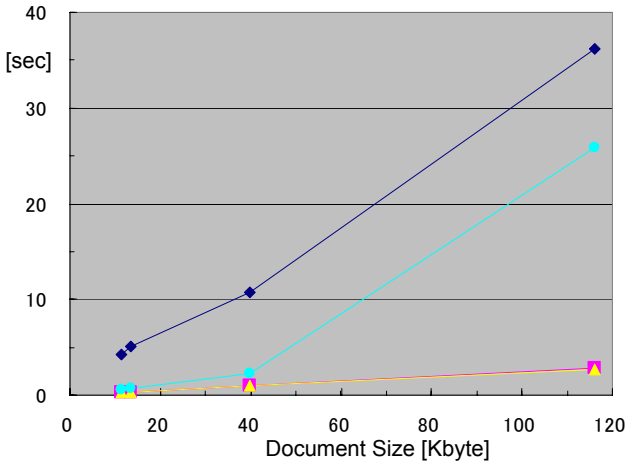
•Processing time of gunzip and SAX2 are measured respectively.
•Commercial SAX2 parser for BREW is provided by Reaxion [13].
•Man-Made XEUS decoder was developed before the XEUS decoder generator is available. Man-Made XEUS decoder works for a small subset of SVG namespace.

Evaluation of XEUS Decoder on BREW

- Decoding Time -

Decoding Time [sec]

Doc. Size	gunzip+ SAX2 (gunzip)	XEUS (SAX)	XEUS (DOM)	Man-Made XEUS
11,469	4.171 (0.015)	0.270	0.510	0.295
13,556	5.060 (0.070)	0.311	0.755	0.332
39,709	10.683 (0.215)	0.938	2.320	0.934
115,986	36.126 (0.582)	2.760	25.890	2.754



• Generated XEUS decoder (SAX) is about 14 times faster than gunzip+SAX2.
• Generated XEUS decoder (SAX) is even to Man-made decoder.
• Although gunzip is fast, parsing time for a raw text is very slow.

Evaluation of XEUS Decoder on BREW

- Object Size -

Comparison of object size (byte)

SAX2 [13]	XEUS (SAX)	XEUS (DOM)	Man-made XEUS (SAX)
22,012	24,596	26,644	50,616

As the memory space for BREW applications is about 150KB in KDDI's 5304T handset, object size of XEUS decoder is feasible enough. Note that the object size of generated XEUS decoder is much smaller than that of man-made XEUS decoder, although man-made decoder has some limitations in the decode process.

Summary and Consideration

- Schema-aware compression is proposed. Schema-aware means namespace-dependent. XEUS is a complete example of such a compression method. KDDI implemented XEUS and evaluated it.
  - Encoder: better compressibility than conventional methods, feasible encoding time
  - Decoder: feasible decoding time and object size are measured on commercial cellar phones with different middleware (Java and BREW)
- Design goal is to reduce decoder's computation cost. This implies the reduction of transmission bandwidth.
- As applications in a cellar phone are assumed, document size is 10KB to 100KB.
- XEUS sheet is described in "*yet another*" schema language. But the authors suppose current well-known schema language is NOT enough to describe a code dictionary for transmission.
- Encoded binary by XEUS is not supposed to be used for random access in a document, dynamic update and streaming. The optimized binary format for such applications is much different to current XEUS.

Bibliography

[1] H.Liefke and D.Suciu. Xmill: an efficient compressor for XML data, In Proceedings of the 2000 ACM SIGMOD, pp.153-164,2000

[2] James Cheney, Compressing XML with Multiplexed Hierarchical Models, in Proc. of the 2001 IEEE Data Compression Conference, pp. 163-172

[3] Bruce Martin and Bashar Jano: http://www.w3.org/TR/wbxml/

[4] Wireless Application Protocol Forum, Binary XML Content Format Specification Ver. 1.3, 2001

[5] Arei Kobayashi, kazunori Matsumoto and Naomi Inoue: Performance evaluation of XML document Encoding with Universal Sheet, in Proc. of 2003 Forum on Information Technology (FIT) (will be appered, in Japanese)

[6] Kazunori Matsumoto, Arei Kobayashi and Naomi Inoue: Generation of XML-Aware Decoder for Mobile Phones, in Proc. of 2003 IPSJ Spring Conference, (in Japanese)

[7] http://xml.apache.org/xerces-c/index.html

[8] A. Schmidt, F. Wass, M. Kersten, D. Florescu, M. Carey, I. Manolescu, R. Busse: Why And How To Benchmark XML Database, SIGMOD record 30(3): 27-32(2001)

[8] http://www.extreme.indiana.edu/xgws/xsoap/xpp/mxp1/

[9] http://www.kxml.org/

[10] http://www.wilson.co.uk/xml/minml.htm

[11] http://nanoxml.n3.net

[12] http://www.grinninglizard.com/tinyxml/

[13] http://www.reaxion.com/