Compact Binary XML

Michael Conner
IBM
mconner@us.ibm.com
9/24/03

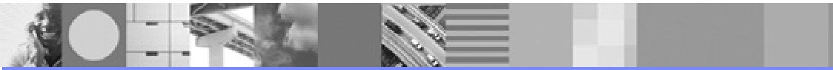
Why Binary XML – Document Size

- There is a great deal of structural overhead for typical business data
  - <myns:shoeSize>9</myns:shoeSize>

  -
    ```
    <myns:path>
      <myns:position><myns:x>23</myns:x><myns:y>14</myns:y></myns:position>
      <myns:position><myns:x>24</myns:x><myns:y>16</myns:y></myns:position>
      <myns:position><myns:x>25</myns:x><myns:y>18</myns:y></myns:position>
    </myns:path>
    ```
- This overhead can be critical in bandwidth limited environments (e.g., client support, both traditional and pervasive) or when messages are large.
- Possible fixes:
  - Short, non-mnemonic tags: <x:z>9</x:z>
  - Custom value representations: <myns:path>(23,14)(24,16)(25,18)</myns:path>
  - Zip-style compression

Why Binary XML – Processing Time

- Lexical rules are somewhat complex
- There is a great deal of string processing
- Validation, signing and encryption can all be very expensive
- This overhead can be critical in high volume scenarios, low-footprint scenarios, and processing constrained scenarios.
- Possible fixes:
    - Striped down parsers
    - Don’t validate
    - Hardware assist
    - Schema compilers

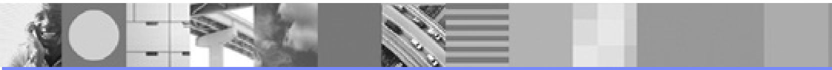


Binary encoding levels

1. **Stream-structured binary** – only XML structure is in binary, values are still characters
   - Preserves all XML values except direct human readability
2. **Tree-structured binary** – only XML structure is in binary, values are still characters, tree structure is captured (parent, child, siblings)
   - Cannot be streamed, but may be more efficient for partial message processing
   - Additional structure information increases message size
3. **Full binary** – both structure and values are in binary
   - May be somewhat faster to process, probably neutral on message size
   - Introduces values representation and portability issues
4. **Schema dependent binary** – Knowledge of the document’s schema is used to remove tags and compress values
   - Can be very compact
   - Message becomes very fragile
   - Very complex encoding rules

CBXML Goals: Reduce XML costs without compromising XML value

- Benefits
  - Remove much of the space overhead from XML structure
  - Streamline the lexical phase of XML to reduce parsing footprint and processing time
- Values
  - Preserve the portability of XML messages (no byte order issues, no platform bias)
  - Preserve the self-descriptive nature of XML, keep all tags
  - Accurate rendering of the full XML infoset including namespaces (could extend to full XML 1.1 spec) – Can easily define an exact conversion to/from character XML for a simple canonical form
  - Simple standalone tool to convert to/from CBXML to support human viewing and editing
  - Conversion to/from CBXML does not require any context (External dictionary, Schema or DTD)
  - Support streamed processing (pull parsing, partial parsing)
- Issues
  - It’s a new encoding – raises interoperability and support issues
  - It’s not directly human viewable or editable with simple text editors



CBXML Design

- Sequence of typed clauses
- Two primitive types: variable length integer, encoded string
  - Variable length integers: each byte holds 7 bits of data and a flag bit
  - Encoded strings are run-length delimited byte sequences
- Objects (qualified names, namespace maps, content strings, …) are encoded inline on first occurrence and by reference after the first occurrence. However, strings can be duplicated to avoid a requirement for fully sequential processing or to reduce generation costs. (E.g., start over for SOAP body.)
- Document is just a serialization of the essential content of the XML infoset as a series of typed clauses based on the primitives given above

CBXML Encoding Example

```
[  346] startElementAttributes------------- 3
[  347] old: unqualified element name------ 3 = "():person"
[  348] number of attributes--------------- 2
[  349] old: unqualified attribute name---- 2 = "():id"
[  350] new: attribute value--------------- 1
[  351] ----------------------------------- 10,"one.worker"
[  362] old: unqualified attribute name---- 3 = "():contr"
[  363] old: attribute value--------------- 3 = "false"
===========================================
[  364] startElementNamespaceDecls--------- 4
[  365] number of NS Maps------------------ 1
[  366] old: namespace map----------------- 3 = "xmlns=http://foo.com/hr"
[  367] old: qualified element name-------- 4 = "(http://foo.com/hr):name"
[  368] number of attributes--------------- 0
===========================================
[  369] startElement----------------------- 2
[  370] old: qualified element name-------- 5 = "(http://foo.com/hr):family"
===========================================
[  371] characters------------------------- 1
[  372] new: content----------------------- 1
[  373] ----------------------------------- 6,"Worker"
===========================================
[  380] endElement------------------------- 5
===========================================
[  381] endElement------------------------- 5
===========================================
[  382] endElement------------------------- 5
```

```
…
<person id="one.worker" contr="false">
  <name xmlns="http://foo.com/hr">
    <family>Worker</family>
  </name>
</person>
…
```

113 characters => 36 bytes

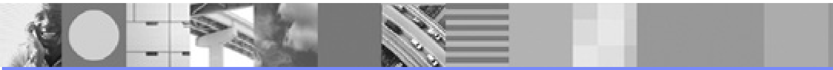


Table example

```
<table:table>
…
<table:row>
…
<table:column>209</table:column>
<table:column>80</table:column>
…
</table:row>
…
</table:table>
```

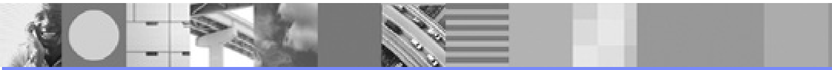
```
XML = 63 bytes

Current CBXML:
2,4,1,1,3,2,0,9,5,2,4,1,24,5 = 14 bytes

CBXML with leaf node support:
12,4,1,3,2,0,9,12,4,24 = 10 bytes

Old style binary: 8 bytes
```

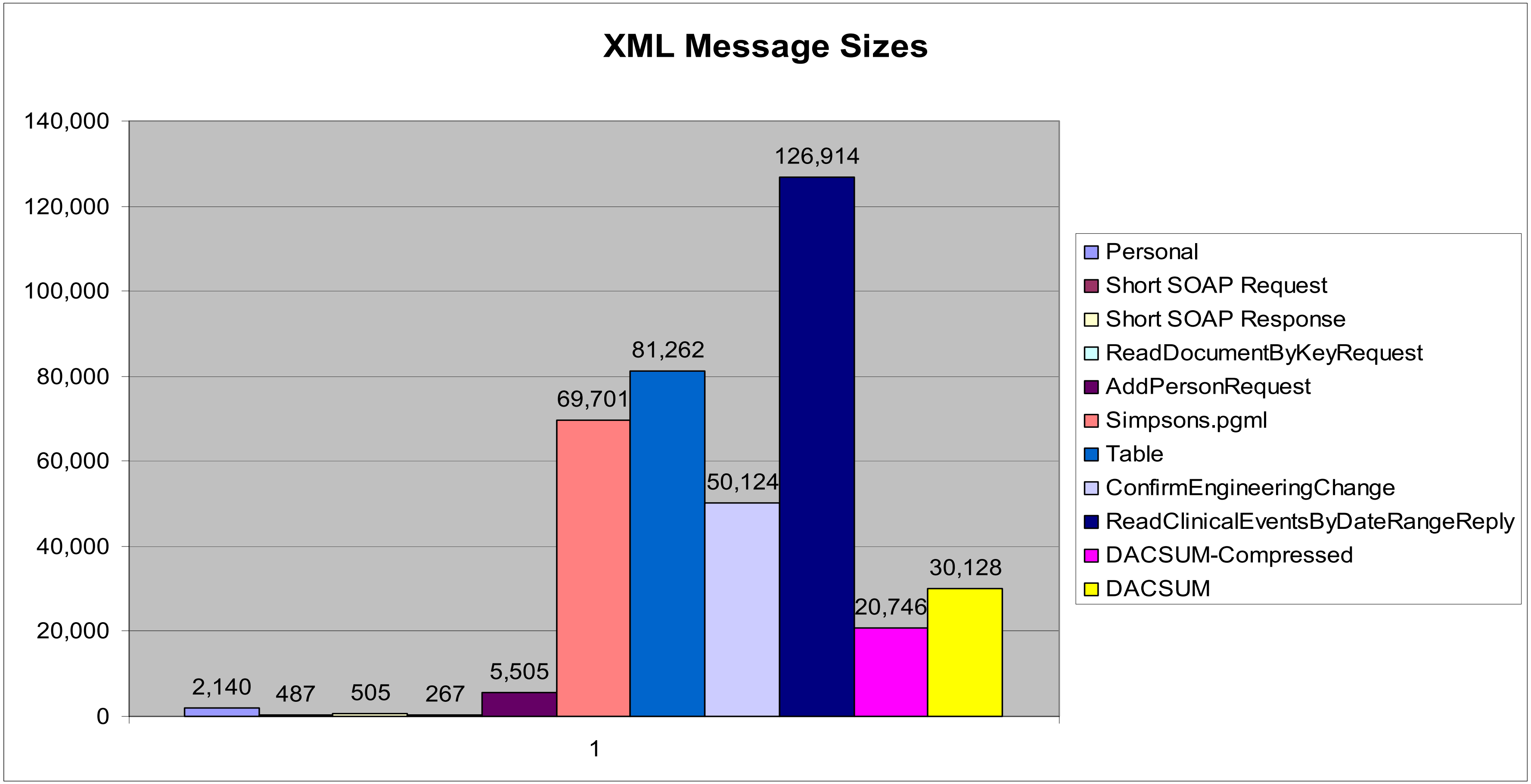
```
[   280] startElement----------------------- 2
[   281] old: qualified element name-------- 4 = "(http://com.ibm.cbxml.testns)table:column"
=============================================
[   282] characters------------------------- 1
[   283] new: content----------------------- 1
[   284] ----------------------------------- 3,"209"
=============================================
[   288] endElement------------------------- 5
=============================================
[   289] startElement----------------------- 2
[   290] old: qualified element name-------- 4 = "(http://com.ibm.cbxml.testns)table:column"
=============================================
[   291] characters------------------------- 1
[   292] old: content----------------------- 24 = "80"
=============================================
[   293] endElement------------------------- 5
```



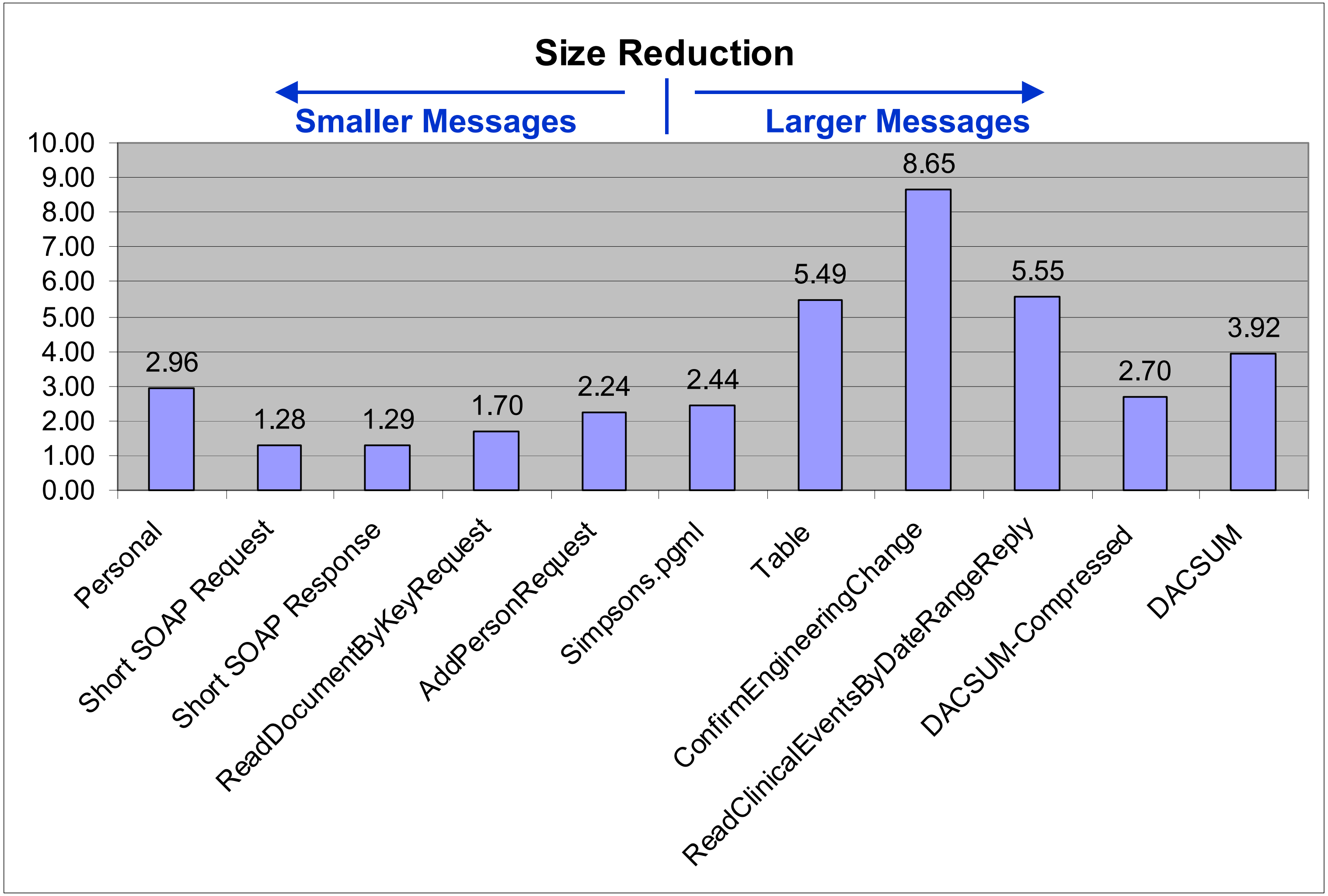
Exact mapping canonical rules

- Start with character representation, canonicalize as follow:
  - Single space between parts of an start element.
  - Double quotes for attribute values (could easily remove this restriction by indicating the quote type on attribute values)
  - No embedded DTD subset
  - Namespace declarations before regular attributes (could relax this if needed)
  - Use <start n=v/> style for elements with no content and no nested elements
- Conversion must follow these rules
  - Preserve order of attributes
  - Preserve content whitespace

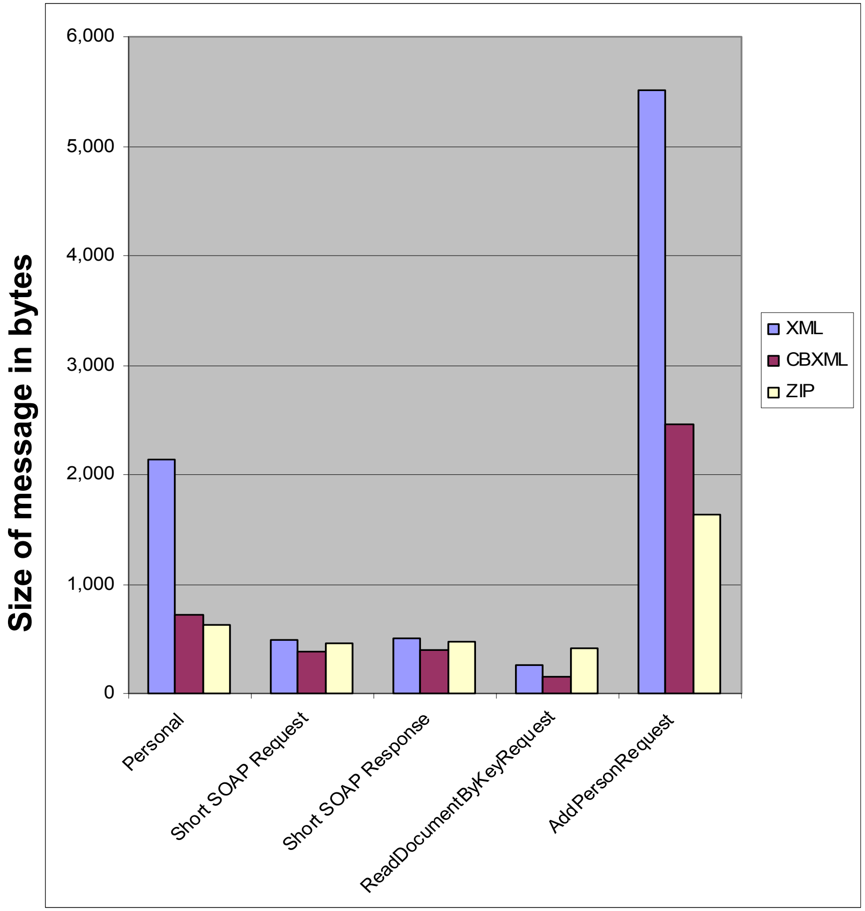
Trial Messages

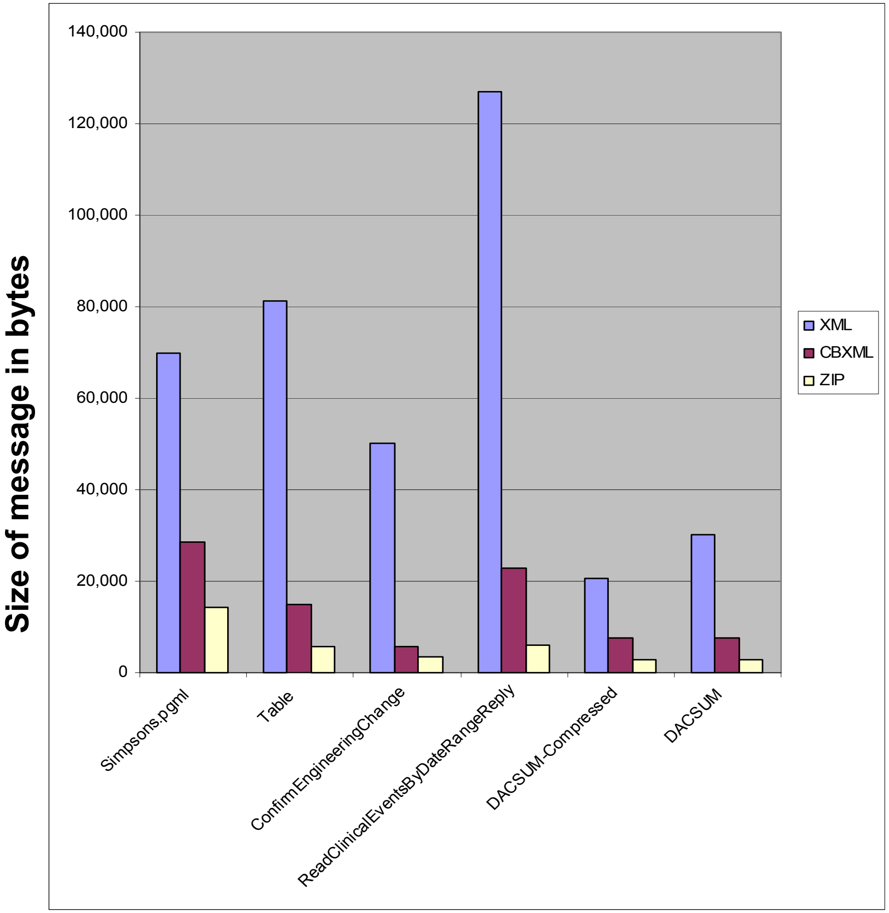


CBXML: Space Advantage

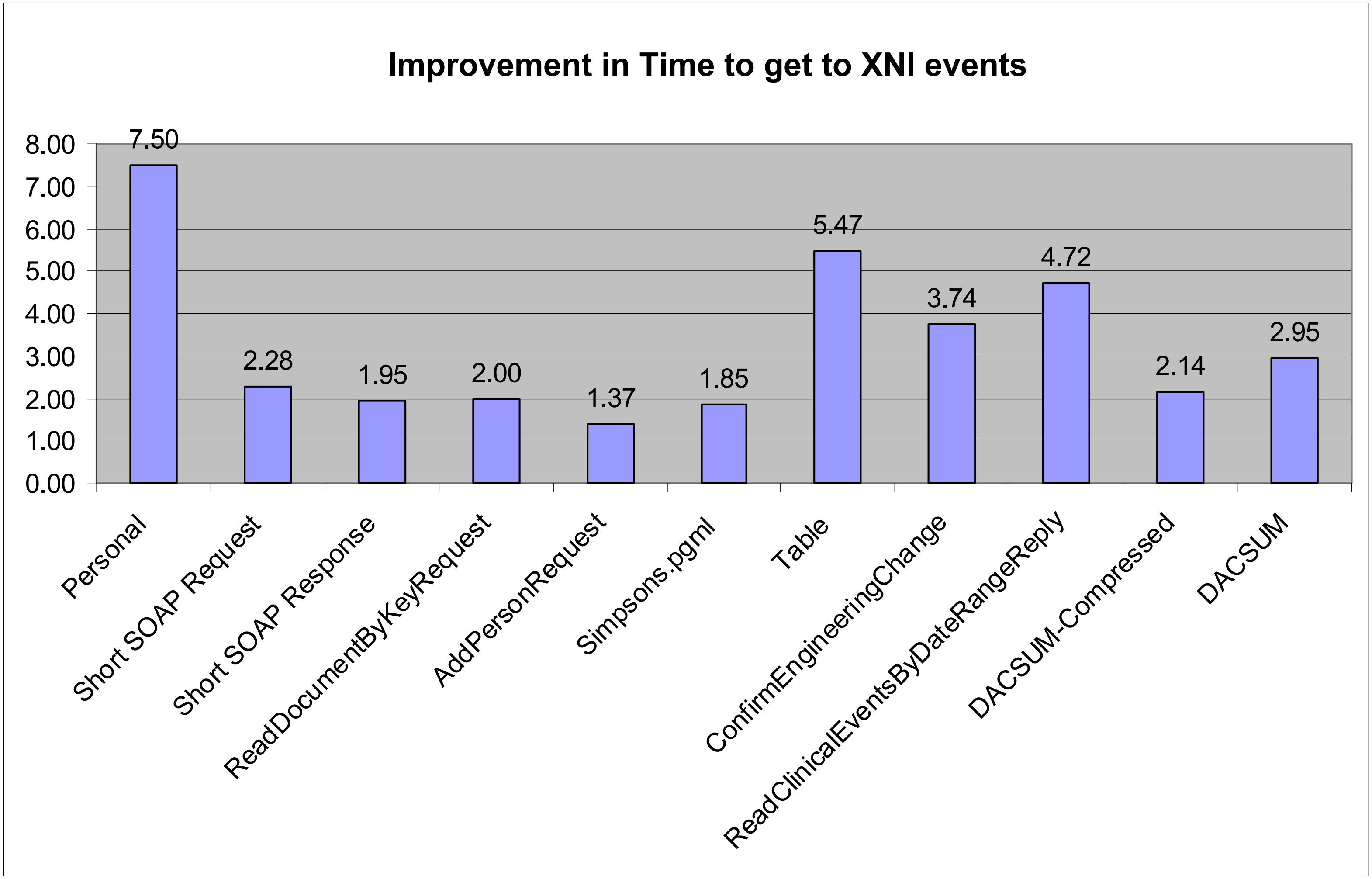


Comparison to ZIP



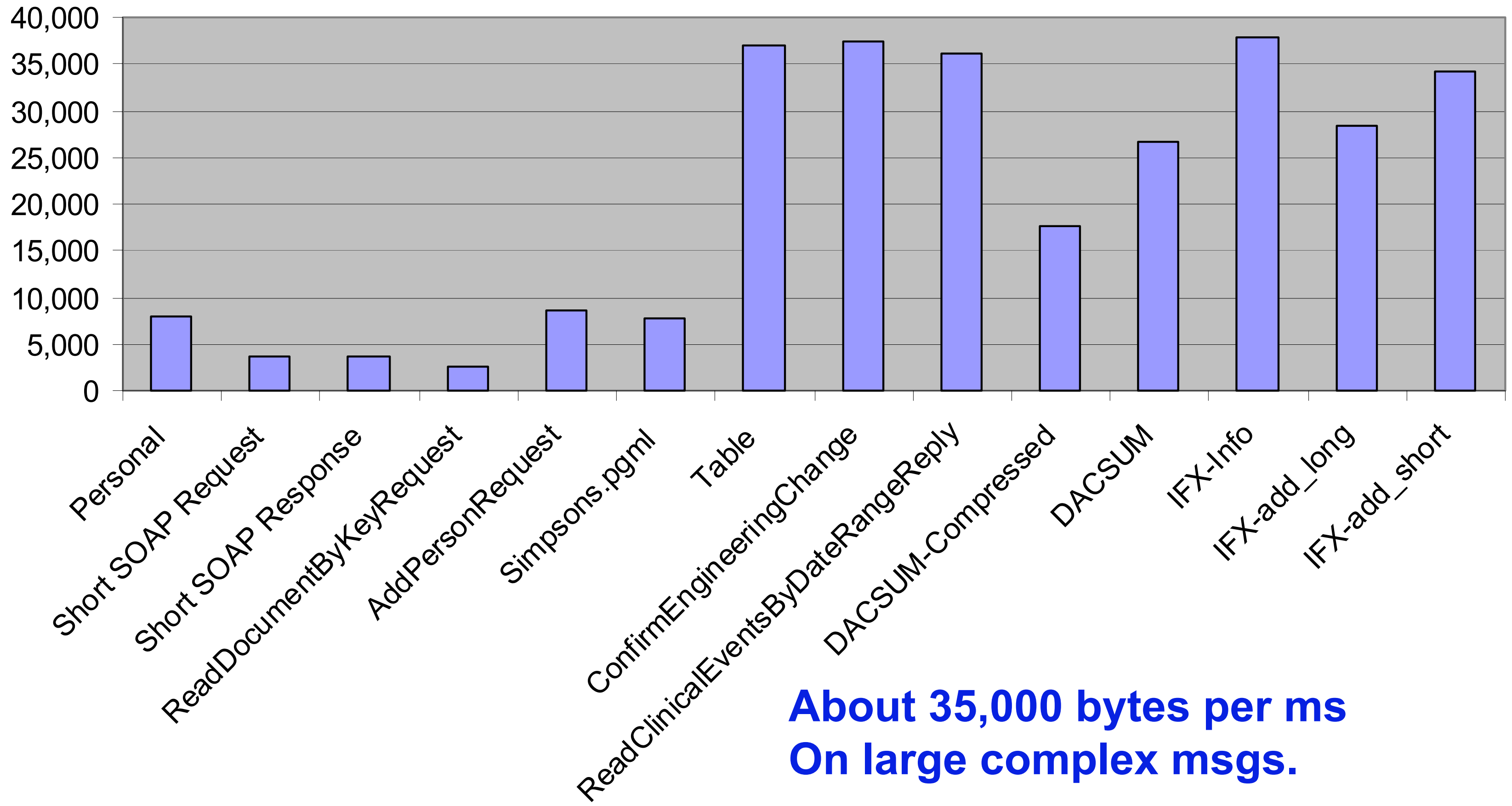


CBXML: Time Advantage



Throughput on .8 Ghz Thinkpad of C

Throughput in bytes per ms for CBXML messages



About 35,000 bytes per ms
On large complex msgs.

Conclusions

- Substantial space and processing time savings can be achieved while preserving almost all of the XML values.
- CBXML examines an approach to binary encoding. The approach can be taken further without compromise to its values or efficiency: idioms, full XML 1.1 representation, relaxed canonicalization rules
- Binary encoding of values (other than mime data) probably is not worth the impact
- ZIP-style compression offers the best size reduction, but can complement binary encoding for increased effect. Comes at a processing cost.

- Speculation: Moving to schema dependent encodings, in any form, would have minor value over stream-structured binary for complex messages, and is not optimal for small messages. (The dictionary is the issue for very small messages.)