Smart Markup-Aware Applications with Apache 2

Nick Kew
WebThing Ltd

http:/ / apache.webthing.com/

The author is available for work!

Contents

1. Preamble
2. Case Study: modularising Site Valet
   + Live Demo
3. Platform for XML Applications
4. Reusable Markup Modules
   + Live Demo (you never knew ApacheCon looked like that)!
   Likely to be the most interesting section for most people.
5. Developing a Module
   Writing a pipelined filter module. Integrating a parser.
   With code from a real (useful, operational) module.

Contents

1. Preamble

What am I doing, where am I coming from, how does Apache 2 fit?

2. Case Study: modularising Site Valet

Describes how Site Valet markup-analysis and processing applications have moved from monolithic to modular, and the benefits this has brought.

3. Platform for XML Applications

mod_xml exposes SAX and DOM APIs for C/C++ applications, XMTP and XSLT filters, and a framework for XMLRPC. This module remains experimental, but is the basis for later work.

Contents

4 Reusable Markup Modules

A filter module works with any handler. With a proxy, you can filter the entire web, and reusability becomes almost automatic! We demonstrate how a markup-aware filter can serve to correct or transform HTML, or build a powerful general-purpose publishing system.

5. Developing a Module

For developers: Choosing a parser, and integrating it into Apache. Discusses the anatomy of a module, and illustrates it with code extracts from mod_proxy_html.

Manifesto

I have a dream ...

The Information infrastructure can liberate us from the shackles of geography. The Virtual workplace (, school, etc) frees us from the chore of the daily commute, yet brings us into more productive regular contact with colleagues worldwide.

* Technology: advance the functionality available to developers and users.
* Standards: prevent lock-in and stagnation.
* Inclusivity: no physical barriers to accessibility.
* Public acceptance: good technology can help, but bad technology gives us all a bad name.

What can I do?

* My household electricity supply supports ... appliances from many suppliers; it's my choice.

* Likewise my website supports ... browsers from many suppliers ... and more importantly, their users!

* My work ... aims to further Web technology while holding to this fundamental principle.

* Site Valet: QA, Standards, Accessibility - tools for Analysis and Reporting

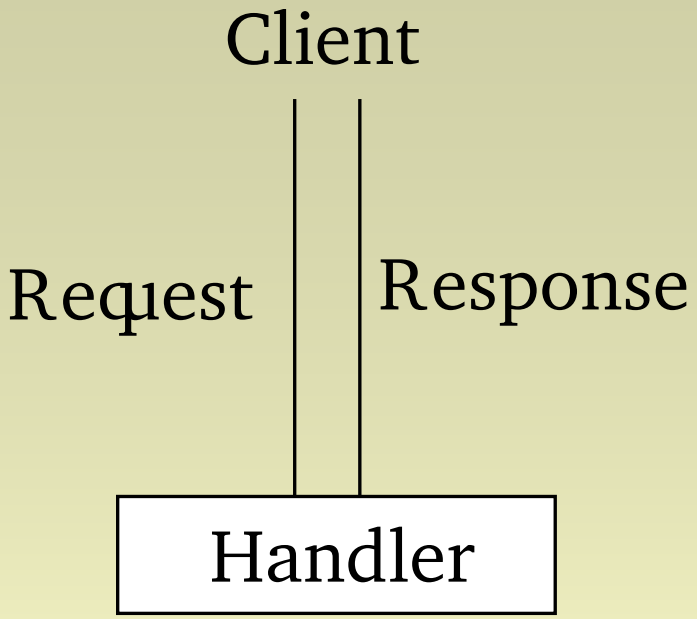
* Modules for publishing and fixup.

Web Server Technology

* CERN HTTPD 1994: frontend for image processing apps.
* Apache and the first WebThing: end-1995
* CGI can do anything - with a monolithic architecture! Likewise mod_perl, servlets, ...
* Apache 2.0 gives us reusable modules and simplified handlers.

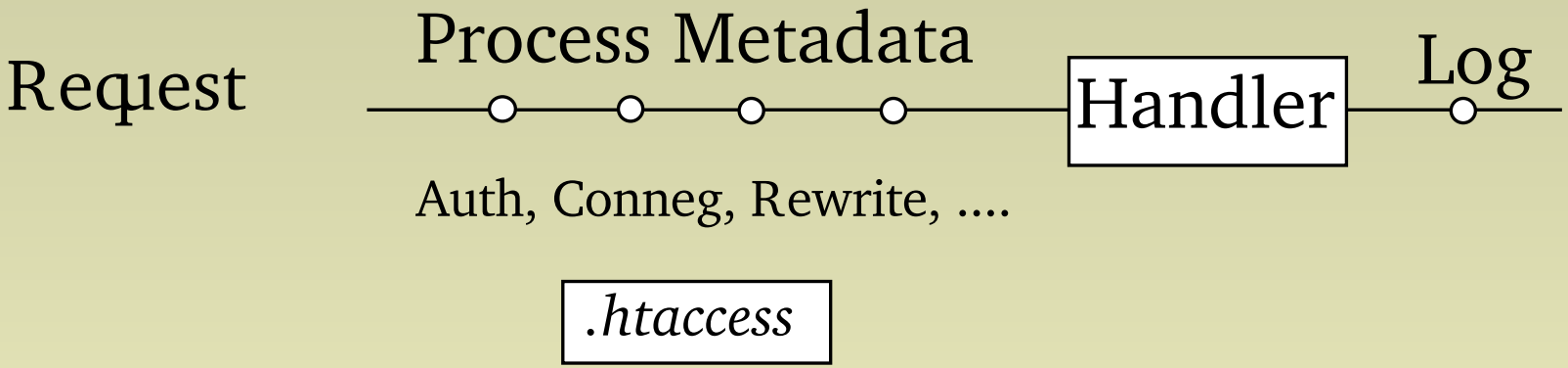
Minimal Webserver

HTTP 0.9 **or** 1.x



Apache 1.x

Taking advantage of HTTP/1.x Headers

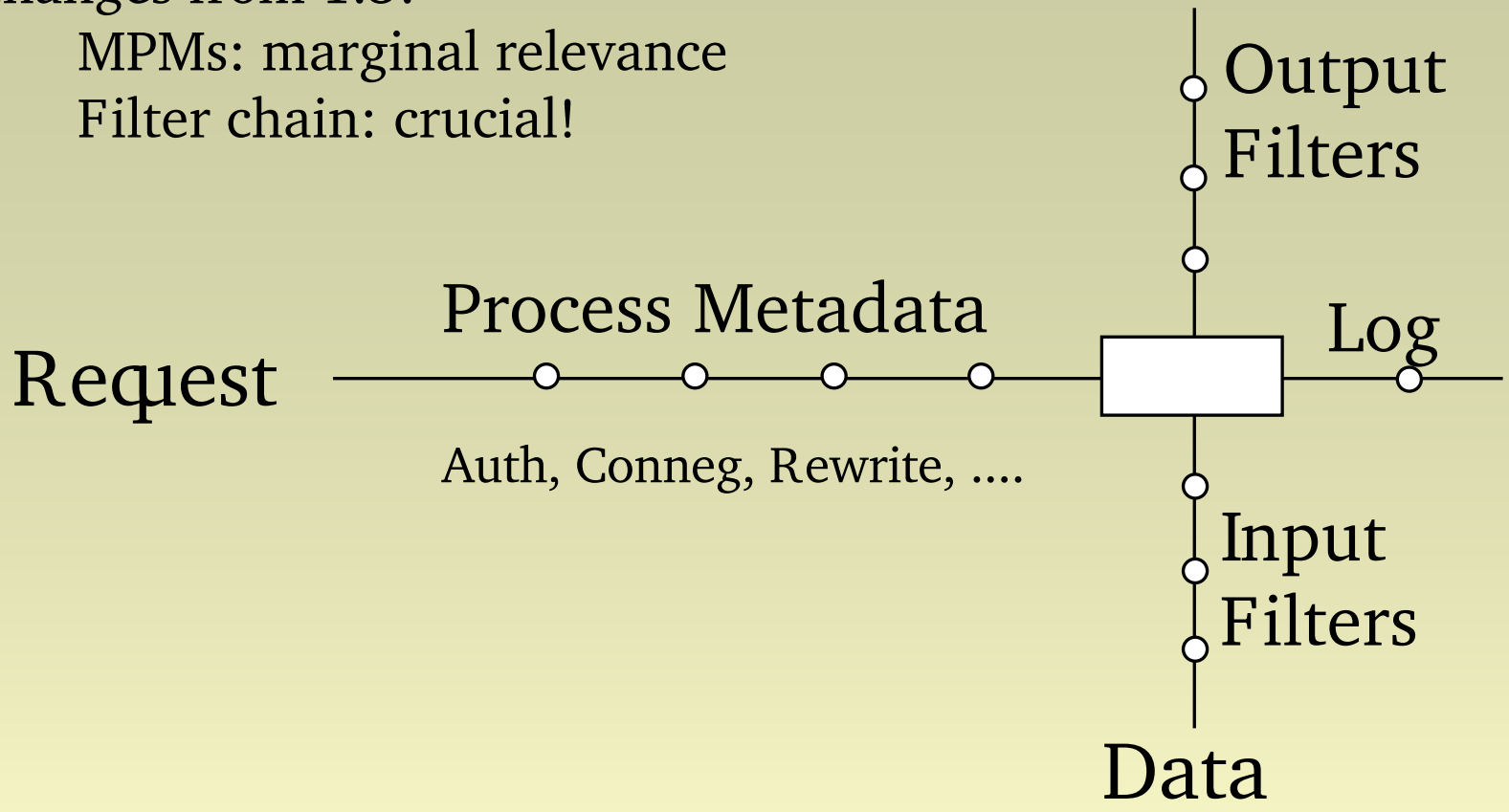


Experiment: XSLT module: nope, XSLT in CGI much better!

Apache 2.0

Changes from 1.3:

* MPMs: marginal relevance
* Filter chain: crucial!



Applications

Steps in Validation

- Decode request
- Fetch/ Upload document
- Sniff doctype/ charset
- Transcode to UTF-8
- Run validating parser
- (parse messages)
- prepare response

Accessibility Analysis

- Decode request
- Fetch/ Upload document
- Run parser
- prepare response

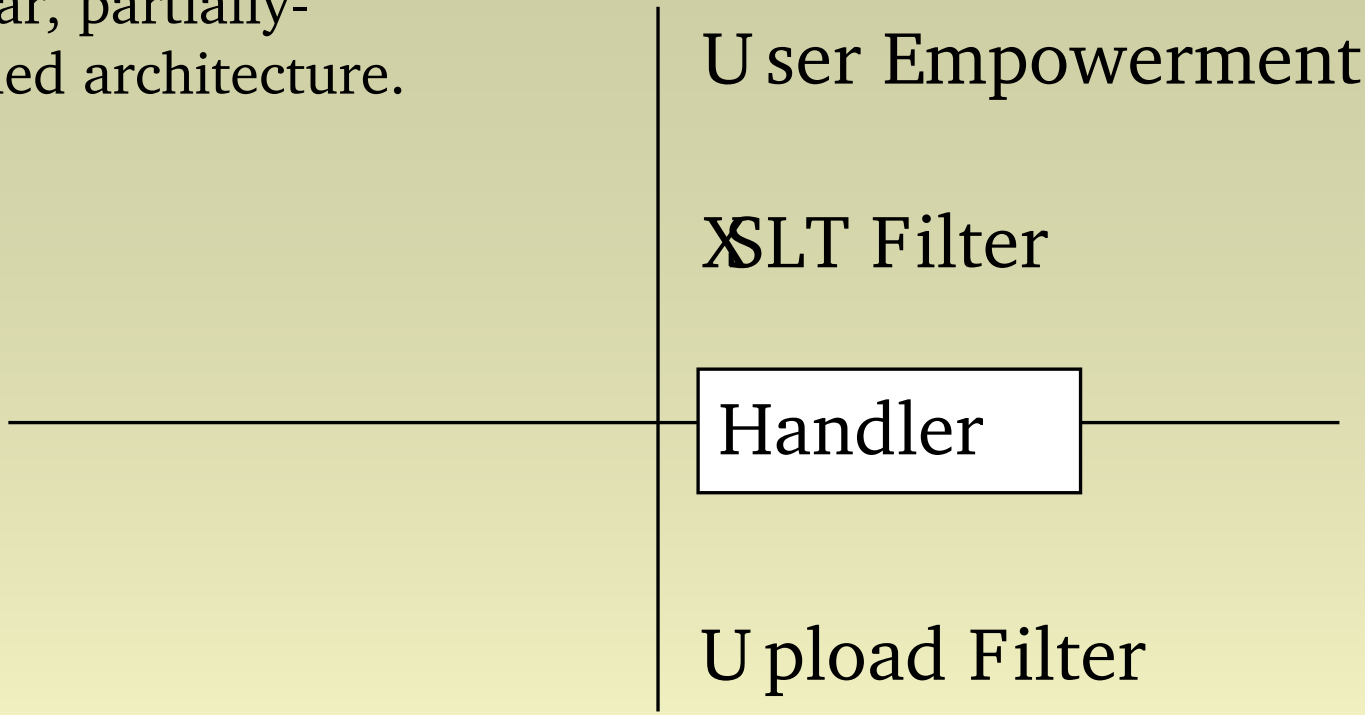
Case Study: Site Valet

* Apache 1.3: Do everything in the Handler
* *Comparison: the W3C Validator*

* Apache 2.0: Modular architecture
* Performance, Flexibility, Modularity, Reusability
* Example: easy response to Client demand
* Live Demonstration

Site Valet

Modular, partially-pipelined architecture.



mod_upload

upload-filter

* Decodes File Uploads
* Parses HTML Form data to a Table
* Handler gets the upload without MIME-gunk as input

tmpfile-filter

* Ensures synchronous input

Exports:

```
apr_table_t* mod_upload_form(request_rec* r)
```

XSLT Module

mod_xml_gnome_xslt

* Optimised XSLT Filter
* Progressive parse fits Apache architecture
* Precompile and cache Transforms
* Accept XML or Doc Tree
* Can run XSLT on HTML

Exports

void modxmlGnomeSetXSLT(request_rec* r, const char* name)

void modxmlGnomeXSLTDoc(request_rec* r, xmlDocPtr doc)

mod_accessibility

* Transform HTML 4and XHTML 1.0
* Accessibility, Usability, User Choice
* Fast/ efficient single-pass SAXparse
* No browser requirement (not even cookies)
* Strip problematic presentational markup
* Linearisation options - user control
* Data discovery
* Overview options
* Customised publishing, transforms, SSI-like
* Live Demo later

mod_tee

mod_tee

* A rapid response to user demand
* As a module, it services **all** handlers
* User can now get email copy of any Valet report (or even page) on-demand

Handlers

mod_validator

* Same purpose as W3C Validator
* Modular architecture makes it at once simpler and more powerful

mod_htnorm

* Accessibility analysis (AccessValet)
* Re-use of all the same components as Validator

Comparison

	W3C	Page Valet	AccessValet
Read Req(1)	CGI.pm	getargs()	getargs()
Get Data (1)	LWP	HTTPClient	nanoHTTP
Upload Req	CGI.pm	mod_upload	mod_upload
Sniff Doc	Custom	Custom	N/A
Transcode	Text::iconv	iconv(3)	N/A
ParseDoc	onsgmls	osp/xercesc	libhtnorm
Parse Result	Custom	N/A	N/A
Prepare Report	Custom	XSLT	XSLT
Presentation Opts	N/A	mod_acc	mod_acc
Email Option	N/A	mod_tee	mod_tee
Key			
reusable code	one-off code	filters	fork()

Reusable Modules

General-purpose components

* Already seen: upload, xslt, accessibility, tee
* mod_xml: platform for XML Apps and Webservices
* filter + proxy: universal transform
* mod_proxy_html (single-purpose spinoff)
* Yet Another Publishing System
* Live Demo: mod_accessibility mirror (unexpected high traffic!)

mod_xml

Input Filters: XMTP, XMLRPC
Handlers: application hooks for SAX and DOM
Multi-parser (Xerces/ Xalan; libxml2/ libxslt)
XSLT Filters
Internal Optimisations; persistence

Exports: (see also XSLT module)

```
void* get_apphook(server_rec*, const char*)
void* set_apphook(server_rec*, const char*, void*)
void* set_apphook_if(server_rec*, const char*,
                     void* (*) (server_rec*, void*), void*)
```

mod_xml API

SAX and SAX2:

* Just define the callbacks

DOM:

* modxml_main

Both:

* hooks available for persistent objects
* tight integration with filters
* pre-parse and post-parse callbacks available for things like init and cleanup

Content Filtering with SAX

Examples: mod_accessibility and mod_proxy_html
An output filter works with any handler, so is universally reusable

* Fixup of bad markup
* Template-based publishing
* HTML:SSI and the like
* XML: Namespace-based dispatch
* Markup/ Ruleset-driven rewrites
* Data and Metadata discovery

mod_proxy_html

* single-purpose module
* mod_accessibility spinoff

<a href="**http://internal.server/**foo/ bar.baz">a link that won't work from the outside world</ a>

Proxy with mod_proxy_html

<a href="**http://proxy.example.com/internal.server/**foo/ bar.baz">a link that will work from the outside world</ a>

Detailed (developer-level) discussion later

mod_accessibility

* Transform HTML 4and XHTML 1.0
* Accessibility, Usability, User Choice
* Fast/ efficient single-pass SAXparse
* No browser requirement (not even cookies)
* Strip problematic presentational markup
* Linearisation options - user control
* Data discovery
* Overview options
* Customised publishing, transforms, SSI-like
* Live Demo coming up ...

Markup Fixup

* Normalisation to HTML4or XHTML1 Strict

```
<p align="right">paragraph text
<font ...>different text</font>
and some more text.</p>
```

* Text-only representation

```
<img src="apachecon.png" alt="ApacheCon">
```

Analysis: TOC

<h1>....</h1>
bla bla bla
<h2>....</h2>
.....
<table summary="...." otherattributes>
<caption>...</caption>
.....
</table>

Linearisation

```
<table><tr>
<td>first text presented in left column</td>
<td><img src=...  alt="image in middle"></td>
<td>second text presented in right column</td>
</tr></table>

<div class="table">
<p>first text ...</p>
<p><img src=...  alt="image in middle"></p>
<p>second text ...</p>
</div>
```

Data Discovery

```
<p>To learn more, <a href="more.html">click here</a>.</p>
```

```
<p>To learn more, <a href="more.html" title="Data Discovery">click here</a>.</p>
```

Templating

```
Var sponsor file    /path/to/sponsors
```

startElement handler:

```
<sponsor/ >
<p class="sponsor">Sponsored by ..... </ p>
```

comment handler:

```
<!--#include file="/ path/ to/ sponsors"-->
<p class="sponsor">Sponsored by ..... </ p>
```

Publishing

<head>,</head>,<body>,</body> are implied, so we can hook site-wide templates on them:

```
<link rel="stylesheet" type="text/css" href="style.css" />
</head>
<body>
<div id="logo"><a href="/"><img src="logo.gif" alt="Site Valet"></a></div>
body text .....
<div id="navbar">.....</div>
<div id="footer">.....</div>
</body>
```

mod_accessibility

Live Demo Now!

Writing a Module

Apache API + Parser = Module

Using mod_proxy_html as an example, we go through the steps in developing a markup-processing module.

Choosing a Parser

	SAX/push	HTML	Threadsafe	XSLT
Expat	Yes	No	Yes	sablotron
libxml2	Yes	Heuristic	Yes	libxslt
XercesC	No	No	Yes	XalanC
Tidy	No	Heuristic	No	N/A
OpenSP	No	Rigorous	No	(OpenJade)

A No can be overcome, but it's more work and complexity.

A rigorous HTML parser is required for Validation, but a heuristic parser suffices for most purposes - including our filters.

A Filter module

```
static void proxy_html_hooks(apr_pool_t* p) {
  ap_register_output_filter("proxy-html", proxy_html_filter,
        proxy_html_filter_init, AP_FTYPE_RESOURCE) ;
}
module AP_MODULE_DECLARE_DATA proxy_html_module = {
        STANDARD20_MODULE_STUFF,
        proxy_html_config,
        proxy_html_merge,
        NULL,
        NULL,
        proxy_html_cmds,
        proxy_html_hooks
};
```

Basic Data Struct

```
typedef struct {
  struct urlmap* next ;
  const char* from ;
  const char* to ;
}urlmap ;

typedef struct {
  htmlSAXHandlerPtr sax ;
  ap_filter_t* f ;
  urlmap* map ;
  htmlParserCtxtPtr parser ;
  apr_bucket_brigade* bb ;
}saxctxt ;
```

filter_init(1)

```
static int proxy_html_filter_init(ap_filter_t* f) {
  saxctxt* fctx ;
  xmlCharEncoding enc = xmlParseCharEncoding(
      ctype2encoding(f->r->pool, f->r->content_type)) ;

      /* remove content-length filter [chopped] */

  fctx = f->ctx = apr_pcalloc(f->r->pool, sizeof(saxctxt)) ;
  fctx->sax = setupSAX(f->r->pool) ;
  fctx->f = f ;
  fctx->bb = apr_brigade_create(f->r->pool,
      f->r->connection->bucket_alloc) ;
  fctx->map = ap_get_module_config(f->r->per_dir_config,
      &proxy_html_module);
```

filter_init(2)

```
/* The parser converts to utf-8 internally */
ap_set_content_type(f->r, "text/ html;charset=utf-8") ;

/* We've unset Content-Length,  so we'll chunk if possible */
if ( f->r->proto_num >= 1001 ) {
  if ( ! f->r->main && ! f->r->prev )
    f->r->chunked = 1 ;
}
/* hack relies on enc being set if the encoding is exotic */
fctx->parser = htmlCreatePushParserCtxt
      ( fctx->sax , fctx, "    ", 4, 0, enc) ;
return OK ;
}
```

Auto-Config

```
static saxctxt* check_filter_init (ap_filter_t* f) {

  if ( f->r->proxyreq&& f->r->content_type ) {
    if ( strncasecmp(f->r->content_type, "text/ html", 9)
        && strncasecmp(f->r->content_type,
            "application/ xhtml+xml", 21) ) {
      ap_remove_output_filter(f) ;
      return NU LL ;
    }
  }
  if ( ! f->ctx )
    proxy_html_filter_init(f) ;
  return f->ctx ;
}
```

Filter callback

```
static int proxy_html_filter(ap_filter_t* f,
                             apr_bucket_brigade* bb) {
  apr_bucket* b ;
  const char* buf = 0 ;
  apr_size_t bytes = 0 ;
  saxctxt* ctxt = check_filter_init(f) ;
  if ( ! ctxt )
    return ap_pass_brigade(f->next, bb) ;
 /* main loop (next slide) */
  apr_brigade_destroy(bb) ;
  return APR_SUCCESS ;
}
```

Main Loop

```
for ( b = APR_BRIGADE_FIRST(bb) ;
      b != APR_BRIGADE_SENTINEL(bb) ;
      b = APR_BUCKET_NEXT(b) ) {
  if ( APR_BUCKET_IS_EOS(b) ) {
    htmlParseChunk(ctxt->parser, buf, 0, 1) ;
    htmlFreeParserCtxt(ctxt->parser) ;
  }else if ( apr_bucket_read(b, &buf, &bytes,
          APR_BLOCK_READ)  == APR_SUCCESS ) {
    htmlParseChunk(ctxt->parser, buf, bytes, 0) ;
  }else {
    ap_log_rerror(APLOG_MARK, APLOG_ERR, 0, f->r,
          "Error in bucket read") ;
  }
}
```

SAX Handlers

```
static htmlSAXHandlerPtr setupSAX(apr_pool_t* pool) {
  htmlSAXHandlerPtr sax
       =    apr_pcalloc(pool, sizeof(htmlSAXHandler) ) ;
  sax->startDocument = pstartDocument ;    /* DOCTYPE */
  sax->endDocument = pendDocument ;        /* SENDEOS*/
  sax->startElement = pstartElement ;
  sax->endElement = pendElement ;          /* skip if empty */
  sax->characters = pcharacters ;          /* HTML Escape */
  sax->comment = pcomment ;                /* passthrough */
  sax->cdataBlock = pcdata ;               /* passthrough */
  return sax ;
}
```

startElement

```
typedef struct {
        const char* name ;
        const char** attrs ;
}elt_t ;

static void pstartElement(void* ctxt, const xmlChar* name,
                          const xmlChar** attrs ) {
  saxctxt* ctx = (saxctxt*) ctxt ;
  static elt_t linked_elts[] = {
      / * table of HTML elements having %URI attributes */
      / * in the form of:
      {"a" , {"href" , NULL }},
      */
  };
```

startElement(2)

```
  ap_fputc(ctx->f->next, ctx->bb, '<') ;
  ap_fputs(ctx->f->next, ctx->bb, name) ;
  if ( attrs ) {
    const char** linkattrs = 0 ;
    const xmlChar** a ;
    elt_t* elt ;
    for ( elt = linked_elts;  elt->name != NULL ; ++elt )
      if ( !strcmp(elt->name, name)  {
        linkattrs = elt->attrs ;
        break ;
      }
    / *      loop over attributes (next slide)        */
  }
  ap_fputc(ctx->f->next, ctx->bb, '>') ;
}
```

attributes

```
for ( a = attrs ; *a ; a += 2 ) {
  const xmlChar* value = a[1] ;
  if ( linkattrs && value ) {
    int is_uri = 0 ;
    const char** linkattr = linkattrs ;
    do {
      if ( !strcmp(*linkattr, *a) ) {
        is_uri = 1 ;
        break ;
      }
    }while ( *++linkattr ) ;
    if ( is_uri ) {
      /* remap value */
    }
  }
```

attributes(2)

```
  if ( is_uri ) {
    urlmap* m ;
    for ( m = ctx->map ; m ; m = (urlmap*)m->next ) {
      if ( ! strncasecmp(value, m->from, strlen(m->from) ) ) {
        value = apr_pstrcat(ctx->f->r->pool, m->to,
              value+strlen(m->from) , NULL) ;
        break ;
      }
    }
  }
if ( ! value )
  ap_fputstrs(ctx->f->next, ctx->bb, " ", a[0], NULL) ;
else
  ap_fputstrs(ctx->f->next, ctx->bb, " ", a[0],
            "=\"", value, "\"", NULL) ;
;
```

References

All the modules discussed are at
http://apache.webthing.com/

The case study, Site Valet, is at
http://valet.webthing.com/

The author's personal homepage is at
http://www.webthing.com/~nick/

The author is available for work!

Page Note 1:
Prepared using Kpresenter, which I'm learning on-the-fly.
ApacheCon require notes to be supplied in PDF format.

Page Note 2:
The talk is in five unequal sections, to be accompanied by live demos (assuming I have a working 'net connection for the presentation!).

Page Note 3:

Page Note 4:

Page Note 5:
Here in the UK - and to varying extents elsewhere - we have huge transport problems and congestion - and people who spend several hours a day just in commuting! We have a serious need to stop thinking "transport" and start thinking "communications" instead. Free up roads and rails for those with a genuine reason to travel, while at the same time improving communication and productivity at work!

Page Note 6:
My work supports QA, standards compliance and accessibility. There is no excuse for violating them and forcing users in to proprietary and unaffordable Client systems on the Web!

Page Note 7:
More historic information is at my homepage - http://www.webthing.com/~nick/

Page Note 8:
The minimal webserver accepts a request and returns a response. Within that, you *can* do anything, but it's a shame to reinvent the wheel ...

Page Note 9:
Apache 1.x, like other webservers, processes a request in stages with hooks for things like access control and content negotiation before the main handler. But if there's complex data processing, it still all has to be done in the main Handler.

Page Note 10:
Apache 2.0 adds a second dimension to processing. The Filter chain is effectively a Data axis, allowing modular, pipelined processing. We harness this axis to turn monolithic applications into reusable components.

Page Note 11:
Validation and Accessibility Analysis are two of the tasks performed by Site Valet. They share some common data processing steps. We handle these with reusable input and output filters, and thereby simplify the main handlers. We gain pipelining for free:-)

Page Note 12:

The W3C Validator (http://validator.w3.org/) still runs under Apache 1.3, and is a monolithic application performing a similar task to the Valet tools. We use it as a comparison.

Page Note 13:
The modular architecture uses one input and three output filters. Some of them are not strictly part of the application, but instead serve to empower users; more later.

Page Note 14:
The upload module handles file uploads, reducing the handler's workload.
http://apache.webthing.com/mod_upload/

Page Note 15:
The XSLT module serves to prepare reports from all handlers. It originated as part of mod_xml, and preserves some of the architectural optimisations.
http://apache.webthing.com/mod_xml/

Page Note 16:
mod_accessibility transforms outgoing HTML and presents options to users. It also offers SSI-like publishing. More later
http://apache.webthing.com/mod_accessibility/

Page Note 17:
mod_tee was a rapid response to a user's request to be able to receive reports by email. It "clones" an outgoing page, and is turned on/off by the Form data submitted. As an output filter module, it is automatically available to subscribers with all the valet tools!

Page Note 18:
Some of the Valet applications are still CGI. These two are modules, taking full advantage of the modular architecture.
http://apache.webthing.com/

Page Note 19:
Comparison: the W3C Validator and Page Valet perform the same task. Page Valet has just one area of "custom" (substantial, non-reusable) code to W3's three, and links to the parser library rather than running a commandline tool and parsing its output. Use of XSLT for report preparation - as opposed to custom templates in a monolithic architecture - makes it easy to offer a choice of views to users. Access Valet shares many of the same components as Page Valet.
http://valet.webthing.com/page/
http://validator.w3.org/
http://valet.webthing.com/access/

Page Note 20:
The talk now moves from the modularisation case study to focus on the modules. The most interesting are filters.

Page Note 21:
mod_xml is the "kitchen sink" markup-aware module. It was also my first work with Apache 2.0, and is experimental. Only two "real" applications have been deployed, and one of those has been superseded. Probably most interesting in terms of spinoff: the XSLT module is in regular use, and the input filters could be of interest for a webservices platform.

Page Note 22:
mod_xml exposes new APIs for SAX and DOM applications in C or C++

Page Note 23:
Perhaps the most interesting work with Apache is the SAX-based output filters. They are of course hugely faster and lighter on resources than XSLT, yet offer very useful processing, a nd can be added to more-or-less any application!

Page Note 24:
mod_proxy_html is the simplest of these modules. It does for HTML links what the ProxyPassReverse directive does for HTTP Headers.
http://apache.webthing.com/mod_proxy_html/

Page Note 25:
mod_accessibility is my "flagship" work in this area. It performs a number of different transformations to markup, and empowers everyone - especially users.
http://apache.webthing.com/mod_accessibility/

Page Note 26:
Simple functions include normalising to Valid/Strict HTML, and extraction of text-only pages. The user is presented with a menu of presentation options, so they only get the changes th ey really want.

Page Note 27:
We can prepare an instant and fully-linked table-of-contents from "major" elements.

Page Note 28:
We can linearise tables and frames (for the latter we also fetch and parse the framed pages).

Page Note 29:
We can fetch data from linked documents, in this case adding useful information to a meaningless link.

Page Note 30:
We can add publishing - which saves parsing the page a second time with SSI.

Page Note 31:
And we can make site-wide modifications. Unlike other such modules, we can do this without resorting to horribly botched and invalid HTML.

Page Note 32:
For a live demo of mod_accessibility, we'll show a mirror of apachecon (at http://mirrors.webthing.com/apachecon.com/) and take suggestions from the audience to view other sites thr ough an accessibility-enhanced proxy.

Page Note 33:
The final section deals with writing a markup filter module. We'll use a simple example: mod_proxy_html.

Page Note 34:
We're using SAX, and a progressive push-parser is required to work in a pipelined architecture. Expat and libxml2 both offer that, and are (coincidentally) also the two fastest XML parsers in existence according to the XML benchmark project. We select libxml2 for its HTML parser module.

Page Note 35:
Finally, I'll explain the essentials at a code level. Functions we'll look at later are highlighted in blue.

Page Note 36:
The essential data structs ...

Page Note 37:
The filter_init function is called before the main processing. We set up the parser and context. We also remove the Content-Length filter (which was generating bogus results in some cases). To avoid losing HTTP keep-alive, we invoke HTTP Chunked Encoding for the response.

Page Note 38:
Note the encoding: this module (unlike some in the family) doesn't support setting charset via the HTML META hack. Servers being proxied should send proper HTTP headers if using exotic charsets.

Page Note 39:
We don't know the MIME type of a proxied document until we've fetched it. If it's not HTML, then we don't want to filter it, so we quietly remove ourself from the filter chain.

Page Note 40:
The heart of a filter module is a callback function. The main handler feeds outgoing data through it.

Page Note 41:
Within the loop, htmlParseChunk feeds data to the SAX parser. This is the central interface between the Apache API and the markup library API.

Page Note 42:
The filter_init sets up callbacks for SAX events. The one that does most of the work is startElement; the others just write out (broadly) whatever went in.

Page Note 43:

Page Note 44:

Page Note 45:

Page Note 46:

Page Note 47: