The binary encoding in OpenMath 2

Michael Kohlhase

School of Engineering & Science

International University Bremen, Germany

http://www.faculty.iu-bremen.de/mkohlhase

Why oh Why do we need a Binary Encoding for OM

- OPENMATH setup: abstract OM objects, XML encoding, binary encoding (The core of OPENMATH is the OM object model, not the XML encoding)
- Content Objects can get rather large, (think finite group representations, Coq proof terms)

  and XML is verbose for generality (a factor of 5++ can help)
- parse-time is proportional to file size (so gzip does not help)
- in some applications (e.g. MBASE), OM Objects are better stored as binary blobs
- if we do not standardize it, ad-hoc solutions will bloom (interoperability will suffer)

Binary Encoding How-To

- Idea: Use single bytes token for start (and stop) tags
  - bit 1-5 for the token identifier (what kind of element)
  - bit 6 is a status bit, currrently only used for streaming flag" (new in OM2, see below)
  - bit 7 is the "sharing bit" (new in OM2, facilitates sharing)
  - bit 8 is the "long flag" (4-byte string/sequence lengths)

Parts of the Grammar

start	→	[24] object [25]	|	[24+64] object [25+64]
integer	→	[1] [_]	|	[1+64] [n] id:n [_]
	|	[1+32] [_]		
	|	[1+128] {_}	|	[1+64+128] {n} id:n {_}
	|	[1+32+128] {_}		
	|	[2] [n] [_] digits:n	|	[2+64] [n] [m] [_] digits:n id:m
	|	[2+32] [n] [_] digits:n		
	|	[2+128] {n} [_] digits:n	|	[2+64+128] {n} {n} [_] digits:n id:n
	|	[2+32+128] {n} [_] digits:n		
symbol	→	[8] [n] [m] cd:n sym:m	|	[8+64] [n] [m] [k] cd:n sym:m id:k
	|	[8+128] {n} {m} cd:n sym:m	|	[8+64+128] {n} {m} {k} cd:n sym:m id:k
application	→	[16] object objects [17]	$bigl\|$	[16+64] {m} id:m object objects [17]

A Binary Encoding Example

Hex	Meaning
58	begin object tag
2	version 2.0 (major)
0	version 2.0 (minor)
10	begin application tag
08	symbol tag
06	cd length
05	name length
61	a (cd name begin
72	r .
69	i .
74	t .
68	h .
31	1 .)
74	t (symbol name begin
69	i .
6d	m .
65	e .
73	s .)
10	begin application tag
08	symbol tag
06	cd length
04	name length
61	a (cd name begin
72	r .
69	i .
74	t .
68	h .
31	1 .)
70	p (symbol name begin
6c	l .
75	u .
73	s .)
05	variable tag
01	name length
78	x (name)
05	variable tag
01	name length
79	y (variable name)
11	end application tag
10	begin application tag
48	symbol tag (with share bit on)
01	reference to second symbol seen (arith1:plus)
45	variable tag (with share bit on)
00	reference to first variable seen (x)
05	variable tag
01	name length
7a	z (variable name)
11	end application tag
11	end application tag
19	end object tag

Implementation and Evaluation

- The OM2 binary encoding has been implemented based on the INRIA C library API
  - XOM (XML-encoded OM) to BOM (Binary OM) translation (based on Xerces/Sax)
  - BOM to XOM translation (based on Xerces/DOM)
  - implemented in C/C++, GPL, will be available soon from `http://www.faculty.iu-bremen.de/mkohlhase/kwarc/software`

Sample	XOM	BOM	B/X%	Gzip XOM	Gzip BOM
1	764 kb	206 kb	26.96%	198 kb	162 kb
2	659 kb	174 kb	26.4%	166 kb	142 kb
3	1359 kb	378 kb	27.81%	178 kb	149 kb

- still lacking: streaming, structure sharing, comparison to competition

Representing Integers on BOM

- Integers are encoded depending on how large they are.
- Small integers (token identifier 1)
  - Integers between -128 and 127 are encoded by a single byte.
  - Example: 16 is encoded as `0x01 0x10`.
  - Integers between -2^{31} and 2^{31} are encoded by tag 129 (long flag 6 set) followed 4-byte integer (most significant byte first).
  - Example: 128 is encoded as `0x81 0x00 0x00 0x00 0x80`

Representing Large Integers on BOM

- Large Integers (not arbitrary length but close)
  - token identifier 2
  - 1/4-byte number of digits,
  - sign/base byte: + (`0x2B`) or `-` (`0x2D`) or-ed with base mask bits: `0` for base 10 or `0x40` for base 16 0 for base 10 or 0x40 for base 16
  - a string of digits (as characters) in their natural order.
- Example 8589934592 (2^{33}) is encoded as `0x02 0x0A 0x2B 0x38 0x35 0x38 0x39 0x39 0x33 0x34 0x35 0x39 0x32`
- Even larger Integers by general streaming technology (new with OM2)

Streaming with the Binary Encoding

- some basic objects (integers, strings, bytearrays, and foreign objects) can be extremely large

- New: use the fifth bit as the streaming bit
- If the fifth bit is not set, this packet is the final packet of the basic object. (⇝ OM1 compatibility)
- If the bit is set, then more packets of the basic object will follow directly after this one.
- Idea: start processing the object immediately (most significant first)

Streaming a Large Integer

Hex	Meaning
22	begin streamed big integer tag
FF	255 digits in packet
2B	sign +
...	the 255 digits as characters
22	begin streamed big integer tag
FF	255 digits in packet
2B	sign + (disregarded)
...	the 255 digits as characters
2	begin final big integer tag
42	68 digits in packet
2B	sign + (disregarded)
...	the 68 digits as characters

A more efficient Representation for Large Integers?

- Bill Naylor: representation as base 10 or base 16 digits wastes half+ the space (why not use bytes directly)
- Problem: not many token identifiers left (have to think of OM3)
- Idea: use the sign/base byte (interpret byte encoding as base 256 digits)
  - remember: + (`0x2B`) or `-` (`0x2D`) or-ed with base mask bits: `0` for base 10 or `0x40` for base 16
  - new: use base mask bit 1 `0x80` for byte encoding.
  - Pro: OM1-compatible, space requirements like for small integers

Proposed Wording for OM2

This ’sign/base’ byte is + (`0x2B`) or `-` (`0x2D`) for the sign or-ed with the base mask bits that can be `0` for base 10 or `0x40` for base 16 or `0x80` for “base 256”. It is followed by the sequence of digits (as characters for bases 10 and 16 as in the XML encoing, and as bytes for base 256) in their natural order. For example, [. . .] and the hexadecimal number `xfffffff1` is encoded as `0x02 0x08 0x6b 0x66 0x66 0x66 0x66 0x66 0x66 0x66 0x31` in the base 16 character encoding and as `0x02 0x04 0xFF 0xFF 0xFF 0xFI` in the byte encoding (base 256).

The Competition: Binary Formats for General XML

- WAP: special purpose tags for (subset of) html (similar to BOM)
- BOX (Binary Optimized XML) (`http://box.sf.net`)
  - use `[1] 3 OMA ... [3]` for `<OMA> ... </OMA>` and
  - `[2] 73 ... [3]`, where OMA is the 73^{th} element name in the document.
  - Attributes, namespaces, . . . similar.
  - simple, elegant, covers all XML, allows structure sharing
- BIM (Binary Encoded MPEG-7) (searchon `http://www.mpeg.org`)
  - schema-based (do not transmit default values, element names)
  - streaming by tree-update semantics (has someone seen the implementation?)
- W3C has chartered a WG on binary XML formats (wait for Rec, use BOM in interim)

Conclusions and Further Work

- BOM gives significant space reductions even without full sharing
- BOM is somewhat less essential in OM2, since XOM allows sharing
- More space savings for large integers possible.
- BOM may become obsolete, but is good interimsolution (at least)
- further evaluation needed:
  - comparison to generic approaches: BiM BOX
  - parse/generation time comparison (not just space)
  - large scale examples
- Call: for an OM test/benchmark suite (maybe in a new OM network?)