Web Services Resource Framework

Building WS-Resources with WSRF::Lite.

Mark Mc Keown
UK Grid Support Centre
The Deliverables...

- What are Web services?
- What was OGSI?
- What is WSRF?
- What are the other options?
- Break.
- Using Perl for Web Services.
- WSRF::Lite.
- Using WSRF::Lite.
What are Web Services?

- “A service is an entity that exchanges messages, usually with other services, with well-defined boundaries”

- “A Web service shares those characteristics, but adds the constraint that the messages exchanged between Web services are in fact SOAP messages carried over some suitable transport protocol” - Jim Webber.

- “A software system designed to support interoperable machine-to-machine interaction over a network” - OGSA

- “A thing that does something” – the pub last Friday.
SOAP

HTTP Headers

<Envelope>
  <Header>
    XML...
  </Header>
  <Body>
    XML...
  </Body>
</Envelope>
Simple Object Access Protocol?

or

Service Orientated Access Protocol?

“Originally the acronym stood for Simple Object Access Protocol, but that name is now considered to be outdated, so it no longer stands for anything”

OGSA Glossary
WSDL – Web Service Description Language

- Describes the Web service interface: the messages that can be exchanged, the operations, where to contact the service, what transport protocols can be used etc…

- A bit like a header file for a library – but holding a lot more information.

- Does not describe what the service does, how it does it or how you should use the service – ie does not provide semantic information about the service.
SOA – Service Orientated Architecture

- “This term is increasingly used to refer to an architectural style of building reliable distributed systems that deliver functionality as services, with the additional emphasis on loose coupling between interacting services.”

  OGSA Glossary

- But not the “Object Orientated Architecture” 😊
Publish, Find and Bind Triangle

Service Oriented Architecture

Discovery Agency

Service Requester

Service Provider

Find

Interact

Publish

Service Description
RPC Style

- RPC – similar to making a function call.

```c
int add( int value1, int value2 )

<body>
  <add>
    <value1 xsd:type="int">10</value1>
    <value2 xsd:type="int">5</value2>
  </add>
</body>
```
Rather than the service providing remote functions - we can think about exchanging messages with a service.

<MathsOperation>
  <Operation name="ADD">
    <Arg>5</Arg>
    <Arg>10</Arg>
  </Operation>
</MathsOperation>
<MathsOperation>
  <Operation name =“DIVIDE”>
    <Top>
      <Operation name=“ADD”>
        <Arg>5</Arg>
        <Arg>10</Arg>
        <Arg>15</Arg>
      </Operation>
    </Top>
    <Bottom>3</Bottom>
  </Operation>
</MathsOperation>
OGSI – Open Grid Service Infrastructure

- Web Services are stateless, but we need to manage state in a standard way!

- “Grid services are stateful Web Services.”

- Create a service to manage each new piece of state.

- Provided a standard set of mechanisms for managing the state of a Grid service.
Provided a mechanism for collecting a group of Grid services together – building blocks for registries.

Provided Notification mechanisms – services can notify clients of changes in state.

Support portType inheritance – all Grid services were derived from a base Grid service portType.
Counter Service

- Problem: Want to manage a set of Counters that can be accessed through a Web service interface.

- Clients should be able to create a new Counter for their own use.

- In OGSI each Counter would be modelled by a separate Grid service.

- To create a new Counter a client would make an OGSI createService call to a Counter Grid service Factory.

- The Factory would return the address (GSR/GSH) of the new Counter Grid service for the client use.
OGSI Counter

Client

Counter Grid Service

Counter Factory Service

Grid Service Container

add Destroy

GSH/GSR createService
OGSI Counter Service

POST  http://vermont.man.ac.uk/Counter/1233544564

<Envelope>
  <Body>
    <add>
      <value>10</value>
    </add>
  </Body>
</Envelope>
Garbage Collection

- If we create all these services without eventually cleaning them up we will leak resources.
- OGSI allowed the client to destroy the Grid Service.
- However there is no guarantee that the client will actually do Garbage Collection.
- OGSI Grid services also have lifetimes, so that after a set period they expire and release the resources they used.
Why OGSI didn’t succeed…

- Poor tooling support, OGSI introduced extensions to WSDL that the mainstream Web service community did not pick up.

- Too much stuff in one specification.

- **Too object orientated.**
WS-RF Web Service Resource Framework

- WS-RF effectively has replaced OGSI since January 2004.
- Addresses the problems with OGSI.
- Doesn’t use inheritance – instead we compose portTypes.
- Simply a re-factoring of OGSI – I wish 😞
- Instead of Grid services we have WS-Resources.
WS-Resource Counter

Client

createResource

add

Destroy

WS-Addressing

EPR

Web Service

Counter Resource

counterID=1

add

Counter Resource

counterID=2
“Implied” Resource Pattern

- No explicit createService operation, but still have the notion of a factory – effectively any Web service that will create a WS-Resource for you.

- The factory returns a WS-Addressing Endpoint Reference, the client uses this EPR to communicate with the WS-Resource.

- The EPR holds an identifier for the WS-Resource.

- The client puts the identifier for the WS-Resource in the SOAP Header.

- The client should NOT reason about the identifier.

- The Web service uses the identifier to decide which WS-Resource to use.
WS-Addressing

- Warning: WS-Addressing is not yet in a standards track 😞
- An EPR:

```xml
<EndpointReference>
  <Address>http://vermont.man.ac.uk/Counter</Address>
  <ReferenceProperties>
    <counterID>1</counterID>
  </ReferenceProperties>
</EndpointReference>
```
**SOAP**

<table>
<thead>
<tr>
<th>POST <a href="http://vermont.man.ac.uk/Counter">http://vermont.man.ac.uk/Counter</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Envelope&gt;</td>
</tr>
<tr>
<td>&lt;Header&gt;</td>
</tr>
<tr>
<td>.....</td>
</tr>
<tr>
<td>&lt;counterID&gt;1&lt;/counterID&gt;</td>
</tr>
<tr>
<td>&lt;/Header&gt;</td>
</tr>
<tr>
<td>&lt;Body&gt;</td>
</tr>
<tr>
<td>&lt;add&gt;&lt;value&gt;10&lt;/value&gt;&lt;/add&gt;</td>
</tr>
<tr>
<td>&lt;/Body&gt;</td>
</tr>
<tr>
<td>&lt;/Envelope&gt;</td>
</tr>
</tbody>
</table>
WS-ResourceProperty

- Provides a “projection” of the WS-Resource’s state.

- The ResourceProperties can be described in an XML document – the WSDL for the service should have a pointer to this document.

- `getResourceProperty` and `getMultipleResourceProperties` operations allows client to query the state of the WS-Resource

- `setResourceProperty` operation allows client to modify the state of the WS-Resource – supports Insert, Delete and Update.
<xs:schema .....>
    <xs:element name="foo" xsd:type="int" />*
    <xs:element name="bar" xsd:type="string" />?
    <xs:element name="count" xsd:type="int" />?
</xs:schema>
<setResourceProperty>
  <Insert>
    <foo>10</foo>
    <foo>15</foo>
  </Insert>
  <Update>
    <count>12</count>
  </Update>
  <Delete ResourceProperty="bar" /> 
</setResourceProperty>
Unlike in OGSI, lifetime management is optional in WS-RF.

Destroy and setTerminationTime operations allows the client to control the lifetime of the WS-Resource.

The lifetime of the WS-Resource is just another WS-ResourceProperty – can use getResourceProperty to find termination time.

However you **CANNOT** set the termination time through the setResourceProperty operation – must use setTerminationTime!!
A standard way to report errors:

```xml
<BaseFault>
  <Timestamp>…</Timestamp>
  <OriginatorReference>…</OriginatorReference>?
  <ErrorCode>…</ErrorCode>?
  <Description>…</Description>*
  <FaultCause>…</FaultCause>*
</BaseFault>
```
WS-ServiceGroup

- Mechanism to group a set of WS-Resources together – basic building block for registries.

- WS-Resources come and go, need to garbage collect stale entries in the ServiceGroup – but how?

- When we register a WS-Resource in a ServiceGroup a new WS-Resource is created by the ServiceGroup.

- The sole purpose of this new WS-Resource is to control the lifetime of the entry in the ServiceGroup – destroy this WS-Resource and the entry disappears.
Each ServiceGroupEntry has:
- An EPR to the WS-Resource that has been registered.
- An EPR to the WS-Resource that controls the entry.
- Content – the meta data the client wants to associate with the WS-Resource he has registered.

The ServiceGroup can define a set of rules about the content e.g. only Counter WS-Resources can be registered in this ServiceGroup.
What happens when a service provider wants to move a WS-Resource between hosts?

Need a mechanism for the client to find the new address of the WS-Resource.

Replaces the OGSI GSR/GSH mechanism.

NOT YET RELEASED 😞
A client can contact a WS-Resource that supports WS-Notification and ask to be notified whenever the state of the WS-Resource changes.

The client will have to listen for notifications. (firewalls?)

How do we garbage collect subscriptions? We don’t want a service sending us notifications forever…

Again we use a WS-Resource to control the subscription.

A new WS-Resource is created when a subscription request is made, when this WS-Resource is destroyed or expires the notifications stop.
Other Options


- “Explicit” Resource Pattern also known as plain vanilla Web Services.

- REST.
"Explicit" Resource Pattern.

- Simply put the identifier into the SOAP body rather than the header – this is what we do now!!

```xml
<Envelope>
  <Body>
    <add>
      <counterID>1</counterID>
      <value>10</value>
    </add>
  </Body>
</Envelope>
```
REST Representational State Transfer

- Originally described by Roy Fielding in his PhD thesis.

- The idea is to make use of the existing technology, URL, XML, HTTP etc. to build Web services.

- For example HTTP supports:
  - HEAD (get the Meta data about the resource)
  - POST (create a resource)
  - GET (get the state of the resource)
  - PUT (modify the state of the resource)
  - DELETE (destroy the resource)
RESTful Counter

- Client POSTs a XML message to a URL to create a new Counter resource – receives a URL to the new Counter resource.
- GET on the new URL returns value of Counter – the service returns XML or HTML or PDF - whatever the client requests.
- The return should also contain “xlinks” – allow the client to navigate to other useful information or services.
- POST or PUT a XML message to the URL to modify the Counter.
- Use DESTROY to destroy the Counter.
REST forces you to think very carefully about the service you want to provide and how to structure your data.

- Potentially allows you to load your data with semantic information.

- Always useful to think about how your service would be built in a RESTful way.

- Not very much tool support for building RESTful services.
Towards true Web Services?
Jim Webber
Break...
Perl – Are you mad?

- “If Perl is the solution, you’re solving the wrong problem.”
  Erik Naggum.

- “Perl as a language has less a design than a thousand special features flying in close formation.”

- “Perl: the first post-modern computer language.”
  Larry Wall.
SOAP

HTTP Headers

<Envelope>
  <Header>
    XML...
  </Header>
  <Body>
    XML...
  </Body>
</Envelope>
supports SOAP 1.1 and 1.2, XML-RPC, UDDI publishing and query.

- Transport Protocols: HTTP, HTTPS, Jabber, FTP, SMTP, POP3, MQSeries....

- Services can be hosted in Apache or using a standalone Perl daemon.

- “Don’t be mislead by the Lite suffix – this refers to the effort it takes to use the module, not its capabilities”

  Pavel Kulchenko
#!/usr/bin/perl

use SOAP::Transport::HTTP;

SOAP::Transport::HTTP::CGI
    ?>dispatch_to( ‘/home/mmk/modules’ )
    ?>handle;
package HelloWorld;

sub Hello {
    my ($self, $name) = @_;  
    return "Hello ". $name;
}

1;
#!/usr/bin/perl
use SOAP::Lite;

my $service = SOAP::Lite->service( "URLtoWSDL" );

my $ans = $service->Hello( "Mark" );

if ($ans->fault) { die $ans->faultstring }

print $ans->result."\n";
#!/usr/bin/perl
use SOAP::Lite;

$ENV{HTTPS_CA_DIR} = "/etc/grid-security/certificates/";
$ENV{HTTPS_CERT_FILE} = $ENV{HOME}."/.globus/usercert.pem";
$ENV{HTTPS_KEY_FILE} = $ENV{HOME}."/globus/userkey.pem";

my $service = SOAP::Lite? > service( "URLtoWSDL" );

my $ans = $service? > Hello( "Mark" );

if ($ans? > fault) { die $ans? > faultstring }
print $ans? > result. "\n";
#!/usr/bin/perl

use SOAP::Lite;

my $service = SOAP::Lite?->service( shift @ARGV );

my $function = shift @ARGV;

my $ans = $service?$function( @ARGV );

if ($ans?fault) { die $ans?faultstring } print $ans?->result."\n";
WSRF::Lite started out as OGSI::Lite which started out as an exercise to understand OGSI and the concepts behind Grid services.

The name is of course derived from SOAP::Lite.


WS-ServiceGroup, WS-BaseFault, WS-Notification still to do.
Implementing WS-RF

- Need to provide the WS-RF defined operations.

- Need to maintain the state between calls to the service –
  - Use a process to manage the state of the WS-Resource
  - Use a process to manage the state of a number of WS-Resources
  - Use a file or database to hold our state.
WSRF::Lite

- WS-RF services can be hosted using a stand alone script, using the WSRF::Lite Container or using Apache.

- Can use HTTPS for security – WS-Security is on the TODO list.

- Works with Java implementations (Globus, IBM, University of Indiana...), .NET implementations (University of Virginia, HP) and Python (Lawrence Berkley National Lab).
Process based WS-Resource

Client → Container (SOAP, HTTP)

Container → WS-Resource (SOAP, UNIX-Socket)

WS-Resource → Container → Client (SOAP, HTTP)
File based WS-Resource

Client -- SOAP -- Container

Host Machine

HTTP

File

Client -- SOAP -- Container

HTTP

File

ODBC

Database
Installing WSRF::Lite

- Requires a number of Perl modules installed: XML::DOM, LWP, SOAP::Lite etc.

- These can be installed using CPAN:
  - perl -MCOREAN -e shell
  - cpan> Install SOAP::Lite

- Important to use an up to date CPAN mirror.

- Modules can be installed in user space.
WSRF::Lite tarball

- Two Container scripts – one for http the other https
- A modules directory with some sample WS-Resource modules.
- README and TODO file.
- A directory with some simple WSRF client scripts and a README on how to use them.
- A directory with some test scripts for WSRF functionality.
- A directory, WSRF, which has the Lite.pm module.
Running Container…

- The environmental variable WSRF_MODULES needs to point to the directory where the WS-Resources modules are.

- The Container script forks on each new connection – there is a pre-fork Container script which offers improved performance.

- By default the Container script uses port 50000, but the script is easy to change – the port needs to be open in any firewall.

- The directories /tmp/wsr and /tmp/wsr/data need to be created – the UNIX sockets and file based WS-Resources use these directories.
The modules directory...

- **logs directory** – stdout/stderr from each WS-Resource is sent here, the name of each log file corresponds to the ID of the WS-Resource.

- **MultiSession** – modules in this directory use a single process to manage multiple WS-Resources

- **Session** – modules in here use either a file or database to manage their state – the module is loaded when a call is made to a WS-Resource.

- **WSRF** – one process per WS-Resource.
The process based Counter example.

- The factory Service that creates Counter WS-Resources is in $WSRF_MODULE/Session/CounterFactory.

- Since the factory does not need to maintain state between calls we can put it in the Session directory.
Package Counterfactory;
use ....

sub createCounterResource {
    my $envelope = pop @_; # get SOAP envelope
    my ($class, @params) = @_; # get input params

    my $newService = WSRF::Resource->new(...); # create service
    my $resourceID = $newService->ID(); # get the resource ID

    $newService->handle(@params); # start service – pass in
    params # to service’s init method

    my $wsa = WSRF::GSutil::createWSAddress(...); # create WSA EPR

    return WSRF::Header::header($envelope),
          SOAP::Data->value($wsa)->type('xml'); # return SOAP header # and WSA EPR}
Simple Counter

Package Counter;
use...
@ISA = qw(WSRF::WSRL);
# inherit core WSRF functionality

sub init {};
# called by factory

my $count = 0;
# this is our state

sub add {
  my ($class, $value ) = @_;
  $count = $count + $value;
  return $count;
}
# add operations

...
Using ResourceProperties

Package Counter;
use...

ResourceProperties{count} = 0;
# NotDeletable{count} = 1;
# NotModifiable{count} = 1;                  # declare count as RP
# controls if count can be deleted/
# modified by setResourceProperties

sub add {
    my $envelope = pop @_;  
    my ($class, $value) = @_;  

    ResourceProperties{count} = ResourceProperties{count} + $value;

    return WSRF::Header::header($envelope), ResourceProperties{count};  
    }
- WS-BaseFaults.
- WS-ServiceGroups.
- WS-Notification.
- WS-Security.
- More support for REST.
- WS-Context.
- Fix bugs, write documentation…
References

- “Modeling Stateful Resources with Web Services”
  - http://www.globus.org/wsr

- WS-RF Specifications.

- WS-GAF
  - http://www.neresc.ac.uk/ws-gaf/

- REST
  - http://webservices.xml.com/pub/a/ws/2002/02/06/rest.html
  - “Programming Web Services with Perl” by Ray and Kulchenko
References...

- **RPC v Document Style**

- **SOA v OOA**
  - [http://savas.parastatidis.name](http://savas.parastatidis.name)
  - [http://jim.webber.name/](http://jim.webber.name/)
  - Google 😊

- **SOAP::Lite**
  - [http://www.soaplite.com](http://www.soaplite.com)
  - [http://www.majordojo.com](http://www.majordojo.com)
  - “Programming Web Services with Perl” by Ray and Kulchenko
References...

- **WSRF::Lite**
  - [http://www.sve.man.ac.uk/Research/AtoZ/ILCT](http://www.sve.man.ac.uk/Research/AtoZ/ILCT)

- **OGSI**
  - [https://forge.gridforum.org/projects/ogsi-wg](https://forge.gridforum.org/projects/ogsi-wg)