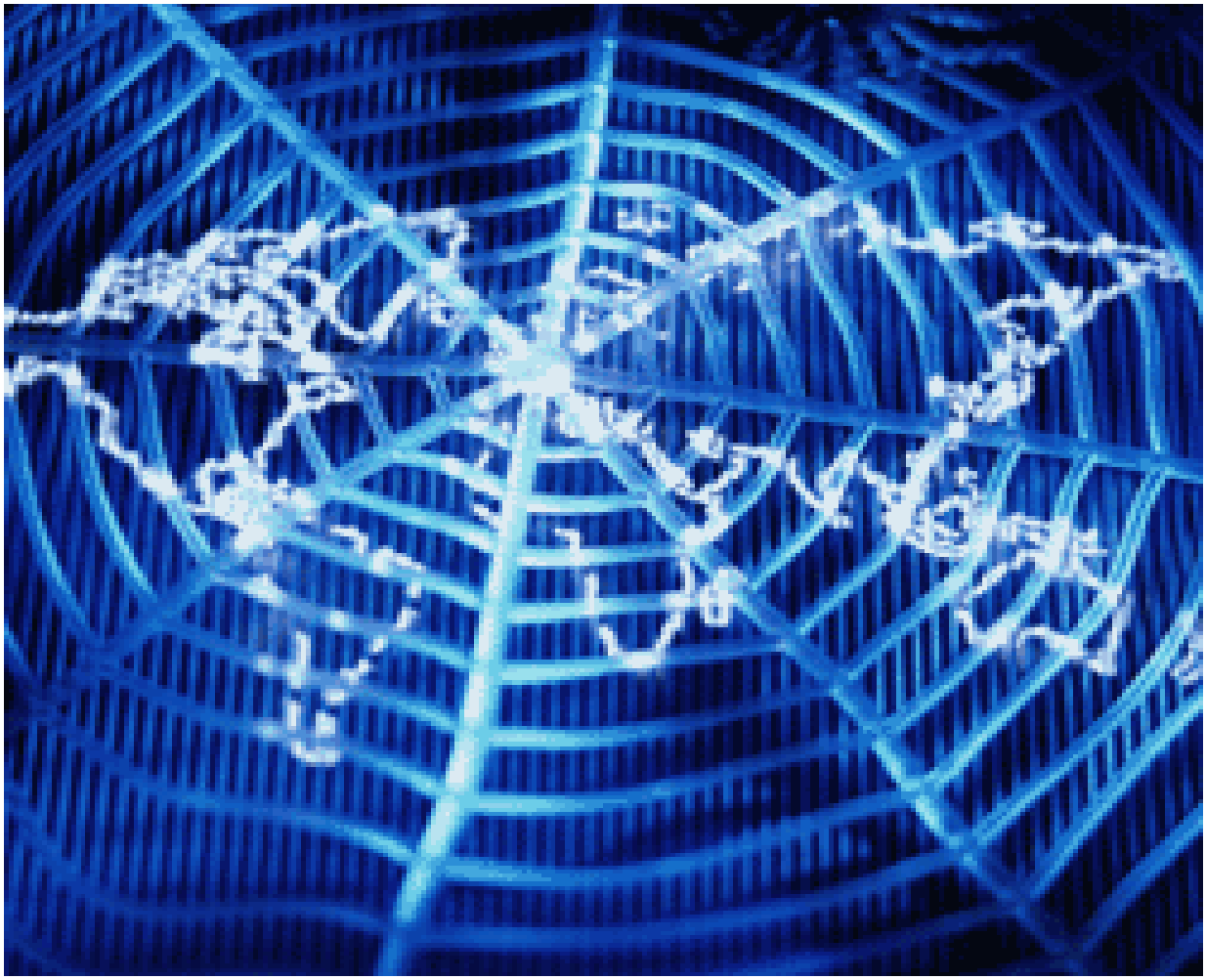
'Análisis de la Arquitectura de Globus Toolkit 4'

José Luis Vázquez Poletti
Grupo de Arquitectura de Sistemas Distribuidos y Seguridad
Facultad de Informática – Universidad Complutense de Madrid
jlvazquez@fdi.ucm.es
http://asds.dacya.ucm.es/jlvazquez/

08/10/04

Contenidos

- Introducción a los Web Services
- WSRF de Globus
- Migración Conceptual de GT 2.4 a GT 4
- Instalación y Configuración (Diario de bitácora)



*"**Morpheus :** I imagine that right now you're feeling a bit like Alice, tumbling down the rabbit hole."*

INTRODUCCIÓN A LOS WEB SERVICES

XML, el lenguaje de los Web Services

A pesar de su sencillez aparente, XML está transformando completamente la creación y el uso de software. El Web revolucionó la comunicación entre usuarios y aplicaciones. XML está revolucionando la comunicación entre aplicaciones o, de forma más general, la comunicación entre equipos, pues ofrece un formato de datos universal que permite adaptar o transformar fácilmente la información.

Como veremos a continución, XML es la base de los Servicios Web, así que antes vamos a hacernos una idea clara de qué aspecto tiene la información en lenguaje XML.

XML es la solución a un problema de comunicación entre programas de ordenador. La información generalmente queda fuertemente ligada al programa con que fue creada, y es así como se pierde mucho tiempo en pasar de RTF a PS a PDF a páginas HTML ó a otros formatos de definición de documentos.

Muchos Linuxeros, sobretodo Debianitas, saben de un nuevo standard llamado DocBook, que pretende dar solución a este problema. Precisamente, DocBook es un caso particular de XML. DocBook se aplica sobre documentos ... ¿pero si se trata de una imagen o un sonido? XML intenta ser un formato absolutamente genérico, con el que describir CUALQUIER tipo de fichero. Así, por ejemplo SVG 1.0, Scalable Vector Graphics, es un formato basado en XML para representar imágenes vectoriales. DSML 1.0, Directory Services Markup Language, otro formato basado en XML, específico para definir directorios. XHTML 1.0, Extensible HyperText Markup Language, uno se puede imaginar que es un formato basado en XML para representar páginas web.

Cada día aparecen nuevas propuestas para estadarizar formatos basados en XML. Ya existen muchas herramientas de programación preparadas para trabajar con formatos basados en XML. Es una tendencia y se está consolidando. Los programadores con mentalidad abierta, codifican sus ficheros de datos en "algo" basado en XML. Así si un día necesitan portar sus datos a otras aplicaciones, pueden usar herramientas ya hechas y probadas.

Muy pronto, si no es ya, que una aplicación trabaje con formatos basados en XML será una cualidad tan valorada como que sea GPL.

Queda claro que XML es ya un formato universal, que lo están adoptando multitud de programadores y que puede abarcar cualquier caso de intercambio de información, pero la pregunta que a muchos os habrá surgido es: ¿por qué no intercambiar los datos mediante ficheros de texto plano? Linux lo usa en casi todas las configuraciones de sus programas: httpd.conf, my.cnf, inetd.conf, etc, etc y todos sabemos lo bien que funciona y lo abierto que es... Efectivamente, pero ¿y si necesitamos transmitir información con estructura? Ahí es donde resulta interesante XML.

Imaginemos el caso de describir una página web con un fichero de texto... se puede hacer, de hecho se hace constantemente usando HTML, que puede grabarse como un fichero de texto plano. Sin embargo, HTML no podría ser lo que es, sin su lenguaje de marcas. Luego ya no es sólo un texto

plano, sino un texto y sus marcas. Las marcas sirven para estructurar la información: separar el título del contenido, por ejemplo.

XML es una generalización de este proceso. En XML se marca TODO. Cualquier información transmitida por un XML está perfectamente estructurada. Para ello se emplea también un lenguaje de marcas. Como XML es un formato universal, y no puede limitarse a saber describir sólo páginas web, por ejemplo, las marcas no son fijas, sino variables según el subformato (definido en los archivos llamados DTD).

Un ejemplo sencillo: una colección de 750 recetas de cocina. ¿Escribirlas todas en KOffice? ¡Qué peligro! ¿Qué pasaría si me dicen que es necesario tenerlas en HTML? ¿O imprimirlas en un libro usando programas específicos y cierto estilo de tipografía para los ingredientes? Entonces es conveniente poner mayor atención al elegir el formato a emplear. ¿Sólo texto? podría ser... si no incluimos fotos, ni índices, ni tablas, ni... etc, etc. Es evidente que una receta se divide en partes bien identificadas, así que usar sólo texto sería perder información sobre dónde está cada cosa en el documento.

Agent Smith: *"Never send a human to do a machine's job."*

XML nos dice que podemos estructurar la información en un árbol. Es decir imaginar a la receta como un componente, que a su vez esta formado de componentes, y así sucesivamente. Cada componente podría tener texto y/o más componentes. Una posible estructura sería imaginar que la receta tiene un componente llamado "necesitamos". No todo el texto estaría dentro de "necesitamos", solamente aquellas cosas que el cocinero de la receta necesitaría para llevarla a cabo. Dentro podríamos tener uno o más componentes llamados "ingrediente". Veamos como se ve esto (usando ya la sintáxis de XML).

```
<receta>
...
        <necesitamos>
                <ingrediente>2 cucharadas de azúcar</ingrediente>
                <ingrediente>3 manzanas</ingrediente>
        </necesitamos>
...
</receta>
```

¿Se adivina cuál es la sintaxis de XML? Es simplemente encerrar al texto que pertenece a un componente entre <componente> y </componente>. Bueno, en realidad ya se ve que estos componentes son "tags".

El XML se completa mediante una "hoja de estilo", que es una descripción de cómo debe verse una información en un determinado medio. A un mismo documento XML se le pueden aplicar distintas hojas de estilo según convenga. Por ejemplo usando una hoja de estilo por cada medio en la que se debe representar la información.

Existen actualmente dos lenguajes de hojas de estilo: CSS (Cascading Style Sheets: que ya está parcialmente implementado en los navegadores de WWW) y XSL (eXtended StyleSheet Language: El W3 Consortium creó este nuevo lenguaje de hojas de estilo)

XML sirve para que muchos programas interpreten bien cualquier tipo de dato. No sólo eso. XML sirve para que algunos programas hablen entre ellos sin intervención humana. ¿Para qué? Computación Distribuida, Interoperatividad, Monitorización,... son situaciones en las que resulta imprescindible este tipo de comunicación. XML es también la solución en estos casos. Como veremos ya, muy pronto, los Servicios Web son un caso particular de "Computación Distribuida" y XML es su lenguaje base.

Los servicios Web XML permiten que las aplicaciones compartan información y que además invoquen funciones de otras aplicaciones independientemente de cómo se hayan creado las aplicaciones, cuál sea el sistema operativo o la plataforma en que se ejecutan y cuáles los dispositivos utilizados para obtener acceso a ellas. Aunque los servicios Web XML son independientes entre sí, pueden vincularse y formar un grupo de colaboración para realizar una tarea determinada.

Los Servicios Web XML

Bien, esta puede ser una primera definición de Servicio Web. Pero, quizá, lo que ahora más interese es saber ¿para qué sirve un Servicio Web? La respuesta puede ser otra pregunta: ¿Para que sirve en programación una rutina? Todos sabemos que una rutina es como una caja negra, que encierra cierto proceso o algoritmo, y que cumple una función clara. Muchas rutinas y un guión central componen un programa en lo que se llama "programación estructurada". Un Servicio Web viene a ser una rutina en Internet.

Pero, ¿por qué se llama "Servicio Web" y no "Rutina en Internet"? Los protocolos que soportan los servicios web se comunican normalmente por el puerto 80, y basándose en HTTP, métodos GET y PUT. Esto hace que podamos acceder a ellos al igual que lo hacemos en una página web. La diferencia entre una página web y un Servicio Web, es que la página la visita cualquier individuo interesado, mientras que el servicio sólo lo visitan programas que lo requieren.

De modo, que el conjunto de Servicios Web en Internet es una World Wide Web paralela, de carácter no humano, sino cibernético. Vamos, que los ordenadores ya hablan sólos a través de Internet.

Agent Smith: *"Like the dinosaur... Look out that window. You had your time. The future is our world, Morpheus. The future is our time."*

Los Protocolos de Web Services

Hay un convenio generalizado que nos da a entender que los Servicios Web se invocan en Internet por medio de protocolos estándar basados en XML. Hoy en día hay dos grandes tendencias: XML-RPC y SOAP. A la hora de programar un servicio web, hay que decidir qué protocolo usar, porque un protocolo es incompatible con el otro. De modo que si programamos nuestro servicio web con XML-RPC, no podremos invocarlo desde un lenguaje de programación que trabaje con SOAP, como por ejemplo .Net de Microsoft.

Tanto SOAP (Protocolo de acceso a objetos simple, Simple Object Access Protocol) como XML-RPC son lenguajes de mensajería basada en XML, estandarizados por el consorcio W3C, que especifican todas las reglas necesarias para ubicar servicios Web XML, integrarlos en aplicaciones y establecer la comunicación entre ellos.

Brevemente, la diferencia entre SOAP y XML-RPC es su complejidad. XML-RPC está diseñado para ser sencillo. SOAP por el contrario está creado con idea de dar un soporte completo y minucioso de todo tipo de servicios web. La curva de aprendizaje de XML-RPC es muy suave, por lo que un programador novato en este campo, puede conseguir resultados satisfactorios con poco esfuerzo. Con SOAP no pasa esto, pero a cambio, se dispone de más potencia. Por ejemplo, con XML-RPC no se puede elegir el conjunto de caracteres a utilizar en los Servicios Web. En SOAP se puede elegir entre US-ASCII, UTF-8 y UTF-16.

Algo que puede decantarnos por uno u otro, sin necesidad de leer más, es saber que XML-RPC es más "Unix friendly" que SOAP.

Por el contrario, en favor de SOAP podemos añadir que el famoso Servicio Web de Google está basado enteramente en SOAP, y que el proyecto Apache, dispone ya de un módulo de trabajo con SOAP, que se puede usar como una librería cliente para invocar servicios SOAP disponibles en cualquier lugar o como una herramienta del lado del servidor para implementar servicios SOAP accesibles.

WSDL: El “Constructor” de los Servicios Web

Introducción

WSDL significa “Lenguaje de Definición de Web Services” y no es más que un documento escrito en XML (esto nos suena), en el que se describen las operaciones (o métodos) que dicho servicio ofrece.

Un documento WSDL usa los siguientes elementos:

- **<portType>** Las operaciones realizadas por el Web Service. Es el más importante y podría ser comparado a una librería de funciones (o un módulo, o una clase) en un lenguaje de programación tradicional.
- **<message>** Los mensajes usados por el Web Service y pueden consistir en una o más partes. Estas partes podrían ser comparadas como parámetros de una llamada a una función en un lenguaje de programación tradicional.
- **<types>** Los tipos de datos usados por el Web Service.
- **<binding>** Los protocolos de comunicación usados por el Web Service, definiendo sus detalles, así como el formato de los mensajes.

Aquí podemos ver un ejemplo simplificado de una parte de un documento WSDL:

```
<message name="getTermRequest">
```

```
  <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
  </operation>
</portType>
```

En el ejemplo, **porType** define "glossaryTerms" como el nombre de un **port**, y "getTerm" como el nombre de una **operation**.
La operación "getTerm" tiene un **input message** llamado "getTermRequest" y un **output message** llamado "getTermResponse".

Los **message** definen las **parts** de cada mensaje y los **data types** asociados.

Comparado con la programación tradicional, "glossariyTerms" es una librería de funciones, "getTerm" es una función con "getTermRequest" como parámetro de entrada y "getTermResponse", como parámetro de retorno.

Ports

El **port** define el punto de conexión a un Web Service. A continuación podemos ver los cuatro tipos de operaciones permitidos:

- **One-way:** La operación puede RECIBIR un mensaje pero no devolverá una respuesta.
- **Request-response:** La operación puede RECIBIR un mensaje y además, devolverá una respuesta.
- **Solicit-response:** La operación puede ENVIAR una petición y esperará una respuesta.
- **Notification:** La operación puede ENVIAR un mensaje pero no esperará una respuesta.

El tipo de operación más extendida es la **Request-response**.

A continuación veremos un ejemplo de operación del tipo **One-Way**:

```
<message name="newTermValues">
  <part name="term" type="xs:string"/>
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="setTerm">
    <input name="newTerm" message="newTermValues"/>
  </operation>
</portType >
```

El **port** "glossaryTerms" define una operación **one-way** llamada "setTerm". Esta operación permite la entrada con nuevos términos de glosario usando un mensaje "newTermValues" con "term" y "value" como parámetros. Además, no hay definida ninguna salida para esta operación.

En este otro ejemplo, veremos la operación **Request-Response**:

```
<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
 <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
 </operation>
</portType>
```

El **port** "glossaryterms" define una operación **request-response** llamada "getTerm", que requiere un mensaje de enterada llamado "getTermRequest" con un parámetro llamado "term", y devolverá un mensaje de salida llamado "getTermResponse" con un parámetro llamado "value".

Uniendo (Binding) a SOAP

Lo mejor para entender el **binding** empleando **request-response**, es ver un ejemplo:

```
<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
 <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
 </operation>
</portType>

<binding type="glossaryTerms" name="b1">
<soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http" />
 <operation>
   <soap:operation
    soapAction="http://example.com/getTerm"/>
   <input>
     <soap:body use="literal"/>
   </input>
   <output>
     <soap:body use="literal"/>
   </output>
```

```
  </operation>
</binding>
```

El elemento **binding** tiene dos atributos: El nombre y el tipo.

El nombre (se puede usar el que más se quiera) define el nombre del **binding**, así como tipos de atributos para el **port** del **binding**. En este caso, sería el **port** "glossaryTerms".

El **soap:binding** tiene dos atributos: El estilo y el transporte.

El estilo puede ser "rpc" o "document". En este caso usamos el segundo. El transporte define el protocolo SOAP usado, que en este caso será el HTTP.

El **operation** define cada **operation** ofrecida por el **port**. Para cada **operation**, se tiene que definir una acción SOAP, así como la entrada y salida son codificadas. En este caso hemos empleado "literal".

WSDL y UDDI

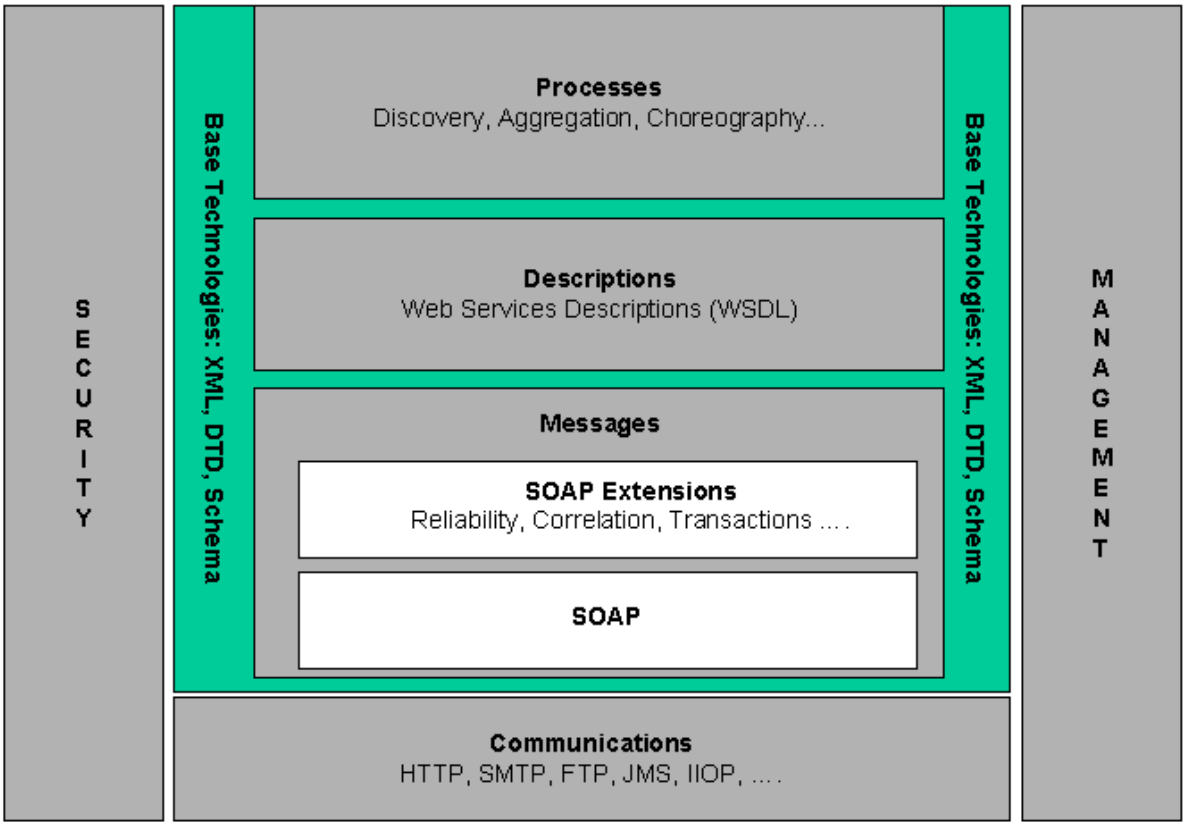
Responde a Universal Description, Discovery and Integration, y como primera aproximación, podemos decir que es un servicio de directorio donde los negocios se pueden registrar y buscar Web Services.

UDDI almacena información sobre Web Services en un directorio, mostrando las interfaces de aquellos que han sido definidas en WSDL. Además, se comunica usando SOAP.

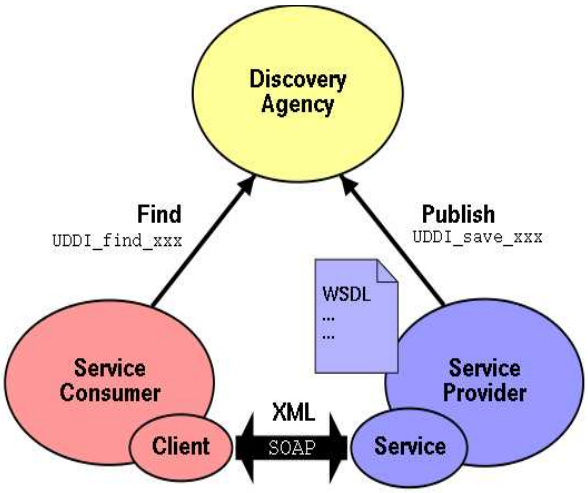
Juntando a todos en la foto

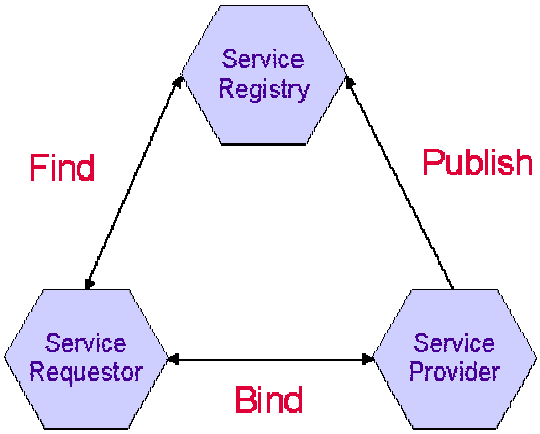
Ahora que conocemos todos los elementos, sería bueno alejarnos un poco y así obtener una perspectiva donde enlazarlos a todos, siguiendo la RFC "una imagen vale más que 1K palabras".

Para empezar, podemos darnos cuenta de donde situar cada concepto usando las capas como filtro:



Y ahora, con dos pseudo-diagramas de interacción, en los que se detallan diversos aspectos de la comunicación y los elementos que intervienen:





Un caso práctico: La API de Google con Perl

Morpheus: *"Neo, sooner or later you're going to realize just as I did that there's a difference between knowing the path and walking the path."*

Instalamos el módulo de SOAP

Una de las grandes maravillas que acompañan a nuestro intérprete de Perl por defecto instalado en cualquier Unix es el CPAN (Comprehensive Perl Archive Network), que al más puro estilo "apt-get" de Debian, nos ayudará en nuestra tarea para incorporar funcionalidades SOAP a nuestros programas, instalando las dependencias necesarias de existir.

Como root, ejecutaremos la siguiente instrucción en la línea de comandos:

perl -MCPAN -e shell

Si es la primera vez que lo hacemos, CPAN nos preguntará si queremos hacer la configuración manualmente. Digamos que no, ya que por norma, la autoconfiguración suele hacerse de forma exitosa.
Acto seguido, le diremos a CPAN que módulo queremos incorporar:

install SOAP::Lite

A partir de ese momento, CPAN se conectará a Internet y se descargará los paquetes necesarios. Ante las preguntas que nos puedan ir surgiendo (pruebas, instalación de dependencias), es recomendable seleccionar la opción por defecto, dejemos que las máquinas decidan por ellas solas.

Una vez instalado, saldremos de la consola CPAN:

quit

Nuestro primer programa en Perl con la API de Google

Google facilita una API que permite incorporar la funcionalidad de su motor de búsqueda a nuestros programas, y esto se realiza mediante SOAP. Para ello, tendríamos que registrarnos a fin de obtener una clave de usuario que será enviada con la petición en nuestro programa. Este paso es innecesario para nosotros, ya que dispondremos ya de una clave obtenida por otros medios.

En este ejemplo, crearemos un pequeño cliente SOAP en Perl que nos devuelva la primera coincidencia (al más puro estilo “Voy a tener suerte” de Google).

Creamos un fichero de texto que llamaremos *google.pl* donde escribiremos el siguiente código:

```
#!/usr/bin/perl -w
use SOAP::Lite;

print "\nBuscando en Google \"$ARGV[0]\"...\n\n";

my $query = "$ARGV[0]";
my $googleSearch = SOAP::Lite -> service("http://api.google.com/GoogleSearch.wsdl");
my $result = $googleSearch -> doGoogleSearch(
   "iwnUXUtHj3bteg5FWfBJDwui3SPeB+iy",     # Clave de acceso al API de Google
   $query,                  # Palabras clave de búsqueda
   0,                       # Indice del primer resultado mostrado
   10,                      # Numero de resultados obtenidos
   "false",                 # Filtro
   '',                      # Restricción
   "false",                 # Búsqueda Segura
   '',                      # lr
   'latin1',                # ie
   'latin1'                 # oe
);


print $result->{resultElements}->[0]->{title};
print "\n";
print $result->{resultElements}->[0]->{URL};
print "\n";

print "\n¡Busqueda finalizada!\n";
```

No olvidemos poner los permisos necesarios para que se pueda ejecutar la aplicación:

chmod +x google.pl
¡Muy bien! Ahora no tenemos más que ejecutarla poniéndole como argumento la cadena a buscar. Si deseamos poner más de una palabra, las uniremos con el símbolo '+'. Por ejemplo:

[jlvazquez@auriga Seminario081004]$ ***./google.pl gridway+dacya***

Buscando en Google "gridway+dacya"...

Distributed Systems Architecture & Security
http://asds.dacya.ucm.es/GridWay/index.php

¡Busqueda finalizada!

La página encontrada nos suena, ¿no?

Nuestro segundo cliente SOAP en Perl: Un diccionario inglés-castellano de sobremesa

Ahora usaremos la API de BabelFish. Crearemos otro fichero, esta vez de nombre *babelfish.pl* y escribiremos en él el siguiente código:

```
#!/usr/bin/perl -w
use SOAP::Lite;

print ">> $ARGV[0] en castellano es: ";
print SOAP::Lite
    ->proxy('http://services.xmethods.net/perl/soaplite.cgi')
    ->uri('urn:xmethodsBabelFish')
    ->BabelFish("en_es","$ARGV[0]")
    ->result;
print "\n";
```

Después de darle los permisos de ejecución convenientes, procedemos a hacer una prueba:

```
[jlvazquez@auriga Seminario081004]$ ./babelfish.pl Grid
>> Grid en castellano es: rejilla
```

A partir de ahora, esas palabras tan escurridizas que encontramos en algunos artículos no lo serán tanto.

En SOAP también hay servidores

Los ejemplos anteriores nos han servido para aprender a hacer clientes que se conectasen a un Web Service en SOAP.

Para hacer un servicio SOAP, necesitaremos tener un servidor web funcionando. En este caso, tratándose de Perl, necesitaremos que el soporte para CGIs esté activado, y esto siempre está por defecto (todo el mundo, hasta Bill Gates con su Hotmail, adora los CGIs).

Supongamos que estamos en Estados Unidos y el PC que nos dejan tiene una pequeña aplicación que nos dice la temperatura en ¡Grados Farenheit! Como nosotros venimos del sistema métrico decimal, quizás se nos haga difícil adaptarnos a esa escala sobre todo si quizás estemos pasando unos días. Bien, entonces destripando un poco el sistema, observamos que podemos extraer el valor numérico de dicha temperatura y quizás pudiéramos usar un Web Service en nuestro favor.

Además, se nos ocurre que, ante la inminente visita de uno de los representantes estadounidenses de Globus, no estaría mal que estuviera informado sobre la temperatura que hace en España unos días

antes de visitarnos.

Entonces procedemos a crear un fichero llamado *temper.cgi* en el raíz que contendrá el siguiente código:

```
#!/usr/bin/perl -w

 use SOAP::Transport::HTTP;

 SOAP::Transport::HTTP::CGI
   -> dispatch_to('Temperatures')
   -> handle;

 package Temperatures;

 sub f2c {
     my ($class, $f) = @_;
     return 5 / 9 * ($f - 32);
 }

 sub c2f {
     my ($class, $c) = @_;
     return 32 + $c * 9 / 5;
 }
```

No olvidemos de darle los permisos de ejecución pertinentes.

Y ahora le toca el turno al cliente. Así que abrimos un fichero de texto con el nombre *temperatura.pl* e introducimos el siguiente código:

```
#!/usr/bin/perl -w

 use SOAP::Lite;

 my $operacion = $ARGV[0];
 my $valor = $ARGV[1];

 if ($operacion eq "f2c")
 {
                 print SOAP::Lite
                 -> uri('http://www.soaplite.com/Temperatures')
                 -> proxy('http://services.soaplite.com/temper.cgi')
                 -> f2c($valor)
                 -> result;
 }
 elsif ($operacion eq "c2f")
 {
                 print SOAP::Lite
                 -> uri('http://www.soaplite.com/Temperatures')
                 -> proxy('http://services.soaplite.com/temper.cgi')
                 -> c2f($valor)
                 -> result;
 }
 else
 {
                 print "No humans allowed... only machines here!\n";
 }

 print "\n";
```

NOTA: El servidor usado es uno donde está subido ya en una máquina ejemplo. Para hacerlo con una propia no hay más que cambiar las URLs.

Muy bien, llegó la hora de probar nuestro cliente (una vez añadidos los permisos de ejecución adecuados). En el caso de querer transformar de grados Farenheit a Celsius, no tenemos más que escribir lo siguiente:

[jlvazquez@auriga Seminario081004]$ ***./temperatura.pl f2c 911***
488.333333333333

O bien, si encarnamos al representante estadounidense de Globus:

[jlvazquez@auriga Seminario081004]$ ***./temperatura.pl c2f 20***
68

Como algo opcional, en el cliente hemos puesto una medida de protección para intentar evitar que una tercera persona se dedique a enviar peticiones sin sentido a nuestra máquina (¡hey! No se trata ya de la máquina de Google o de Babelfish). De tal manera que, si el tipo de conversión pedido no entra dentro de lo esperado, el cliente devolverá un mensaje de error evitando lanzar la petición al servidor:

[jlvazquez@auriga Seminario081004]$ ***./temperatura.pl ImAWannabeHacker***
No humans allowed... only machines here!

WSRF DE GLOBUS

Retomando el concepto

Los Web Services dan a los usuarios la habilidad para acceder y manipular estados, valores persistentes, y finalmente esto resulta en una interacción entre Web Services.

WSRF no es más que un proyecto de estándar adoptado por Globus para definir convenciones sobre el manejo de esos estados, de tal manera que se pueda conferir una mejora en el descubrimiento, inspección e interacción con los recursos asociados a estados de forma estándar e interoperativa.

Un poco de historia

Los primeros trabajos al respecto fueron realizados por Globus e IBM, resultando en una arquitectura inicial, así como una especificación. En este ámbito, colaboraron HP, SAP (creadora del famoso Framework de Gestión), Akamai (proveedora de servicios de equilibrio de carga web usando servidores Linux y, curiosamente, Microsoft es uno de sus clientes), TIBCO y Sonic. El resultado fue publicado en Enero del 2004.

WSRF está inspirado en el trabajo del Global Grid Forum, en especial en el 'Open Grid Services Infraestructure (OGSI) Working Group'. De hecho, puede considerarse como una refactorización de los conceptos e interfaces desarrollados en la especificación OGSI V1.0, uniéndole los conceptos de Web Services (un ejemplo sería el WS-Addressing).

Definición de un WS-Resource

Podemos definir un WS-Resource como la unión de un Web Service y un recurso asociado a estados que es:

- Expresado como una asociación entre un documento XML de tipo definido y un Web Service.
- Accedido de acuerdo con el patrón de recurso implícito.

WS-Resource Framework permite definir, crear, acceder a, monitorizar y destruir WS-Resources, empleando mecanismos convencionales de los Web Services. Por contra, no se requiere que el componente Web Service del WS-Resource tenga que ser implantado como un procesador de mensajes asociado a estados.

WSRF está dividido en 5 especificaciones que son las que se verán a continuación en más detalle.

WS-ResourceLifetime

Son los mecanismos para la destrucción de los WS-Resources, incluyendo mecanismos que permiten asignar un momento para dicha destrucción.

El patrón WS-Resource Factory

Se basa en el patrón de diseño Factory estándar que tiene las siguientes ventajas para los objetos de este tipo:

- Separación de la responsabilidad de la creación compleja en objetos de apoyo (helper) cohesivos. Según las características del objeto que se quiere instanciar, se hace una llamada a la clase helper asociada.
- Ocultación de la lógica de creación potencialmente compleja.
- Posibilidad de introducir estrategias para mejorar el rendimiento de la gestión de memoria, como objetos caché y reciclaje.

Este patrón de diseño pretende resolver el problema de quién es el responsable de la creación de objetos cuando existen consideraciones especiales, dando como solución la creación de un objeto de Fabricación Pura que maneje la creación.

*"**Agent Smith:** The great Morpheus. We meet at last.*
***Morpheus:** And you are?*
***Agent Smith:** A Smith. Agent Smith.*

Morpheus: *You all look the same to me."*

Volviendo al WSRF, se define un WS-Resource Factory como cualquier Web Service capaz de crear uno o más WS-Resources.

Identidad del WS-Resource

El componente del recurso asociado a estado de un WS-Resource se identifica. Esto es realizado por el componente Web Service, que incluye un identificador del recurso en las propiedades de la denominada referencia del punto final de WS-Addressing. Así, este punto final pasa a denominarse un 'WS-Resource Calificado', que puede hacerse disponible a otras entidades in un sistema distribuido.

El concepto de 'WS-Resource Calificado' es completamente opaco, ya que no está en su mano el examen o la interpretación de los identificadores de un WS-Resource.

Destrucción del WS-Resource

Cualquiera que pida un WS-Resource estará solamente interesado en su uso durante un periodo limitado. A continuación veremos los dos tipos de destrucción del WS-Resource ofrecidos por el WSRF.

Inmediata: Cualquier receptor de servicios que quiera destruir un WS-Resource inmediatamente, no tiene más que utlizar el punto final asociado al 'WS-Resource Calificado' y mandarle una petición de destrucción. El resultado no se hace esperar, y el WS-Resource termina definitivamente.

Programada: Un WS-Resource Factory puede dar la habilidad de negociar la terminación programada de un WS-Resource ya en tiempo de creación. En añadidura, el receptor del servicio puede usar el punto final asociado al 'WS-Resource Calificado' para cambiar el tiempo para el cual estaba programada la destrucción, tanto para aumentarlo como disminuirlo.

WS-Resource Properties

Se trata de la definición de un WS-Resource, así como mecanismos para obtener, cambiar y borrar propiedades de un WS-Resource en particular.

La especificación de las propiedades de un WS-Resource define el tipo y el valor de aquellos componentes del estado de un WS-Resource que pueden verse y modificarse por los receptores del servicio, a través del interfaz del Web Service. Éstas son las ideas principales:

- El WS-Resource tiene un 'Documento de Propiededades' definido usando el esquema XML. Los receptores del servicio pueden determinar el tipo de un WS-Resource la definición 'portType usando WSDL' de forma estándar (El portType no es más que la definición de un conjunto de operaciones estándar, o lo que es lo mismo, qué funciones realiza un servicio).
- Los receptores del servicio pueden usar intercambio de mensajes de los Web Services para leer, modificar y pedir el documento XML que representa el estado del WS-Resource.

El término 'Propiedad de un Recurso' se emplea para designar un componente individual del estado de un WS-Resource, y tiene su propio documento XML para su descripción.

El Documento de Propiedades de un WS-Resource

Concretamente, la información se expresa empleando una 'XML global element declaration' (GED).

Supongamos un estado C que afecta a 3 componentes de un recurso que llamaremos p1, p2 y p3. Entonces, el documento asociado a las propiedades del recurso, que llamaremos "EjemploPropiedadesRecurso", podría ser:

```
<xs:schema
targetNamespace="http://example.com/EjemploPropiedadesRecurso"
xmlns:tns="http://example.com/EjemploPropiedadesRecurso"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
…
... >
                <xs:element name="p1" type= … />
                <xs:element name="p2" type= … />
                <xs:element name="p3" type= … />

                <xs:element name="EjemploPropiedadesRecurso">
                                <xs:complexType>
                                <xs:sequence>
                                                <xs:element ref="tns:p1" />
                                                <xs:element ref="tns:p2" />
                                                <xs:element ref="tns:p3" />
                                </xs:sequence>
                                </xs:complexType>
                </xs:element>
…
</xs:schema>
```

Los receptores del servicio puede descargar y examinar esta definición XML, que representa el tipo del recurso asociado a estados C.

El receptor sabe que el GED llamado "EjemploPropiedadesRecurso" define el documento de propiedades del recurso del WS-Resource por vía de la declaración del portType en el Web Service.

```
<wsdl:definitions
targetNamespace="http://example.com/EjemploPropiedadesRecurso"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:wsrp=
"http://www.ibm.com/xmlns/stdwip/web-services/ws-resourceProperties"
xmlns:tns="http://example.com/EjemploPropiedadesRecurso"
...>
...
                <wsdl:types>
                                <xs:schema>
                                                <xs:import
                                                namespace="http://example.com/EjemploPropiedadesRecurso"
                                                schemaLocation="..."/>
                                </xs:schema>
                </wsdl:types>
...
```

```
        <wsdl:portType name="NombreDeAlgunPortType"
        wsrp:ResourceProperties="tns:EjemploPropiedadesRecurso" >
                <operation name="...
        ...
        </wsdl:portType>
...
</wsdl:definitions>
```

Composición de Propiedades del WS-Resource

Tener solamente una interfaz de Web Service no resulta productivo. Es por ello que en el estándar WSDL 1.1, el diseñador puede añadir uno sencillamente mediante un copiar-y-pegar de las operaciones definidas en los portTypes ya existentes, de tal manera que se obtendría una composición. Dicha composición también puede afectar a las propiedades del WS-Resource.

En este ejemplo, vemos la modalidad de agregación empleando el atributo xs:ref.

```
<xs:element name="EjemploPropiedadesRecurso">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="tns:p1" />
                        <xs:element ref="tns:p2" />
                        <xs:element ref="tns:p3" />

                        <xs:element ref="xxxx:AlgunaPropiedadRecursoAdicional"
                        xmlns:xxxx= ... />

                </xs:sequence>
        </xs:complexType>
</xs:element>
```

Accediendo a los valores de las Propiedades del WS-Resource

El estado de un WS-Resource puede leerse y/o modificarse a través del documento de Propiedades, empleando mensajes Web Service. Este intercambio de mensajes tiene que estar incluido como operaciones WSDL en cualquier portType que use wsrp:ResourceProperties.

La función más básica consiste en obtener el valor de una única propiedad de un recurso empleando un intercambio de mensajes sencillo (petición/respuesta). Más complejamente, podemos encontrar una variante que nos permite obtener el valor de múltiples propiedades a la vez.

Estas operaciones están recogidas en la operación 'get' de WS-ResourceProperties.

```
<wsrp:GetMultipleResourceProperties>
        <wsrp:ResourceProperty> Nombre <wsrp:ResourceProperty>+
</wsrp:GetMultipleResourceProperties>
```

La respuesta a este mensaje es una secuencia de elementos XML correspondientes a los valores de las propiedades WS-Resources identificadas por los nombres especificados en el mensaje de petición.

Otro ejemplo nos permite ver la petición del valor de la propiedad p1 de un recurso asociado a

estados C.

```
<soap:Envelope>
        <soap:Header>
                <tns:resourceID> C </tns:resourceID>
        </soap:Header>
        <soap:Body>
                <wsrp:GetMultipleResourceProperty>
                <wsrp:ResourceProperty>tns:p1</wsrp:ResourceProperty>
                </wsrp:GetMultipleResourceProperty>
        </soap:Body>
</soap:Envelope>
```

La referencia al punto final usada para designar el objetivo de esta petición contiene un identificador en el componente ReferenceProperties que identifica el recurso C. En este caso, observamos que la información del identificador están en la cabecera SOAP, cumpliendo la codificación estándar.

Si seguimos con el ejemplo, veremos que el WS-Resource respondería con los valores de p1.

```
<soap:Envelope>
        <soap:Body>
                <wsrp:GetMultipleResourcePropertyResponse>
                        <p1>xyz</p1>
                </wsrp:GetMultipleResourcePropertyResponse>
        </soap:Body>
</soap:Envelope>
```

Por otro lado, también existe una operación 'set' que permite insertar, actualizar y borrar valores de las propiedades del WS-Resource. Aquí podemos ver una psuedo-sintáxis para el 'set':

```
<wsrp:SetResourceProperties>
        (
                <wsrp:Insert >
                        {any}*
                </wsrp:Insert> |

                <wsrp:Update >
                        {any}*
                </wsrp:Update> |

                <wsrp:Delete ResourceProperty="Nombre" />
        )+
</wsrp:SetResourceProperties>
```

Hay otra operación que permite al receptor de servicios para hacer un búsqueda. Un uso de esto bien sería la búsqueda de recursos, empleando búsquedas masivas, y basándose en los valores del estado de los WS-Resources.

Y finalmente, existe una última operación, que no menos interesante, a través de la cual, un receptor puede recibir notificaciones en caso de que algún valor de las propiedades del WS-**Resource** cambiara.

WS-RenewableReferences

Define una ampliación del WS-Addressing empleando políticas. Entre otras funciones, define mecanismos que pueden usar para renovar la referencia de un punto final que se ha vuelto inválido, bastante útil en caso de que dicho punto final hace referencia a un WS-Resource.

Una referencia a un punto final de WS-Addressing puede contener además información sobre políticas relativas a las interacciones con el servicio. Está claro que la información recibida por el cliente acerca de una política es susceptible de cambio, con lo que sería muy importante que la referencia al punto final se pudiera renovar, y además, debería existir una información que guiara a la hora de obtener una nueva referencia en caso de invalidez de la actual.

WS-ServiceGroup

Se trata de un interfaz para colecciones heterogéneas de Web Services. Los grupos creados por esta especificación se pueden definir como una colección de miembros que igualan el valor de ciertas variables, expresadas en el contexto de propiedades de recurso.

Los interfaces definidos por WS-ServiceGroup deberían poder componerse con otros interfaces de Web Services.

WS-BaseFaults

Es una tipicación XML para ser usada cuando aparecen fallos en en el intercambio de mensajes asociados a los Web Services.

WS-Notification

WS-Notification es una familia de especificaciones separada. Define un sistema Web Service para publicar y suscribirse a las notificaciones relacionadas con las interacciones en el Framework. Estas operaciones son consideradas de tipo general y asociadas a un tema, de tal manera que el cliente pueda tener un abanico mayor de funcionalidades. Además, se ofrece una perspectiva de bloques para representar y estructurar notificaciones.

Existen 3 subespecificaciones:

WS-BaseNotification: Describe los roles básicos, conceptos y patrones requeridos para permitir a un suscriptor registrarse.

WS-Topics: Presenta una descripción XML de los temas y metadatos asociados a las notificaciones.

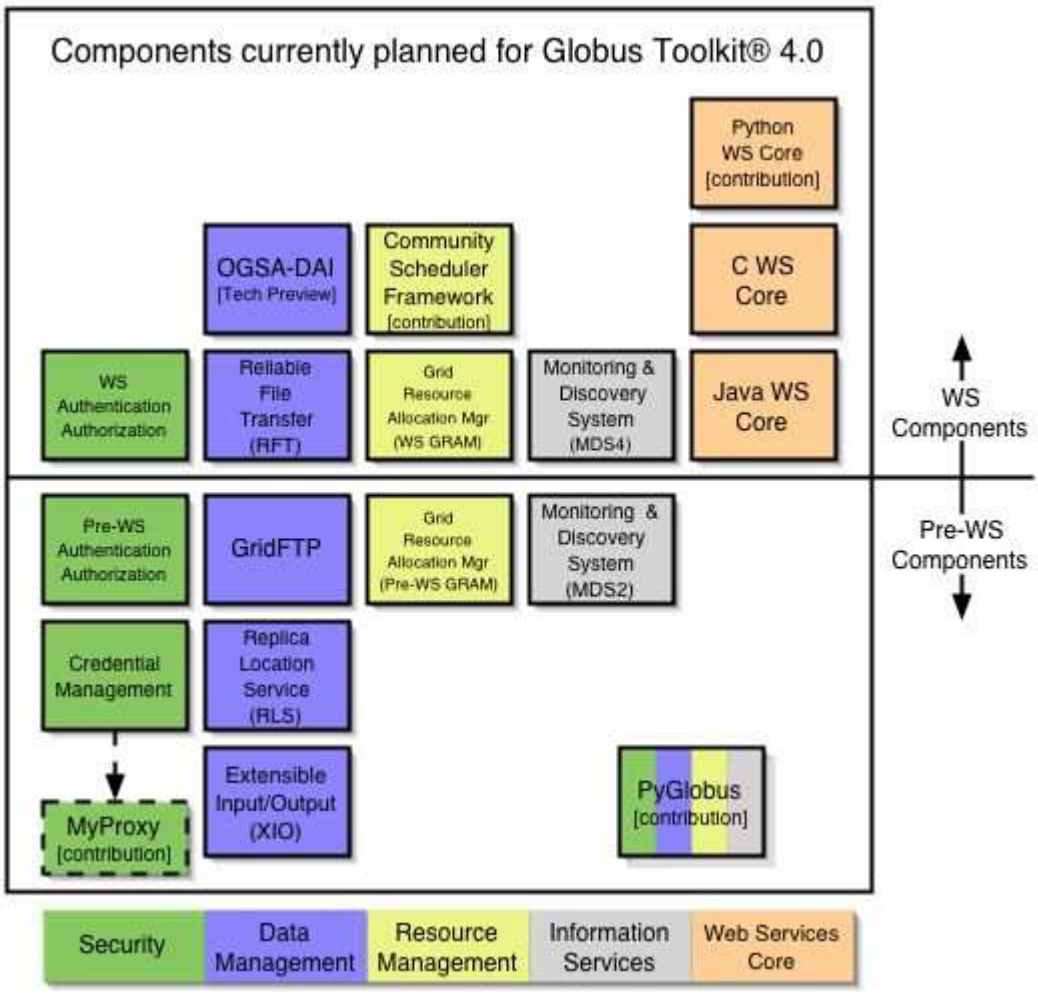
WS-BrokeredNotification: Define el interfaz de un 'NotificationBroker' que implanta

un servicio intermediario para administrar suscripciones para otras entidades productoras de mensajes de notificación.

MIGRACIÓN CONCEPTUAL DE GT 2.4 A GT 4

Una visión panóptica

Son algunos los componentes que cambiarán radicalmente debido a la entrada de los Web Services. Como una imagen vale más que mil palabras, he aquí un pequeño esquema donde se separan los nuevos componentes que reemplazan a los antiguos:



Cada uno de los componentes más importantes serán analizados a continuación.

WS Authentication & Authorization

Contiene las librerías que hacen posible la autenticación y la autorización para los Componentes WS. Esto incluye funcionalidades como autenticación basada en X.509, en GSI SecureConversation, protección de mensajes, y todo un framework de autorización con varios mecanismos.

OGSA-DAI

El objetivo de OGSA-DAI es el proveer métodos uniformes para descubrir, acceder e integrar recursos de datos heterogéneos, principalmente bases de datos, en el Grid. Las bases de datos oficialmente admitidas son:

- MySQL 3.2.x / 4.0.x
- DB2 8
- Oracle 9i / 10g
- Xindice 1.0
- SQL Server 2000
- PostgreSQL 7.x

Adicionalmente, aunque no de forma oficial, muchas bases de datos con el interfaz JDBC, ODBC o XML:DB, que admiten el SQL o el XPath, pueden publicarse usando OGSA-DAI. Los métodos de envío de datos admitidos son SOAP cliente, a fichero, Streams HTTP(S), y GridFTP y FTP a servidor separado.

Reliable File Transfer (RFT)

Este servicio tiene un rol similar al planificador batch en una máquina. Se le envía un trabajo de transferencia de datos, que podría consistir en 10000 o 100000 ficheros, con la información de los tamaños de buffer, streams, ... La petición, junto con la información necesaria para reiniciar (en el caso de ser necesario) y las notificaciones, son almacenados en una base de datos. El servicio entonces ejecuta una tranferencia GridFTP a terceros por el usuario.

WS Grid Resource Allocation and Management (WS GRAM)

WS GRAM ofrece un único interfaz para pedir y usar recursos remotos para la ejecución de trabajos. GT 4 incorpora hasta 2 implantaciones de GRAM:

- Basada en un protocolo Pre-WS no propietario (Pre-WS GRAM).
- Construida usando interfaces WS (WS GRAM).

WS Monitoring and Discovery System (MDS4)

Se trata de un monitor de nivel de Grid basado en WSRF, usado para descubrimiento y manejo de recursos. Sirve para obtener un gran rango de información relativa a los recursos básicos y colas, y puede servir de interfaz para monitorización de sistemas cluster como Ganglia. Cada servicio basado en WSRF ofrece bastante información de monitorización sobre sí mismo, de tal manera que permite a WS MDS usar sus datos.

Igual que cualquier sistema de monitorización Grid, MDS4 ofrece un servicio de indexado donde los datos son recogidos y almacenados. Además, MDS4 almacena información en la base de datos Xindice, basada en XML, y tiene un interfaz web para poder ver los datos fácilmente.

INSTALACIÓN Y CONFIGURACIÓN (DIARIO DE BITÁCORA)

Obtención de las máquinas e instalación de un Sistema Operativo

Se designaron dos máquinas para el Grid de Prueba:

- *columba.dacya.ucm.es* que albergaría la CA.
- *auriga.dacya.ucm.es*

Inicialmente se pensó en la distribución de Linux Gentoo, debido al abanico de posibilidades que ofrece la instrucción *emerge*. Tras emplear bastantes horas para compilar el núcleo en una sola máquina, se optó por descargar del ftp de Rediris una copia de la distribución Mandrake Linux 10.0. El tiempo de instalación y configuración global se redujo a 1 hora y media, habiendo tenido que hacer esperar a que terminara la instalación de un CD para pasarlo a la otra máquina.

Instalación de dependencias

Fue necesaria la instalación de los siguientes paquetes/aplicaciones:

- *j2sdk-1_4_2_04-linux-i586.bin*
- *db2-2.4.14-10mdk.i586.rpm*

- *db2-devel-2.4.14-10mdk.i586.rpm*
- *libgdbm2-1.8.0-24mdk.i586.rpm*
- *libgdbm2-devel-1.8.0-24mdk.i586.rpm*
- *apache-ant-1.6.2-bin.tar.gz*

Cuenta de usuario 'globus', carpetas y Variables de Entorno

Hemos creado una cuenta de usuario llamada 'globus' que tiene el estado de "trabada". Se ha habilitado un home debido a las Variables de Entorno que se describirán a continuación.

Se creó también una carpeta */usr/local/globus* donde los ficheros de globus tendrán cabida y el propio usuario 'globus' tendrá permisos de lectura y escritura.

Respecto a las Variables de Entorno, se ha editado el fichero *.bashrc* situado en el $HOME de cada usuario dejándolo de la siguiente manera:

```
# .bashrc

PATH=/sbin:/usr/sbin:/bin:/usr/bin:/usr/X11R6/bin:/usr/local/bin:/usr/local/sbin:/usr/local/ant/bin:/usr/local/j2sdk1.4.2_04/bin
ENV=$HOME/.bashrc
USERNAME="USUARIO"
export USERNAME ENV PATH

#-----------------------------------------------------------------------------
# Globus enviroment set up
#-----------------------------------------------------------------------------

export GLOBUS_LOCATION=/usr/local/globus
source $GLOBUS_LOCATION/etc/globus-user-env.sh
export GLOBUS_TCP_PORT_RANGE=60000,65535

export X509_CERT_DIR=/etc/grid-security/certificates
export GRID_SECURITY_DIR=/etc/grid-security


# User specific aliases and functions

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi
```

Instalación de Globus Toolkit 3.2.9

Se procedió a descargar de http://www-unix.globus.org/toolkit/downloads/development/ el siguiente fichero:

- *gt3.9.2-wsrf-source-installer.tar.gz*

Hay que tener cuidado a la hora de descomprimirlo, ya que el nombre del dueño de los ficheros figura como *1817*, y esto nos puede dar problemas. Así, usamos el siguiente comando:

tar xzvf gt3.9.2-wsrf-source-installer.tar.gz

Y cambiamos el propietario de todos los ficheros que hay dentro a continuación (por si acaso):

chown -R root:root gt3.9.2-wsrf-source-installer

Una vez hecho esto, y tras meternos dentro de la carpeta descomprimida, pudimos instalar GT 3.2.9 tranquilamente:

./install-wsrf /usr/local/globus

El proceso de instalación es bastante largo, dura aproximadamente 4 horas en un Pentium IV. Inicialmente se construye el *gpt*, que responde a Globus Pack Toolkit, y este va construyendo el resto de los módulos.

Creación de Certificados: Estableciendo la CA

Se ha designado a 'columba' como CA, con lo que en dicha máquina se introdujo el siguiente comando como 'globus' (importante):

/usr/local/globus/setup/globus/setup-simple-ca

En vez de usar el certificado estándar de Globus, se ha decidido crear uno nuevo con las siguientes características:

cn=THE MATRIX CA, ou=TMRCA-columba.dacya.ucm.es, ou=THEMATRIX, o=Grid

Al fin y al cabo, los impulsos eléctricos generados por la humanidad conectada a Matrix son un recurso bastante preciado, ¿no?

Una vez completada la generación del CA, pasamos a finalizar su instalación con el siguiente comando como 'root':

/usr/local/globus/setup/globus_simple_ca_[HASH]_setup/setup-gsi -default

Finalmente, hay que crear un árbol de directorios tal que así:

```
+ /etc
    + /grid-security
        |_/certificates
```

Como estamos creando una nueva CA, tendremos que pasarle a 'auriga' el fichero tar.gz que encontramos en la carpeta */usr/local/globus/setup/globus_simple_ca_[HASH]_setup/*, y una vez hecho esto, ejecutar:

gpt-build FICHERO.tar.gaz gcc32dbg

Sobre la colocación de los certificados, se hablará en un apartado más adelante llamado “Estructura de Certificados”.

Creación de Certificados: Máquinas

Esta operación se ha realizado en ambas máquinas. Para empezar, hay que pedir el certificado de host, y para ello, como 'root' se introduce el siguiente comando:

grid-cert-request -host `hostname`

Nótese la orientación de las comillas, ya que en Mandrake es como revela el nombre del host. De todas formas, no está de más comprobar la correcta sintáxis empleando el comando 'echo'.

Una vez generada la petición, la llevamos a 'columba' (de no estar dicho fichero en ella) y allí, como 'globus', firmamos el certificado:

grid-ca-sign -in hostcert_request.pem -out hostsigned.pem

Y acto seguido, este certificado tiene que ser pasado, como 'root', al */etc/grid-security* de la máquina respectiva y tiene que tener por nombre 'hostcert.pem'.

Creación de Certificados: Usuarios

Como usuario normal, ejecutamos el siguiente comando para solicitar un certificado:

grid-cert-request

Después, en 'columba' y como 'globus', firmamos el certificado de usuario:

grid-ca-sign -in usercert_request.pem -out signed.pem

El certificado final tiene que ser colocado en *$HOME/.globus/* y tiene que tener por nombre 'usercert.pem'.

Podemos comprobar, como usuario normal, el éxito de la operación introduciendo lo siguiente:

grid-proxy-init -debug -verify

Estructura de Certificados

En este apartado se analiza lo que tiene que contener el */etc/grid-security/*, en caso de que haya fallado algo:

/etc/grid-security/globus-user-ssl.conf --> /etc/grid-security/certificates/globus-user-ssl.conf.XXXXXX
Define el nombre empleado para el certificado de usuario
/etc/grid-security/globus-host-ssl.conf --> /etc/grid-security/certificates/globus-host-ssl.conf.XXXXXX
Define el nombre empleado para el host (o servicio)

/etc/grid-security/grid-security.conf --> /etc/grid-security/certificates/grid-security.conf.XXXXXX
Un fichero de configuración base de los anteriores

Los ficheros de la izquierda son enlaces dinámicos de los de la derecha.

/etc/grid-security/certificates/XXXXX.0
Certificado del CA de confianza
/etc/grid-security/certificates/XXXXX.signing_policy
Fichero de configuración con el DN del CA

Como aspecto importante, este árbol de directorios (y sus contenidos) deben pertenecer a 'globus'.