Enabling Web Services by Embedding Axis2/C

Samisa Abeysinghe
WSO2 Inc.
Kaushalye Kapuruge
WSO2 Inc.

Outline

- Axis2/C introduction
- Plugging in environment
- XML in/out model
- AXIOM implications
- Understanding service_client
- Understanding service
- Engaging modules
- Leveraging full Web services stack

Axis2/C Introduction

- Axis2 SOAP engine architecture implemented in C
- Platform independence and portability in mind
  - Influenced by portable runtime (e.g. APR)
  - Ability to plug in allocators, error handling, log handling, and threading mechanisms
- With the intention of binding into any C based platform
  - Scripting languages such as PHP, Ruby, Perl
  - AJAX support – e.g. Firefox XPI, IE ActiveX


ApacheCon
WS-Addressing
WS-Security
WS-ReliableMessaging
WS-Policy
AXIS2
php
Perl
Ruby
US 2006

Design Characteristics

- Stick to the same Axis2 architectural design as much as possible
- Pseudo Object Oriented
  - Ops exposed in header
  - Data hidden in source
  - Macros to hide complex ops syntax
- Portability through environment concept
  - Made available to all functions
  - Encapsulate allocater, log, error, threading

the header axis2_engine.h contains:

```
/* the ops struct*/
struct axis2_engine_ops
{
   axis2_status_t (AXIS2_CALL
*
   send)(struct axis2_engine
*engine,
         const axis2_env_t *env,
         axis2_msg_ctx_t
*msg_ctx);
};

/* the struct itself */
struct axis2_engine
{
   axis2_engine_ops_t *ops;
};
```

```
/* the easy to use macro */
#define AXIS2_ENGINE_SEND(engine,
      env, msg_ctx) \
       ((engine)->ops->send(engine,
            env,msg_ctx))
```

the source engine.c contains:

```
/* the impl struct with data */
typedef struct
axis2_engine_impl
{
   axis2_engine_t engine;
   axis2_conf_ctx_t *conf_ctx;
} axis2_engine_impl_t;
```

Plug in Environment

- Main components
  - Memory allocator
  - Log
  - Error handling
  - Thread pool
- Can implement the interfaces with your choice
  - e.g. emalloc/efree for allocating/deallocating in PHP

Allocator Example

- Override default malloc implementation of allocator
- PHP custom malloc implementation would look like:

```
static void* PHP_AXIS2_CALL php_axis2_malloc_wrapper( axis2_allocator_t
                *allocator, size_t size) {
        return emalloc(size);
}
```

- Use allocator interface for overriding default implementation:

```
allocator->malloc_fn  = php_axis2_malloc_wrapper;
```

- Now when AXIS2_MALLOC built in macro is called, php_axis2_malloc_wrapper would get executed

```
#define AXIS2_MALLOC(allocator, size)  ((allocator)->malloc_fn(allocator, size))
```

XML in/out Model

- Axis2 is designed with XML in/out model in mind
- Engine takes in XML, process it and, gives out result in XML
- Keep this in mind when embedding
- If other forms need to be input/output
  - provide the wrapper layers to deal with conversion

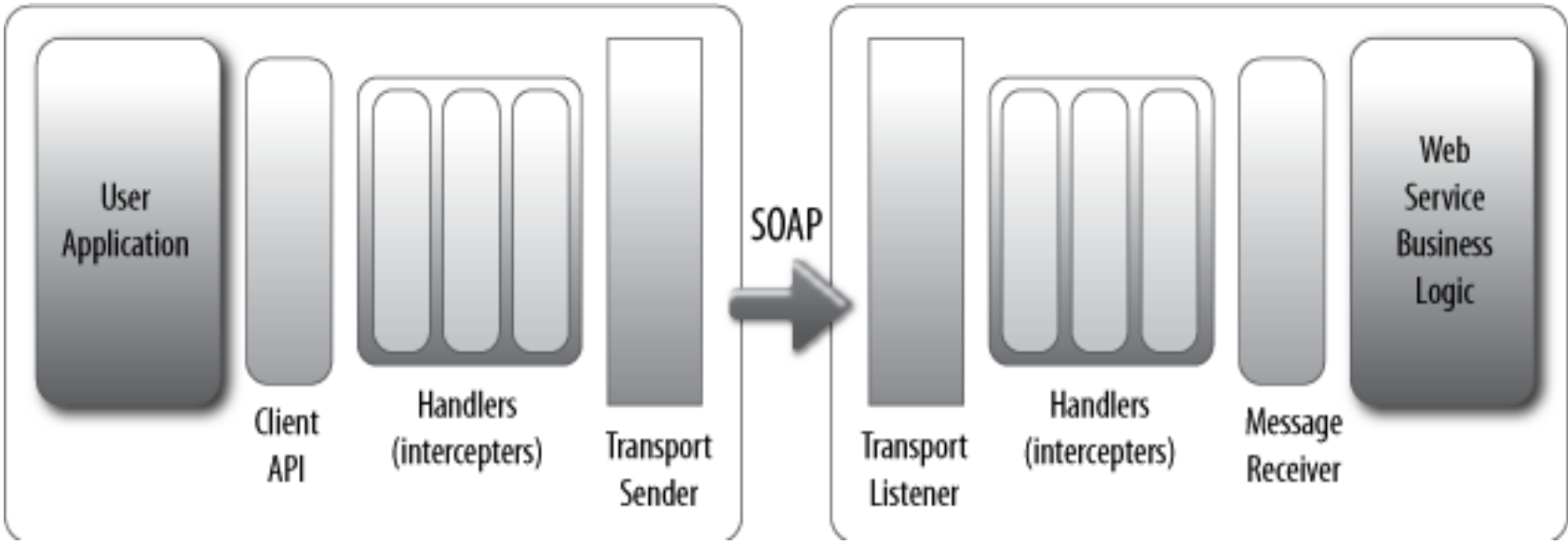
AXIOM Implications

- AXIOM
  - Lightweight and fast XML object model
- Axis2 engine expects XML to be in AXIOM format
- Need to convert back and forth from AXIOM to desired formats
- This is not that complex
  - PHP binding supports both SimpleXML and DOM as input/output formats
  - Firefox XPI takes in XML DOM and returns result in XML DOM format
  - All of those formats converted to AXIOM before being presented to the engine

Understanding service_client

- To consume Services.
- Needs to wrap this interface to support client side
- Takes an options parameter
- Supports synchronous and asynchronous modes of execution

Understanding service_client *contd*.



Wrapping service_client

- It is advisable to stick to a similar interface when wrapping
- Stick to the ground rules of XML in/out
- When this works and when you are confident, explore the contours

Wrapping service_client - PHP

```
$serviceClient = new Axis2ServiceClient("http://localhost:9090/axis2/services/echo");
$resPayload=$serviceClient->sendReceiveSimpleXML($smplxml);
printf("Response = %s \n", $resPayload);
```

Wrapping service_client - PHP

Wrapping code for constructor

```
AXIS2_GET_THIS(obj);
intern = (axis2_object*)zend_object_store_get_object(obj TSRMLS_CC);
client = axis2_svc_client_create(AXIS2_GLOBAL(env),
AXIS2_GLOBAL(client_home));

client_options = axis2_options_create(AXIS2_GLOBAL(env));

AXIS2_SVC_CLIENT_SET_OPTIONS(client, AXIS2_GLOBAL(env),
client_options);
intern->ptr = client;
intern->obj_type = PHP_AXIS2_SVC_CLIENT;
```

Wrapping service_client - AJAX

AJAX Web service client powered by Firefox XPI

```
var req = new SOAPHttpRequest();
req.soapVer = 1.1;
req.onreadystatechange  = listenStatus;
req.open("POST", "http://api.google.com/search/beta2", false);
req.send ( reqContent);
var resultContent = req. responseXML;
```

Wrapping service_client - AJAX

Wrapping code for WS invoke

```
options = axis2_options_create(env);
 svc_client = axis2_svc_client_create(env, client_home);
 if ( NS_SUCCEEDED (rv ) )     {
      ret_node = AXIS2_SVC_CLIENT_SEND_RECEIVE(svc_client, env,
cont_node);
}
 writer = axiom_xml_writer_create_for_memory(env, NULL,
AXIS2_TRUE, 0,                    AXIS2_XML_PARSER_TYPE_BUFFER);
om_output = axiom_output_create (env, writer);
AXIOM_NODE_SERIALIZE (ret_node, env, om_output);
mResponseText = PL_strdup ((char* )
AXIOM_XML_WRITER_GET_XML(writer, env) );
mStatus = SOAPHTTPREQUEST_HTTP_STATUS_SUCCESS;
```

Understanding service

- Service interface
  - Used to represent services provided
- It is this interface that you needs to wrap to support server side
- Looking at service samples may not help here
  - The idea is not to write services
  - Rather to provide a framework to help deploy services

Wrapping service - PHP

PHP Web service interface

```
function echo($text) {
    return $text;
}

$service->addFunction("echo");
$service->process();
```

Wrapping service - PHP

- Wrapping code for constructor

```
 php_worker = AXIS2_GLOBAL(php_worker);
conf_ctx = AXIS2_PHP_WORKER_GET_CONF_CTX(php_worker,
        AXIS2_GLOBAL(env));
conf = AXIS2_CONF_CTX_GET_CONF(conf_ctx,
         AXIS2_GLOBAL(env));
svc = AXIS2_CONF_GET_SVC(conf,
         AXIS2_GLOBAL(env), svc_info->svc_name);
if(NULL != svc){
       svc_info->svc = svc;
} else {
   svc_qname = axis2_qname_create(AXIS2_GLOBAL(env),
     svc_info->svc_name, NULL, NULL);
    svc_info->svc =axis2_svc_create_with_qname(AXIS2_GLOBAL(env),
      svc_qname);
 }
svc_info->msg_recv =
axis2_php_xml_msg_recv_create(AXIS2_GLOBAL(env));
```

Engaging Modules

- Modules pave the way to extend engine capabilities
- Mostly help with SOAP header processing
  - SOAP body could also be dealt with
- Powered by AXIOM representation
- Help with the WS-* specs
- WS-Addressing, WS-Security, WS-Reliable Messaging implemented as modules in Axis2/C
- Engaging a module ensures availability of WS spec that it deals with

Engaging Addressing - PHP

- Conditional engaging of addressing module on client side:

```
if (isset($qosAddr)) {
    $serviceClient->
     engageModule(Axis2Constants::MODULE_ADDRESSING);
    $serviceClient->setOption(Axis2Constants::ADDR_ACTION,
     "http://www.wso2.net/products/tungsten/c/demo/submit");
}
```

Engaging Addressing - PHP

- Wrapping code for engaging a module on client side:

```
if(FAILURE == zend_parse_parameters(ZEND_NUM_ARGS()
        TSRMLS_CC, "s", &mod_name, &mod_name_len)) {
         php_error_docref(NULL TSRMLS_CC, E_ERROR, "Invalid
         parameters");
         return;
}
 AXIS2_GET_THIS(obj);
AXIS2_GET_OBJ(client, obj, axis2_svc_client_t, intern);
AXIS2_SVC_CLIENT_ENGAGE_MODULE(client, AXIS2_GLOBAL(env),
         mod_name);
```

Engaging Addressing - PHP

- Engaging a module on server side does not need specific wrapping code
- Module could be engaged globally through configuration file axis2.xml

Engaging Addressing - AJAX

```
req.options
    ({ wsa_action:"http://ws.apache.org/axis2/c/samples/echoString" } );
req.engage ( "addressing", "vx" );
```

Engaging Addressing - AJAX

- Wrapping code for engaging a module:

```
AXIS2_SVC_CLIENT_ENGAGE_MODULE(svc_client, env,
        AXIS2_MODULE_ADDRESSING);

char* wsa_action = nsnull;

GetOptionByKey (SOAPHTTPREQUEST_OPT_WSA_ACTION,
        &wsa_action );

AXIS2_OPTIONS_SET_ACTION(options, env, wsa_action);
```

Leveraging Full WS Stack

- Axis2 is designed with extensibility in mind
  - Modules are key here
- With Axis2/C
  - MTOM/XOP and WS – Addressing built into the engine
  - WS – Reliable Messaging is available as a module
  - WS – Security is being built, UsernameToken already supported
- Embedding Axis2/C means you have all these for free
  - PHP Axis2 is the first PHP Web service extension to support MTOM and WS – Addressing
  - With Firefox XPI AJAX extension you can use WS – Addressing

Future Plans for Axis2/C

- WS – Policy (Neethi/C)
- WS – Security (Rampart/C)
- WS – Secure Conversation (Rahas/C)
- WS – Eventing (Sawan/C)
- WS – Transactions (Kandula/C)

Embedding Efforts

- Already underway
  - PHP Axis2 – Done with few TODOs
  - Firefox XPI - Done with few TODOs
  - Ruby extension
- More possibilities
  - Perl
  - Python
  - Any other
- Other communities/individuals are welcome to drive these
  - May be you could drive one of them

How to contribute?

- Join the mailing list
  - axis2-c-dev-subscribe@ws.apache.org
- Submit your ideas.
- Share your extension.
- Become a committer.

Links/Lists

- Axis2/C Web site:
  http://ws.apache.org/axis2/c/
- Axis2/C mailing lists:
  http://ws.apache.org/axis2/c/mail-lists.html
- PHP Axis2 in PECL:
  http://pecl.php.net/package/axis2
- Firefox XPI
  http://dist.wso2.net/products/tungsten/ajax/xpi/index
- Embedding Axis2/C
  http://www.wso2.net/articles/axis2/c/2006/10/10/ws-

Thank You.

And Questions please...

Presentation Materials

are available @

http://us.apachecon.com/html/archive.html