University of Glasgow

The Presentation Layer

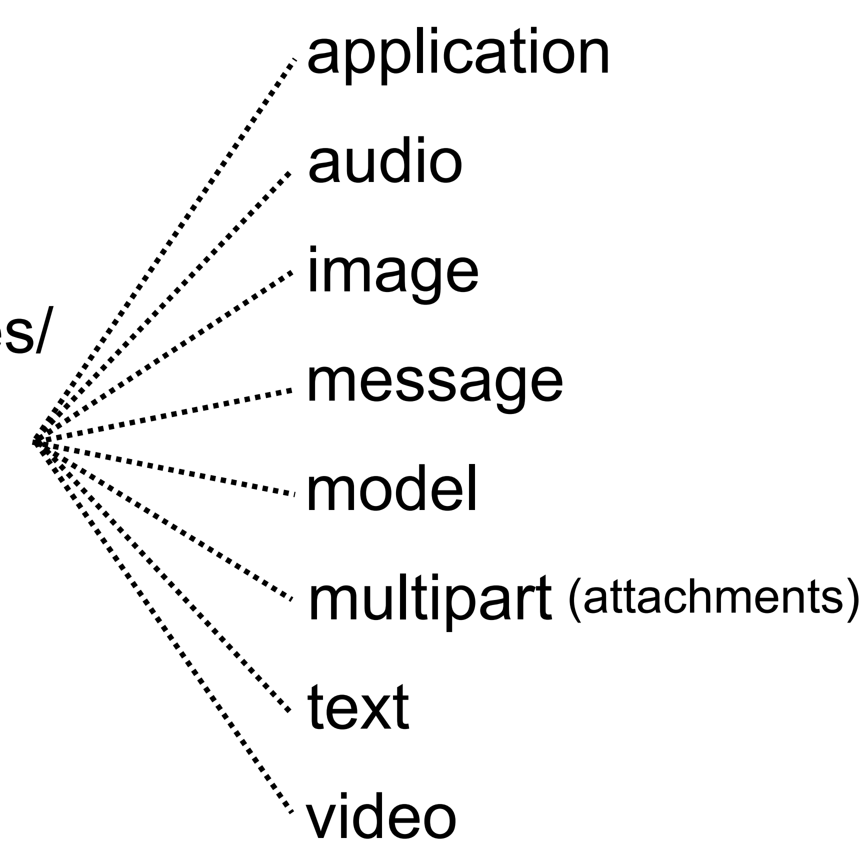
Networked Systems 3
Lecture 16

Presentation Issues

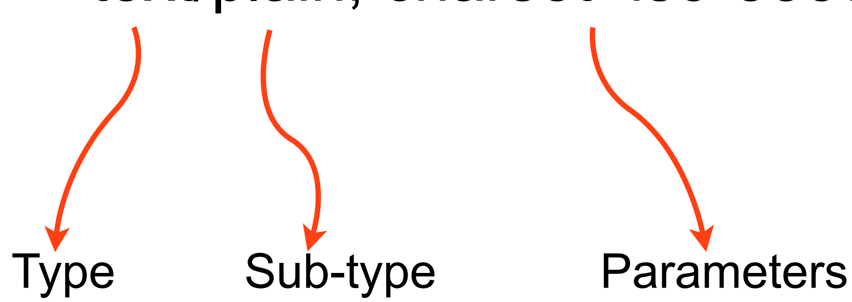
- Managing the presentation, representation, and conversion of data:
  - Media types and content negotiation
  - Channel encoding and format conversion
  - Internationalisation, languages, and character sets
- Common services used by many applications

Media Types

- Many data formats are not self-describing
- Standard *media types* identify the format of the data – signalled in the protocol
  - http://www.iana.org/assignments/media-types/
  - Categorise formats into eight *top-level* types
  - Each has many *sub-types*
  - Each sub-type may have parameters



text/plain; charset=iso-8859-1



The MIME Framework

- Email originally specified for 7-bit ASCII text
- Multipurpose Internet Mail Extensions (MIME) added support for other content types:
  - Uses three new headers to identify content:

```
MIME-Version: 1.0
Content-Type: text/plain; charset=iso-8859-1
Content-Transfer-Encoding: base64
```

  - Virtually identical mechanisms adopted by HTTP, etc.

Session Description Protocol

- A textual format to describe multimedia sessions
  - Uses media types to specify media type, subtype, and parameters

```
v=0
o=csp 2890844526 2890842807 IN IP4 10.47.16.5
s=-
c=IN IP4 224.2.17.12/127
t=2873397496 2873404696
m=audio 49170 RTP/AVP 99
a=rtpmap:99 vorbis/44100/2
a=fmtp:99 configuration=AAAAAZ2f4g9NAh4aAXZvcmJpcwA...
```

```
audio/vorbis; configuration=AAAAAZ2f4g9NAh4aAXZvcmJpcwA...
```

Content Negotiation

- Multimedia sessions need to negotiate codecs
  - A wide variety of codecs exist, and new codecs frequently introduced
- Two stage *offer/answer* model for negotiation
  - *Offer* lists supported codecs in order of preference
  - The receiver picks highest preference codec it also supports, includes this in its *answer*
  - Negotiates a common supported codec in one round-trip time

Offer/Answer Example

```
[Offer]  v=0
         o=alice 2890844526 2890844526 IN IP4 atlanta.example.com
         s=
         c=IN IP4 atlanta.example.com
         t=0 0
         m=audio 49170 RTP/AVP 0 8 97
         a=rtpmap:0 PCMU/8000
         a=rtpmap:8 PCMA/8000
         a=rtpmap:97 iLBC/8000
         m=video 51372 RTP/AVP 31 32
         a=rtpmap:31 H261/90000
         a=rtpmap:32 MPV/90000

[Answer] v=0
         o=bob 2808844564 2808844564 IN IP4 biloxi.example.com
         s=
         c=IN IP4 biloxi.example.com
         t=0 0
         m=audio 49174 RTP/AVP 0
         a=rtpmap:0 PCMU/8000
         m=video 49170 RTP/AVP 32
         a=rtpmap:32 MPV/90000
```

Receiver picks a subset of the media subtypes – and their parameters – to accept → includes them in the answer

Source: RFC 4317

Negotiation in Other Systems

- Similar negotiation frameworks exist in many other systems
  - An HTTP client can send an `Accept:` header listing media types it understands; server will try to format response to match
- Email one of the few widely used applications *without* format negotiation

Channel Encoding

- If the protocol is textual, how do you transport binary data?
  - Encode binary data in a textual format for transfer
    - What is binary data? What is an appropriate textual format?
  - Signal that the content has been encoded
    - The MIME `Content-Transfer-Encoding:` header
    - May require negotiation of an appropriate transfer encoding, if data passing through several systems

What is Binary Data?

- Data that cannot be represented within the textual character set in use
  - If using 7 bit ASCII text, any data using all eight bits
    - Example: very old versions of `sendmail` used the 8th bit to signal that quoted data was present, stripping it off data on input, since email was guaranteed to be 7 bit ASCII only
  - If using EBCDIC, any unassigned character
  - If using UTF-8, invalid multi-byte sequences
- Must be *encoded* to fit the character set in use

Coding Binary Data as Text

- Issues when designing a binary coding scheme:
  - *Must be backwards compatible with text-only systems*
  - Some systems only support 7-bit ASCII
  - Some systems enforce a maximum line length
  - Must survive translation between character sets
    - Legacy systems using ASCII, national extended ASCII variants, EBCDIC, etc.
  - Must not use non-printing characters
  - Must not use escape characters (e.g. $ \ # ; & “ ”)

Base 64 Encoding

- Standard encoding of binary data to textual format
  - Encode 3 bytes (24 bits) into four 6 bit values
  - Represent those values as printable characters as shown
  - Pad to 3 byte boundary using = characters
  - Encode no more than 76 characters per line

000000	A	010000	Q	100000	g	110000	w
000001	B	010001	R	100001	h	110001	x
000010	C	010010	S	100010	i	110010	y
000011	D	010011	T	100011	j	110011	z
000100	E	010100	U	100100	k	110100	0
000101	F	010101	V	100101	l	110101	1
000110	G	010110	W	100110	m	110110	2
000111	H	010111	X	100111	n	110111	3
001000	I	011000	Y	101000	o	111000	4
001001	J	011001	Z	101001	p	111001	5
001010	K	011010	a	101010	q	111010	6
001011	L	011011	b	101011	r	111011	7
001100	M	011100	c	101100	s	111100	8
001101	N	011101	d	101101	t	111101	9
001110	O	011110	e	101110	u	111110	+
001111	P	011110	f	101111	v	111111	/
						(pad)	=

Base 64 Encoding

Binary data: five bytes `10010111 01001101 11101011 00001101 01110101`

Split into six bit chunks, padding with zero bits `100101 110100 110111 101011 000011 010111 0101`*`00`*

Encode, using look-up table, and pad `l03rDXU==`

Average 33% expansion of data (3 bytes → 4)

Quoting Binary Data

- If only a small amount of binary data, can be easier to *quote* the non-textual values
  - Use a special *escape character* to signal start of quote
  - Signal value of un-representable data
- Two common approaches:
  - MIME quoted printable
  - URL encoding

Quoted Printable Encoding

- Convert occasional 8-bit values into a format that can be represented in 7-bit ASCII
  - The escape character is =
    - The escape character is represented as =3d if it appears in the text
  - Replace each 8-bit value with the escape character, followed by the hexadecimal value of the byte being quoted
  - E.g. the iso-8859-1 string straße is quoted as stra=dfe

URL Encoding

- URLs only permitted to contain alphanumeric characters plus $-_.+!*’()
  - All other characters must be encoded before the URL is used
- URL encoding similar to quoted printable, but uses % as the escape character
  - E.g. the iso-8859-1 string straße is quoted as stra%dfe

Internationalisation (i18n)

- What character set to use?
  - A national character set? ASCII, iso-8859-1, koi-8, etc.
    - Need to identify the character set and the language
    - Complex to convert between character sets
  - Unicode?
    - A single character set that can represent (almost?) all characters, from (almost?) all languages
    - 21 bits per character (0x000000 – 0x10FFFF)
    - Several representations (e.g. UTF-8, UTF-32)
    - Just represents characters – still need to identify the language

Unicode and UTF-8

- *Strong recommendation*: Unicode in UTF-8 format
  - UTF-8 is a variable-length coding of unicode characters

Unicode character bit pattern:		UTF-8 encoding:
`00000000 00000000 0zzzzzzz`	→	`0zzzzzzz`
`00000000 00000yyy yyzzzzzz`	→	`110yyyyy 10zzzzzz`
`00000000 xxxxyyyy yyzzzzzz`	→	`1110xxxx 10yyyyyy 10zzzzzz`
`000wwwxx xxxxyyyy yyzzzzzz`	→	`11110www 10xxxxxx 10yyyyyy 10zzzzzz`

  - Backwards compatible with 7-bit ASCII characters
    - Codes in the ASCII range coded identically, all non-ASCII values are coded with high bit set
    - No zero octets occur within UTF-8, so it can be represented as a string in C
  - Widely used in Internet standard protocols

Unicode: Things to Remember

- Unicode just codes the characters, need to code the language separately
  - Different languages have very different rules!
    - Is text written left-to-right or right-to-left?
    - How to sort? e.g. in German, ä sorts after a, in Swedish, ä sorts after z
    - How to do case conversion and case insensitive comparison? e.g., in German, `toupper("straße") = "STRASSE"`
    - How to handle accents? ligatures? ideograms? etc.
  - At the protocol level:
    - Code the characters as UTF-8 and specify the language
    - Let the application-layer programmer worry about using the data!

Language Tags

- IETF maintains standard for identifying languages
  - Surprisingly complex!
  - RFC 4646 describes syntax and semantics of language tags, and rules for how to register new tags (59 pages)
  - RFC 4647 explains how language tags can be compared (20 pages)
  - The list of registered languages is separate

en-GB
English as used in Great Britain

zh-Hans-CN
Chinese written using the Simplified script as used in mainland China

sl-IT-nedis
Slovenian as used in Italy, Nadiza dialect

de-Latn-DE-1996
German, Latin script, orthography of 1996

Sending Raw Binary Data

- Many protocols send binary data directly, not encoded in textual format
  - E.g. TCP/IP headers, RTP, audio-visual data
- Two issues to consider:
  - Byte ordering
  - Byte size

Byte Order

- The Internet standard *network byte order* is big endian
  - Must convert data between *host* and *network* forms

```
#include <arpa/inet.h>
uint16_t htons(uint16_t hs);
uint16_t ntohs(uint16_t ns);
uint32_t htonl(uint32_t hl);
uint32_t ntohl(uint32_t nl);
```

  - Frequent source of bugs, since Intel CPUs are little endian

Byte Size

- Early Internet protocols designed at a time when not all machines used eight bit bytes
  - Many protocols use the term *octet* for precision, when talking about eight bit values
  - Generally possible to ignore the distinction now…
- But… How big is an integer? 16, 32, or 64 bits How is a floating point value represented? Still need careful specification

Questions?