ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERÍA SUPERIOR INFORMÁTICA

Integración del protocolo OAuth en el Servicio de Identidad de RedIRIS

Realizado por
ELENA LOZANO ROSCH

Dirigido Por
DIEGO R. LOPEZ
Y
MANUEL VALENCIA

Departamento
DEPARTAMENTO DE TECNOLOGÍA ELECTRÓNICA

Sevilla, Junio de 2010

Índice general

Índice de figuras

Capítulo 1

Introducción

Este proyecto, "Integración del protocolo OAuth en el Servicio de Identidad de RedIRIS", tiene como objetivo implementar un servicio de autenticación y autorización de acceso a la información aportada por los servicios de RedIRIS[1] para usuarios que pertenezcan a la red académica.

La idea de este servicio es proporcionar un sistema flexible, cuyo aprendizaje e implantación no suponga un esfuerzo excesivo y que permita incorporar aplicaciones personalizadas a los servicios ofrecidos por las instituciones suscritas al mismo.

El caso práctico en el que comenzó a desplegar esta herramienta fue implementar una aplicación cliente que permitiera acceder de forma segura a la información relativa a la inscripción y registro de las *Listas de distribución de RedIRIS*[2]. Esta pretende facilitar el intercambio de conocimientos en la Comunidad académica española usando el correo electrónico, proporcionando un aumento de la colaboración, mantener el contacto entre profesionales y el anuncio de eventos, entre otros.

[1] RedIRIS es la red académica y de investigación nacional, patrocinada por el Plan Nacional de I+D+I y que está gestionada por Red.es

[2] `http://www.rediris.es/list`

Para poder llevar a cabo este proyecto se ha utilizado como tecnología principal OAuth[1], protocolo de autenticación y autorización que desarrollaremos con detenimiento en el capítulo 3.

Uno de los motivos por el cual se ha utilizado esta tecnología ha sido la cualidad de suavizar la curva de aprendizaje en el despliegue del servicio en la institución correspondiente. En el capítulo dedicado a la solución desarrollada (capítulo 4), se exponen este y otros motivos por el cual se eligió este protocolo en concreto, además de la estructura y diseño de la solución realizada.

Este documento contiene, además, un capítulo a modo introductorio (capítulo 2) que comenta los conceptos básicos necesarios para comprender el contexto y la solución aportada y un capítulo (capítulo 5) en el que se comparan los esfuerzos estimados inicialmente con los esfuerzos requeridos para desarrollar el proyecto.

En el capítulo de Conclusiones (capítulo 6), se analizan las decisiones tomadas durante la realización de este proyecto y la proyección de futuro que se le pretende dar al mismo.

En último lugar se encuentran los apéndices, los cuales contienen una guía de implantación donde se explica con todo detalle las formas de instalación, configuración e implantación del servicio, un manual de uso para la aplicación cliente de ejemplo, así como el código implementado para las aplicaciones del servicio.

Capítulo 2

Conceptos generales

2.1. Glosario

Autenticación: Procedimiento por el cual un usuario se identifica en una organización. En caso positivo obtiene unas credenciales con la información asociada a su identidad.

Autorización: Procedimiento por el cual, una vez autenticado el usuario y dependiendo de las credenciales o atributos que tenga, se le podrá permitir el acceso a los recursos.

Proveedor de Identidad: Realiza la autenticación del usuario y emite sus credenciales.

Proveedor de servicio: Comprobará las credenciales y dependiendo de éstas dará acceso al recurso o lo denegará.

Credenciales: Incluirán información acerca de si se ha realizado correctamente la autenticación y los atributos asociados al usuario. Intenta reducir información sensible (nombre, apellidos, DNI etc.) y reflejar datos más generales como tipo de usuario, etc. Podrán ir cifradas de forma que solo los proveedores de identidad y de servicio puedan leerlas.

Organización del usuario: Organización a la cual pertenece el usuario.

Organización propietaria: Organización que tiene los recursos a los que quiere acceder el usuario.

Recurso: Elemento al cual se quiere acceder. Podrá estar protegido.

2.2. Proceso de autenticación y autorización:

Existen dos sistemas de autenticación y autorización: Federados y No Federados: Un sistema no federado usualmente consiste en utilizar una autenticación con nombre de usuario y contraseña, aunque existen otros métodos como el DNI Electrónico, tarjetas de identificación, etc.

A simple vista podemos observar que este modelo puede tener ciertas desventajas:

- Incrementa los costes debido a un aumento en el tiempo de registro: el usuario deberá introducir su nombre de usuario y contraseña múltiples veces.
- Aumento del tiempo de administración, ya que al existir más usuarios hay que gestionar más cuentas de identidad.
- Posibles inconsistencias de credenciales y de permisos de acceso.
- Aumenta el coste en recursos físicos porque hay más red y más sistemas hardware ocupados al almacenar los datos de otras organizaciones.
- Puede llevar a problemas de organización y de políticas de privacidad al ser necesario almacenar más información de los usuarios en otras organizaciones.
- Gran foco de inseguridad debido a que el usuario tiene que recordar múltiples usuarios y contraseñas.

Debido a estas desventajas, se comenzó a implementar una alternativa a estos sistemas: los sistemas federados. El objetivo de éstos consiste en delegar la identificación de cada usuario en la

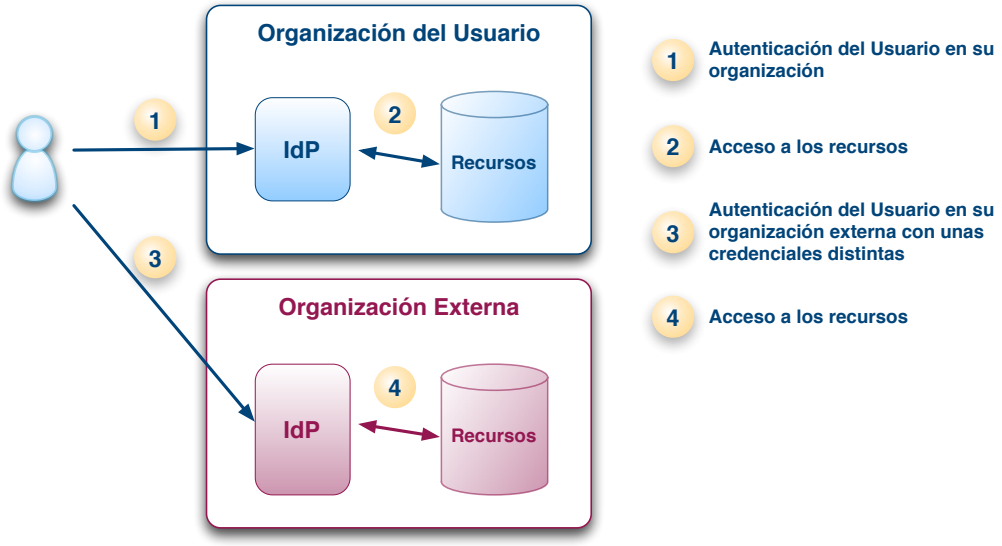


Figura 2.1: Diagrama de Autenticación y Autorización en un Sistema No Federado

organización a la que pertenece, realizándose en ésta sólo la autenticación. El procedimiento que se lleva a cabo con éstos es el siguiente:

- Cuando un usuario desea acceder a un recurso protegido, éste podrá ser de la misma organización del usuario o de una organización externa.
- En ambos casos, el usuario se identificará como hace usualmente (usuario y contraseña u otro método) contra un **Proveedor de Identidad** *(Identity Provider)* de su organización.
- Con este método, el usuario obtiene sus credenciales o atributos, en los cuales se refleja información relacionada con el usuario.
- A partir de aquí, cuando el usuario quiera acceder a otro recurso (externo a su organización o no) no tendrá que autenticarse de nuevo, sino que presentará las credenciales obtenidas anteriormente siendo el **Proveedor de Servicio** *(Service Provider)* el que comprobará las credenciales y según éstas dará acceso o lo denegará.

Figura 2.2: Diagrama de Autenticación y Autorización en un Sistema Federado

Las ventajas de un sistema federado son evidentes:

- Ahorro de costes en tiempo de administración y de identificación.
- Disminución de los costes físicos y de hardware.
- Aumenta la seguridad por haber reducido el número de usuarios y contraseñas diferentes para una misma persona.
- Los recursos están almacenados de forma más coherente.

2.2.1. Soluciones federadas:

Actualmente existen distintas tecnologías que dan solución a los sistemas federados se autenticación y autorización, entre ellos:

PAPI [23] Este software desarrollado por diversas organizaciones españolas permite desplegar una federación de identidad digital con un sistema de delegación en la autorización. Es ampliamente utilizado por las instituciones académicas y científicas españolas.

SAML [28] Lenguaje de marcas de aserciones de seguridad (*Security Assertion Markup Language*). Estándar basado en XML para el intercambio de datos correspondientes a autenticación y autorización entre dominios de forma segura.

OpenID [25] Esta tecnología proporciona un sistema de Single Sign-On descentralizado y abierto donde el usuario se identifica a través de una URL. Cada vez es más popular dentro de las aplicaciones llamadas web 2.0.

OAuth [1] Protección de recursos orientados a delegación; es decir, una aplicación denominada Cliente, quiere obtener un conjunto de datos protegidos por otra aplicación (Servidor). El objetivo es permitir el acceso a unos datos contenidos en el servidor desde la aplicación cliente sin que se compartan las credenciales que los usuarios tienen en la aplicación servidora.

Capítulo 3

OAuth

3.1. Introducción

OAuth (*Open Authorization; Autorización abierta*) es un estándar abierto que permite incluir seguridad en la autenticación para aplicaciones web y de escritorio. Este protocolo surgió de la necesidad de incluir una herramienta que permitiera delegar la autenticación de Twitter[1] usando OpenID[2]. Los desarrolladores encargados del proyecto, al ver que no existía ningún estándar abierto y que el problema era extensible a otras tecnologías, comenzaron a desarrollar uno que se adecuara a las necesidades del momento.

OAuth combina los factores comunes y las buenas prácticas existentes entre sistemas de autenticación de tecnologías emergentes dentro de lo denominado web 2.0, como por ejemplo las librerías

[1] http://twitter.com/
[2] http://openid.net/

Google AuthSub[3], AOL OpenAuth[4], Flickr API[5], Amazon Web Services API[6] , etc.

Aunque OAuth se base en estos protocolos, ha ido un paso mas allá y no solo implementa la forma de proteger un recurso web, sino también elementos de otras aplicaciones, como pueden ser las de escritorio o las móviles.

Para ilustrar las ventajas de OAuth, veamos el siguiente ejemplo: Una aplicación nueva incluye un servicio de avisos de fechas importantes que deberá importar de otra aplicación, por ejemplo, la agenda del gestor de correo. Para poder activar el servicio es necesario que el sujeto en cuestión le de su nombre de usuario y contraseña para así acceder al gestor de correo. El resultado obtenido es que ambas aplicaciones funcionan correctamente, pero existe un inconveniente primordial: la aplicación que da el servicio de avisos ya tiene su contraseña, lo cual puede llevar a un problema de seguridad grave.

OAuth intenta solucionar esta problemática restringiendo el acceso a los recursos. En el ejemplo anterior, mediante OAuth se habría conseguido que la aplicación servidora sólo pudiera acceder a la información del calendario, evitando así tener que darle el nombre de usuario y contraseña y que hubiera un acceso ilimitado a la información contenida en el gestor de correo. En vez de utilizar nombre de usuario y contraseña, OAuth utilizará unas credenciales que permitirán el acceso a uno o varios recursos por un periodo de tiempo determinado.

[3] http://code.google.com/apis/accounts/AuthForWebApps.html
[4] http://dev.aol.com/openauth
[5] http://www.flickr.com/services/api/
[6] http://aws.amazon.com/

3.2. Conceptos Básicos

A continuación se definen los elementos básicos que serán necesarios para la comprensión del protocolo OAuth.

Servidor (*Server*) Es el servidor HTTP donde se encuentran los recursos que están protegidos mediante OAuth y se encarga de denegar o permitir el acceso a los mismos. Es capaz de responder a peticiones de OAuth autenticadas.

Propietario del Recurso (*Resource Owner*) Entidad capaz de acceder y controlar recursos protegidos utilizando unas credenciales determinadas al autenticarse en el Servidor.

Cliente (*Client*) Aplicación web que utiliza OAuth para acceder al Servidor en nombre del Propietario del Recurso o en su propio nombre. Es un cliente HTTP que es capaz de hacer peticiones de OAuth autenticadas.

Recurso Protegido (*Protected Resource(s)*) Datos del Propietario del Recurso a los que desea acceder el Cliente.

Credenciales (*Credentials*) Son un par de identificador único y secreto compartido. Existen tres tipos de credenciales distintas: del Cliente, Temporales y de Token.

Credenciales del Cliente (*Client Credentials*) Identifican y autentican al cliente que hace la petición.

- Identificador del Cliente (*Client Identifier*) Valor que identifica al Cliente en el Servidor.
- Secreto compartido del Cliente (*Client Shared-Secret*)] Secreto que utiliza el Cliente y que lo acredita en el Servidor.

Credenciales Temporales (*Temporary Credentials*) Identifican y autentican a la petición de autorización. Son utilizadas por el Cliente para obtener autorización del Propietario del Recurso. Será

una cadena alfanumérica única relacionada con un secreto del Cliente correspondiente.

Credenciales del Token (*Token Credentials*) Identifican y autentican al acceso concedido. Permite acceder a los Recursos Protegidos y es el paso posterior a la obtención de las credenciales temporales.

En apartados posteriores analizaremos estos elementos con más profundidad.

3.3. Flujo de Autorización

Los pasos principales que se realizan para proveer acceso a unos recursos restringidos mediante OAuth son los siguientes:

1. El Cliente obtiene unas Credenciales Temporales del Servidor.
2. El Cliente pide autorización del propietario del recurso.
3. El Propietario del recurso o Propietario del Recurso valida la petición y da autorización para acceder al mismo.
4. El Cliente intercambia las credenciales temporales por las del token y accede al recurso protegido.

Antes de desarrollar cada uno de estos pasos, es necesario comentar las normas que deben cumplirse para que el procedimiento de autorización tenga garantizado un nivel de seguridad adecuado. Éstas son las siguientes:

- El Cliente deberá poder autenticar tanto al Servidor como al Propietario del Recurso.

- El Cliente deberá poder autenticar el identificador que comparten el Servidor y el Propietario del Recurso.
- El Servidor deberá poder autenticar al Cliente y al Propietario del Recurso, sin necesidad de que éstos tengan que autenticarse entre ellos.

Además, para la correcta comprensión del flujo de autenticación es necesario definir varios elementos básicos que son utilizados en él:

- Para acreditar al Cliente, se utilizarán las **Credenciales del Cliente**, los cuales identificarán al Cliente ante el Servidor.
- Como comentamos anteriormente, en vez de utilizar las credenciales del Propietario del Recurso en las peticiones a recursos protegidos, Oauth utiliza las **Credenciales Temporales** y del **Token**. Las **Credenciales Temporales** son utilizadas por el Cliente para pedir al Propietario del Recurso acceso a los Recursos Protegidos.

  Una vez que éstas son autorizadas por el Propietario del Recurso, se intercambian por las **Credenciales del Token** las cuales serán utilizadas por el Cliente para acceder a los Recursos Protegidos en nombre del Propietario del Recurso.

  Éstas pueden englobar a más de un recurso y tener un tiempo de vida limitado. Solo estas credenciales podrán usarse para acceder a los recursos protegidos, siendo susceptibles de ser revocadas.

A continuación, una vez establecidos estos aspectos básicos, desarrollamos los pasos del flujo de autorización de OAuth.

3.3.1. Obtención de las Credenciales Temporales (*Temporary Credentials*)

El objetivo de este paso es que el Cliente obtenga unas Credenciales Temporales. Para ello, el Cliente envía una petición HTTP POST a la dirección de destino dada por el Servidor para estos casos, denominada *Temporary Credential Request Endpoint*. En ella se envían las Credenciales del Cliente, para que el Servidor pueda contrastarlas con las que tiene almacenadas y verificar la autoría de la petición.

Además, la petición deberá estar firmada y contener los siguientes parámetros:

- `oauth_consumer_key`: Identificador de las Credenciales del Cliente
- `oauth_signature_method`: El método de firma que utiliza el Cliente para firmar la petición.
- `oauth_signature`: La firma tal y como se define en el procedimiento *Signing Requests* (El cual se comenta más adelante).
- `oauth_timestamp`: Marca de tiempo que permitirá a los Proveedores de Servicio mantener los identificadores por un tiempo limitado.
- `oauth_nonce`: Número identificador de la petición.
- `oauth_callback`: URL[7] absoluta al cual el Servidor redireccionará al Propietario del Recurso una vez que obtenga las Credenciales Temporales autorizadas adecuadamente.
- Otros parámetros adicionales podrán ser definidos por el Servidor.

Una vez enviada la petición, el Servidor la recibe y verifica la firma y las Credenciales del Cliente que van en ella. En caso de ser verificada correctamente, el Servidor genera unas Credenciales

[7]Localizador Uniforme de Recursos (sigla en inglés de *Uniform Resource Locator*),

Temporales que representarán al Propietario del Recurso en el Servidor para esa conexión concreta, difiriendo del nombre de usuario y contraseña originales.

Tras generarlos, los devuelve al Cliente codificados en una respuesta HTTP de tal forma que cumplan el RFC3986[10], donde se establece una codificación estándar para las URL.

El Servidor deberá asegurarse de que las Credenciales Temporales no puedan ser intercambiadas por las Credenciales del Token hasta que el Propietario del Recurso dé acceso en la fase posterior donde se obtiene su autorización. La respuesta que el Servidor enviará al Cliente deberá contener los siguientes parámetros:

- `oauth_token`: EL identificador de las Credenciales Temporales.
- `oauth_token_secret`: El secreto compartido de las Credenciales Temporales.
- `oauth_callback_confirmed`: Con valor falso o cierto, confirma o no si el Servidor ha recibido la URL de *callback*. En las versiones más actuales del protocolo, este valor debe estar a verdadero; ya que se utiliza para diferenciar la respuesta de versiones previas del protocolo.
- Otros parámetros adicionales podrán ser definidos por el Servidor.

En caso de que se no se verifique la firma o se rechace la petición por alguna otra razón, el Servidor deberá responder de forma adecuada, es decir, con mensajes de error HTTP 400 o HTTP 401[11], pudiéndose incluir además algunos detalles sobre la causa del rechazo.

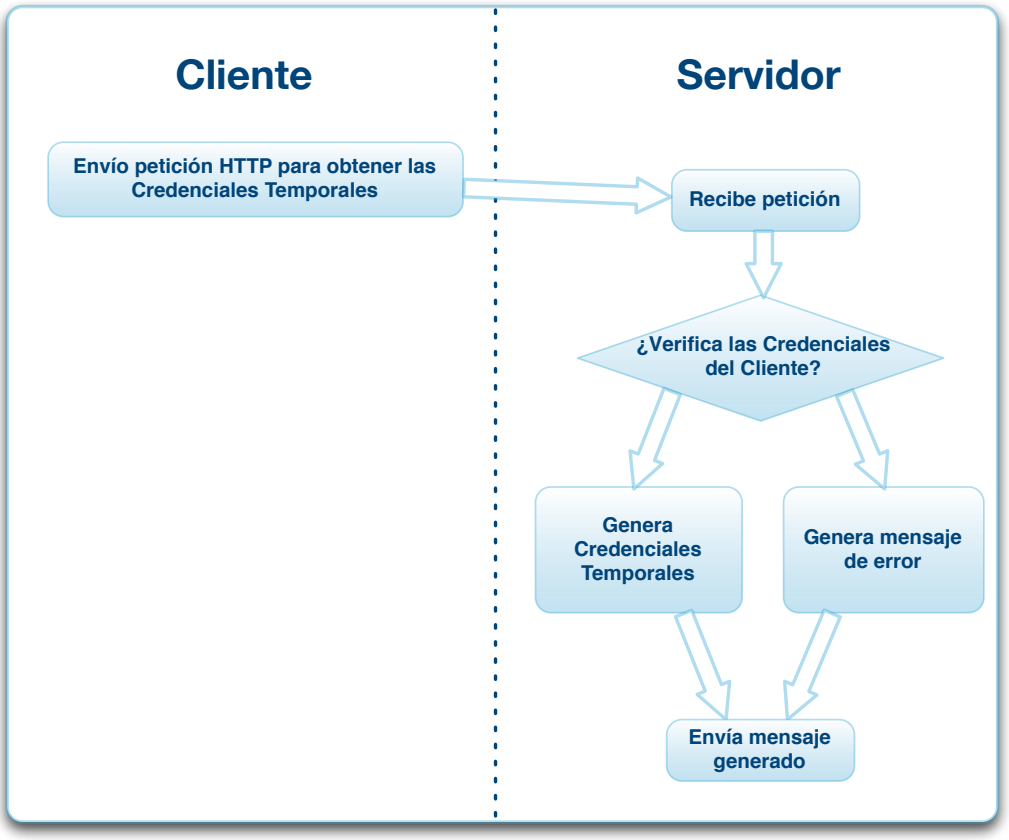


Figura 3.1: Obtención de las Credenciales Temporales.

3.3.2. Autorización del Propietario del Recurso y validación de la petición

El objetivo de este segundo paso es que el Propietario del Recurso autorice las Credenciales Temporales.

Para ello, el Cliente redirige al Propietario del Recurso a la URL que el Servidor tiene establecida para el tratamiento de las autorizaciones, denominada *Resource Owner Authorization Endpoint*.

El Cliente envía en la petición HTTP GET los siguientes parámetros:

- `oauth_token`: El identificador de las Credenciales Temporales obtenidas en el paso anterior.
- Otros parámetros adicionales podrán ser definidos por el Servidor.

Una vez redirigido el Propietario del Recurso en su navegador web a la aplicación servidora, ésta deberá realizar los siguientes pasos, en orden:

1. Autenticar al Propietario del Recurso y verificar que los recursos a los que se desea acceder le pertenecen.
2. Verificar que el identificador de las Credenciales Temporales que ha recibido no se utilizó en otra petición.
3. Generar un token de verificación aleatorio que sea único.
4. Almacenar la relación existente entre el token de verificación, la identidad del Propietario del Recurso y la del Cliente con respecto a las Credenciales Temporales.
5. Enviarle al cliente de forma segura un mensaje con el token de verificación y el identificador de las Credenciales Temporales. Esto se hará mediante la redirección del Propietario del Recurso de nuevo al Cliente (a la URL de *callback* que se dio en pasos anteriores), siendo los parámetros de la Petición HTTP GET los siguientes:
   - `oauth_token`: El identificador de las Credenciales Temporales obtenidas en el paso anterior.
   - `oauth_verifier`: Código de verificación.

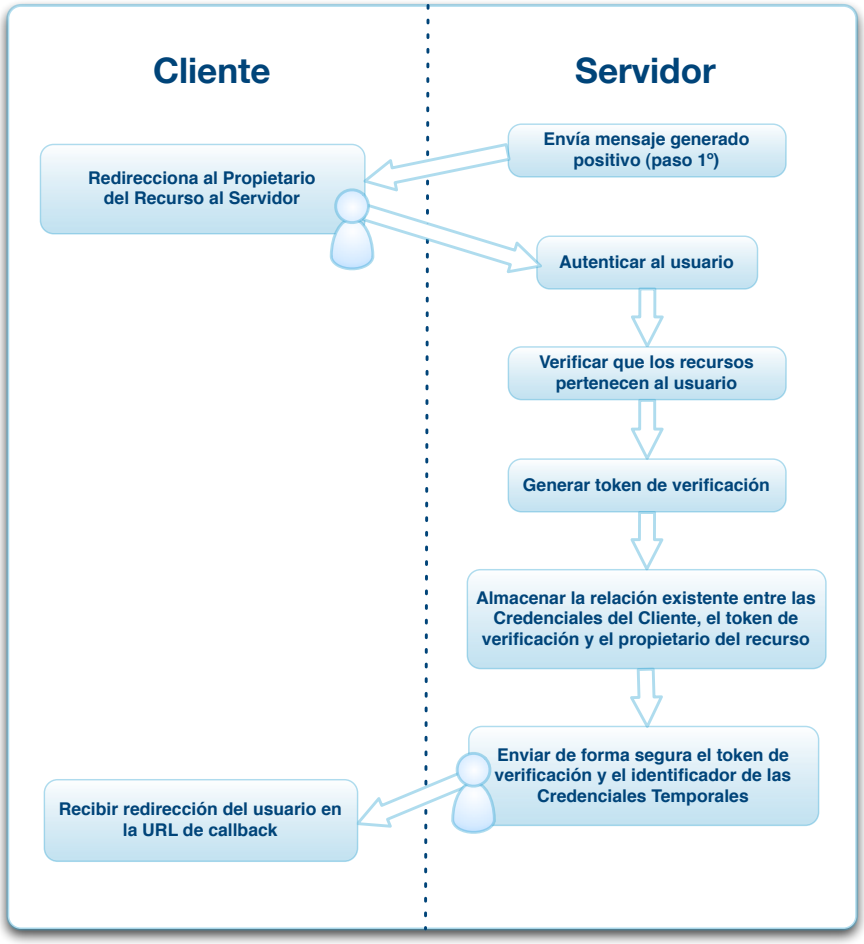


Figura 3.2: Autorización del Propietario del Recurso y validación de la petición.

3.3.3. Intercambio de las Credenciales Temporales por las Credenciales del Token

En este último paso, se intercambian de las Credenciales Temporales por las Credenciales del Token.

Para ello, el Cliente envía una petición HTTP POST a la URL del Servidor especificada para ello, denominada *Token Request Endpoint*. Deberá ir firmada por el método *signing requests* que comentamos posteriormente y contener los siguientes parámetros:

- `oauth_consumer_key`: Identificador de las Credenciales del Cliente.
- `oauth_token`: El identificador de las Credenciales Temporales obtenido anteriormente.
- `oauth_signature_method`: El método de firma que utiliza el Cliente para firmar la petición.
- `oauth_signature`: La firma del tal y como se define en el procedimiento *Signing Requests* (El cual se comenta más adelante).
- `oauth_timestamp`: Marca de tiempo que permitirá a los Servidores mantener los identificadores por un tiempo limitado.
- `oauth_nonce`: Número identificador de la petición.
- `oauth_verifier`: Código de verificación obtenido en el paso anterior.

El Servidor, al recibir la petición anterior, deberá asegurarse de que la firma, las Credenciales Temporales y del Cliente y el Código de Verificación coinciden correctamente y además, que esas Credenciales Temporales no se han utilizado previamente o hayan expirado. Si todos estos aspectos se verifican, el Servidor devuelve un mensaje HTTP con el formato de respuesta

`application/x-www-form-urlencoded` definido por la W3C[8] en la especificación de HTML, en su versión 4.0[12]. Los parámetros obligatorios en el envío de esta petición son los siguientes:

- `oauth_token`: El identificador de las Credenciales del Token.
- `oauth_token_secret`: El secreto compartido de las Credenciales del Token.

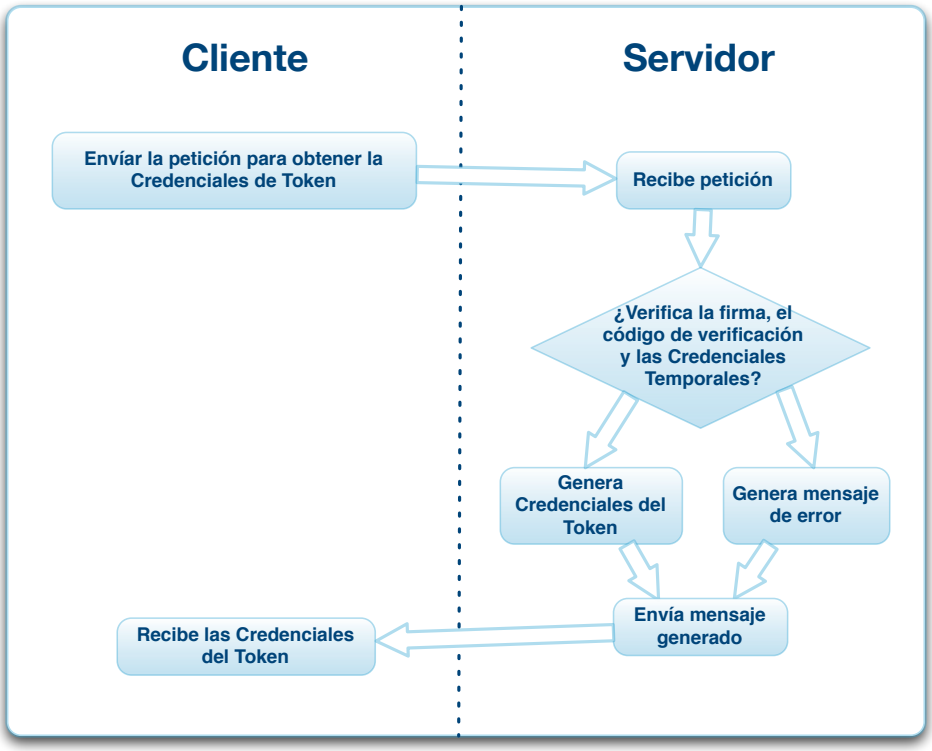


Figura 3.3: Intercambio de las Credenciales Temporales por las Credenciales del Token

El Servidor deberá almacenar la duración, el objetivo y otros atributos autorizados por el Propietario de los Recursos y obligará al Cliente a cumplir estas especificaciones a la hora de acceder

[8] *World Wide Web Consortium*; Consorcio internacional que produce recomendaciones para la World Wide Web

a los Recursos Protegidos.

3.3.4. Acceso a los Recursos Protegidos

Como último paso para que el Cliente acceda a los Recursos Protegidos en nombre del Propietario de los Recursos, éste deberá enviar una petición debidamente firmada por el método *signing requests* (comentado posteriormente) con los siguientes parámetros:

- `oauth_consumer_key`: Identificador de las Credenciales del Cliente.
- `oauth_token`: El identificador de las Credenciales de Token obtenido anteriormente.
- `oauth_signature_method`: El método de firma que utiliza el Cliente para firmar la petición.
- `oauth_signature`: La firma del tal y como se define en el procedimiento *Signing Requests* (El cual se comenta más adelante).
- `oauth_timestamp`: Marca de tiempo que permitirá a los Servidores mantener los identificadores por un tiempo limitado.
- `oauth_nonce`: Número identificador de la petición.
- `oauth_verifier`: Código de verificación obtenido pasos anteriores.
- Otros parámetros adicionales podrán ser definidos por el Servidor.

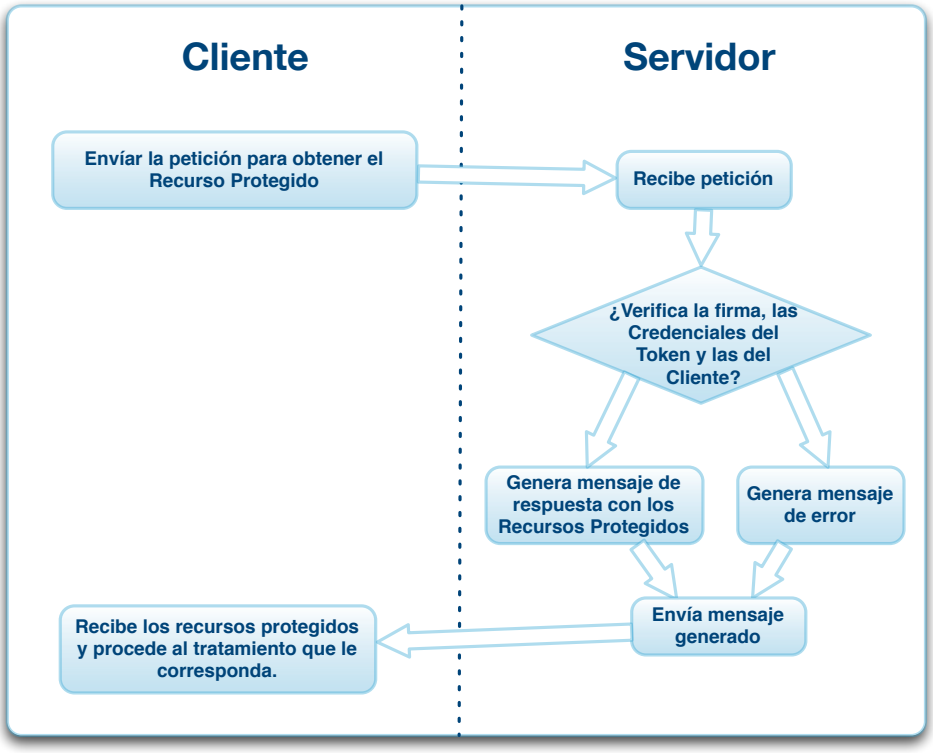


Figura 3.4: Intercambio de las Credenciales Temporales por las Credenciales del Token

3.4. Seguridad en OAuth

Una vez comentado como se realiza la comunicación entre un Servidor y un Cliente puede surgir la duda sobre cuál es la garantía de que en este proceso las peticiones sean interceptadas y modificadas. La seguridad en OAuth viene dadas por la tres peculiaridades existentes en su arquitectura:

- Los datos enviados con las peticiones deben enviarse cifrados para evitar que una tercera persona pueda interceptarlos y hacer un uso ilícito de ellos.
- Debe existir un mecanismo que asocie una petición con unas credenciales concretas, ya que de no realizarse así unas credenciales correctas podrían ser utilizadas en cualquier petición que se enviara.
- Debe permitirse interactuar con dos tipos de credenciales, las del Cliente, que garantizan que éste es un Cliente autorizado previamente y las del Propietario del Recurso, que lo certifican como Propietario del Recurso del Proveedor del Servicio.

La solución a estas problemáticas se da gracias al método por el que OAuth define un método para validar la autenticidad de las peticiones HTTP. Este método se denomina *Signing Request* (Firma de Petición) y consiste, básicamente en que todas las peticiones de tokens y recursos protegidos deberán estar firmados por el Cliente y verificados por el Servidor.

La configuración de este método dependerá del escenario de la aplicación que se tenga. En cualquier implementación de OAuth podemos establecer dos tipos de escenarios básicos de seguridad:

- Acceso Directo (*2-legged scenario*): Hay dos actores en el flujo de autorización: Cliente y Servidor. Este acceso se da cuando el Cliente pide acceso a los recursos en su nombre. En este

caso el Cliente y el Propietario del Recurso serán la misma entidad y sólo será necesario enviar las Credenciales del Cliente.

- Acceso Delegado (*3-legged scenario*): Hay tres actores en el flujo de autorización, Cliente, Propietario del Recurso y Servidor. El Cliente pide acceso a los recursos en nombre de un Propietario del Recurso concreto. En este caso es necesario enviar las Credenciales del Propietario del Recurso y las del Cliente.

En ambos escenarios se utilizarán firmas digitales en vez de enviar las credenciales completas. Esto permite verificar que el contenido de la petición no se ha modificado en el camino entre una aplicación y otra. La garantía de confianza dependerá del algoritmo de firma que se utilice y de la forma de aplicación del mismo. OAuth soporta todo tipo de algoritmos de resumen y dependerán de la configuración que se realice.

Como los métodos de firmado verifican la integridad de los datos, pero no la identidad del que envía el mensaje, es necesario un nuevo elemento de seguridad: Para verificar la identidad de éste, se combina el algoritmo utilizado en la firma con un secreto compartido. Para esto es necesario que ambos lados de la comunicación acepten un secreto que sólo es conocido entre ellos y que se enviará en el resumen, siendo así posible verificar la autoría de la petición.

Según el protocolo sobre el que vaya OAuth, se utilizará un tipo de firma u otro: En caso de ser HTTPS, el método se denomina **PLAINTEXT** y delega la mayor parte de la seguridad en la capa HTTPS, sin tener necesidad de firmas y resúmenes. Cuando va sobre HTTP, los métodos de seguridad deben ser mucho más elaborados, siendo éstos son los siguientes:

- **HMAC-SHA1**: Ofrece secreto compartido simétrico y SHA1[8] como método resumen.
- **RSA-SHA1**: La clave privada del Cliente se usa exclusivamente para firmar peticiones y sirve como secreto compartido asimétrico. Esto funciona de forma que tanto el Cliente como el

Servidor utilizan una clave para firmar la petición y otra para verificarla. La clave, Privada para el Cliente y Pública para el Servidor, deben coincidir de tal forma que solo el par adecuado podrá firmar y verificar la petición. La ventaja de utilizar secreto compartido asimétrico es que el Servidor no tendrá acceso a la clave privada del Cliente, disminuyendo así la probabilidad de que se filtre la clave.

El procedimiento de firmado protege el contenido de cambios durante la transmisión por el medio y el secreto compartido asegurará que las peticiones solo puedan ser emitidas por un consumidor autorizado. Esta estructura aún permite un tipo de ataque concreto, mediante el cual una tercera parte intercepta los paquetes de la transmisión y los reutiliza sin modificarlos. Para prevenir este tipo de reenvíos, OAuth incorpora un mecanismo mediante el cual se puede incluir en la firma dos nuevos elementos:

- Número identificador (*nonce*): El Cliente podrá generar una cadena única por cada petición que envíe al Servidor; si éste ve que el identificador se repite podrá prevenir el ataque mediante reenvío.
- Marca de Tiempo (*timestamp*): Añadiendo una marca de tiempo en cada petición permitirá a los Servidores mantener los identificadores por un tiempo limitado, ahorrando así una gran cantidad de espacio de memoria. Cuando una petición llega con una marca de tiempo que es anterior a la franja de tiempo que se tiene en el sistema, el Servidor la rechazará.

Aunque pueden utilizarse independientemente, la mejor configuración posible en cuanto a seguridad se refiere, es la combinación de ambos elementos.

3.4.1. Otras Consideraciones de Seguridad

Aunque el protocolo de OAuth intenta prevenir con su especificación los riesgos de seguridad que existen, no puede solucionar otros problemas de seguridad que son intrínsecos en las comunicaciones

en red. A continuación comentamos consideraciones a tener en cuenta acerca de estos riesgos y las decisiones que deben tomarse para prevenirlos.

Transmisión de Credenciales Los servidores deben asegurarse de que las transmisiones de las credenciales se realiza de forma segura, utilizando mecanismos de seguridad de la capa de transporte como pueden ser TSL[9] o SSL [10].

Método de firma RSA-SHA1 Las peticiones hechas mediante este método no usan los secretos compartidos que se definen en el protocolo, si no que se utiliza la clave privada del Propietario del Recurso, por lo que la seguridad del mismo depende de la seguridad con la que sea almacenada la clave privada en cuestión.

Método de firma PLAINTEXT Con este método no se protege de ninguna forma las credenciales, ya que está pensado especialmente para realizarse sólo en conexiones protegidas mediante los protocolos de la capa de transporte SSL o TLS.

Confidencialidad de las peticiones El protocolo de OAuth provee un mecanismo para verificar la integridad de las peticiones, pero no da garantía de confidencialidad. Si no se toman las precauciones adecuadas, terceras personas podrán tener acceso al contenido de la petición. Para evitar este problema deberán utilizarse los ya citados protocolos de seguridad de la capa de transporte.

Spoofing[11] Los Servidores deberán tener en cuenta de que existe la posibilidad de que existan servidores fraudulentos que se hagan pasar por Clientes. Para evitar este problema deberán utilizarse también TSL o SSL.

Esquema de Autorización Es recomendable utilizar el esquema de Autorización de HTML descrito

[9] *Transport Layer Security*
[10] *Secure Sockets Layer*[7] Ambos son protocolos criptográficos que proporcionan comunicaciones seguras por una red.

en el RFC2616[9], ya que de esta forma se protege la cabecera donde hay parámetros de la petición; evitando que así se pueda filtrar información sensible.

Almacenaje de las Credenciales sin cifrado Debe tenerse en cuenta, que las credenciales que se almacenen en el Servidor deben estar debidamente protegidas, ya que éste las necesita de forma *plaintext*, es decir; sin cifrado, cualquiera que accediese a la base de datos del Servidor podría obtener las Credenciales de todos los Clientes.

Secreto de las Credenciales del Cliente En el caso de un Cliente web, es relativamente fácil mantener segura las Credenciales del Cliente. Sin embargo, cuando es una aplicación de escritorio o una aplicación móvil, las Credenciales deberán estar incluidas en el código fuente de la aplicación, lo cual las hacen mucho más vulnerable.

Phishing[12] Este problema puede provocar que los usuarios sean redirigidos a copias de la aplicación web del Servidor, para así robar sus contraseñas; por esto se debe intentar informar a los Propietarios de los Recursos acerca de los riesgos del *phishing* y dar mecanismos que les faciliten la comprobación de autenticidad de sus aplicaciones.

Alcance de las Peticiones de Acceso Los Servidores deberán tener en cuenta que si necesitan definir un alcance determinado para el acceso a los recursos deberán dar servicio para cada tipo de alcance; asegurándose de informar al Propietario del Recurso de los datos a los cuales va a dar acceso y los riesgos que esto implica.

Entropia de Secretos En caso de que no se utilice un Protocolo de Seguridad de la capa de Transporte (SSL, TSL), es posible que personas no autorizadas accedan a las peticiones de autenticación y firma, por lo que podrían capturar los paquetes y por fuerza bruta obtener las claves. Para evitar esto se recomienda asignar a los secretos compartidos una longitud tal que no sea posible descifrarla por fuerza bruta en el tiempo que tenga de vida.

Ataques de Denegación de Servicio Aunque el protocolo no cubre este riesgo, es necesario que

los implementadores de aplicaciones con OAuth lo tengan en cuenta, ya que es posible que al necesitar muchos identificadores de mensajes (*nonces*) a la vez, la máquina comience a generarlos y haga los recursos se agoten rápidamente. Será necesario establecer un equilibrio entre la dificultad de cómputo al generar éstos códigos y el problema de Denegación de Servicio.

Ataques Criptográficos En el algoritmo de hash utilizado en OAuth, SHA1[8], se conocen debilidades por las que no se hace recomendable su uso. Esta debilidad no puede resolverse aún en OAuth, pero se está estudiando la posibilidad de inclusión de otros métodos de firma más seguros en futuras versiones del protocolo.

Limitaciones de las firmas El método utilizado para las firmas no cubre la petición completa (por ejemplo, excluyen las cabeceras) por lo que los servidores deberán utilizar mecanismos adicionales para proteger esos elementos.

Como conclusión de estas consideraciones, podemos decir que es importante saber y entender las limitaciones de los secretos compartidos, simétricos y asimétricos a la hora de implantar OAuth en nuestro sistema.

Capítulo 4

Contexto y Solución

4.1. Escenario inicial

El objetivo de este proyecto era realizar un modelo de aplicación genérico que pudiera utilizar cualquier organización adscrita a RedIRIS y que permitiera acceder a unos recursos determinados, pero intentando evitar la situación de tener que generar y almacenar nuevos usuarios y contraseñas. Además de esta aplicación genérica, era necesario crear una aplicación que controlara el acceso a los recursos en cuestión.

Para cumplir estos objetivos se eligió OAuth como tecnología básica del proyecto. Esto se debe a que OAuth aporta varias características que nuestro sistema requería:

- Protege el acceso a unos recursos determinados.
- Utiliza unas credenciales que permiten el acceso a los recursos por un periodo de tiempo concreto.
- Las credenciales pueden ser revocadas.
- La información intercambiada se puede transmitir de forma segura.

- No es necesario almacenar usuario y contraseña en la aplicación

Además de estos aspectos, había uno especialmente relevante que nos hizo decidirnos por OAuth: la necesidad de hacer una aplicación cliente cuya implementación y despliegue no requiriese un esfuerzo desorbitado a las personas que se encargasen de implantar esta aplicación.

Al implementar la solución con OAuth se vio la necesidad de, además de implementar las dos aplicaciones anteriores, desarrollar una aplicación de registro, por lo que los objetivos básicos del proyecto se vieron aumentados al tener que incluir este requisito.

4.2. Elección de implementación

Debido a que la implementación de los servicios de identidad digital de RedIRIS son, en su mayoría, desarrollados en código PHP, se decidió implementar en este lenguaje la solución.

En este lenguaje se pueden encontrar las siguientes librerías para OAuth:

Librería	Consumidor	Proveedor	Version	¿Genérica?	Documentación
Extensión PHP escrita por John Jawed[14]	SI	NO	1.0a	SI	SI
Basic PHP Library por Andy Smith[15]	SI	SI	1.0	SI	SI
Simple Oauth Library por Cal Henderson[16]	SI	NO	1.0a	SI	NO
HTTP_OAuth pear package[17]	SI	SI	1.0a	SI	SI
Componente OAuth para CakePHP[19]	SI	NO	1.0a	NO	Escasa
Componente OAuth para Elgg[20]	SI	NO	1.0	NO	NO
Componente OAuth para Zend[21]	SI	SI	1.0a	NO	SI

Para decidirnos por una de ellas tuvimos en cuenta si era genérica o específica para una aplicación concreta o si implementaba las nuevas versiones de OAuth y funcionalidades tanto para Cliente como para Servidor.

En nuestro caso, la que más se ajustaba a nuestro problema fue el paquete PEAR[13] HTTP_OAuth, el cual, además de estas funciones, aporta una documentación y comunidad adecuadas en las cuales apoyarnos para implementar nuestra solución.

4.3. Solución

Para solucionar la problemática expuesta, se decidió implementar tres sistemas independientes:

- Cliente: Aplicación que permite acceder a los recursos almacenados en el servidor.
- Servidor: Provee un sistema de acceso a los recursos que controla.
- Registro: Aplicación que permite registrar las aplicaciones clientes para así establecer sus credenciales.

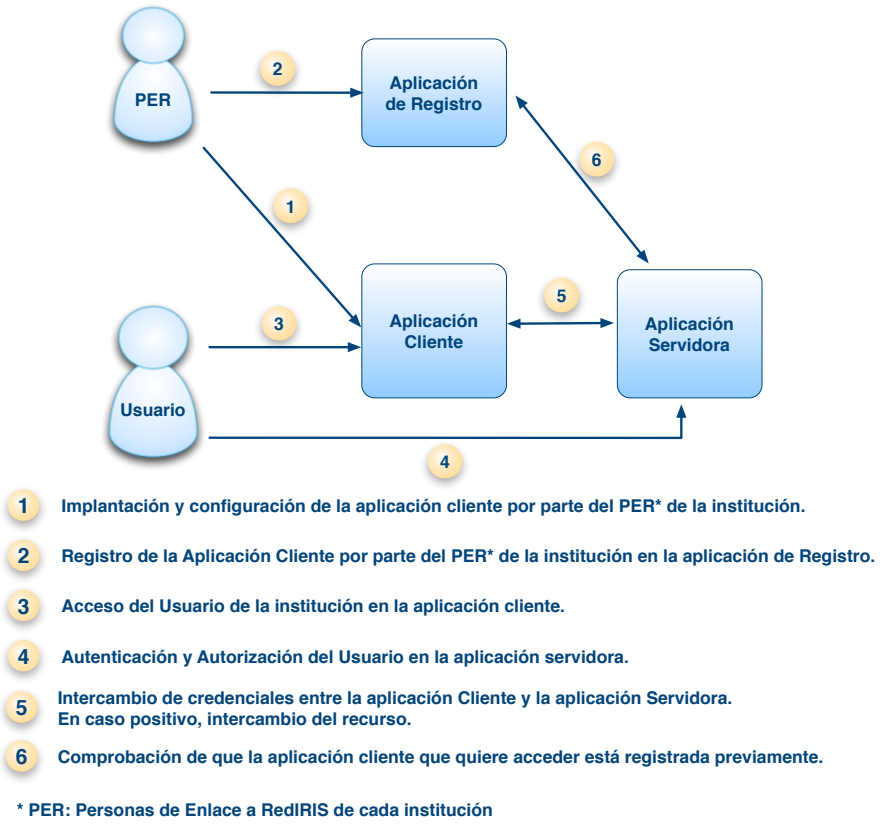


Figura 4.1: Procedimiento de uso de la solución - Gráfico y Leyenda

Para una mejor compresión de la solución llevada a cabo se explicarán cada uno de estos sistemas en secciones diferentes expuestas a continuación.

4.3.1. Registro

Este elemento es la aplicación de registro de las credenciales del cliente. Ésta era necesaria por la propia arquitectura de OAuth, que obliga a tener unas Credenciales de aplicación Cliente que poder intercambiar con el Servidor para garantizar su integridad.

Figura 4.2: Página de inicio de la Aplicación de Registro

Tras un estudio de diseño de las necesidades y requerimientos de los usuarios de esta aplicación, se estableció lo siguiente:

- La aplicación de registro debe tener tres elementos (figura 4.2): Interfaz de registro (figura

4.3), interfaz de administración (figura 4.4) e interfaz de visualización de la petición de registro (figura 4.5).

Figura 4.3: Página de registro de aplicaciones Cliente

Figura 4.4: Interfaz de administración

- Al interfaz de registro solo pueden acceder los denominados PERs, es decir, Personas de Enlace a RedIRIS de cada institución, los cuales son las personas con privilegios suficiente como para dar de alta una aplicación Cliente de OAuth en su institución.

- Las aplicaciones se registran con un nombre que contine la institución a la que pertenece el

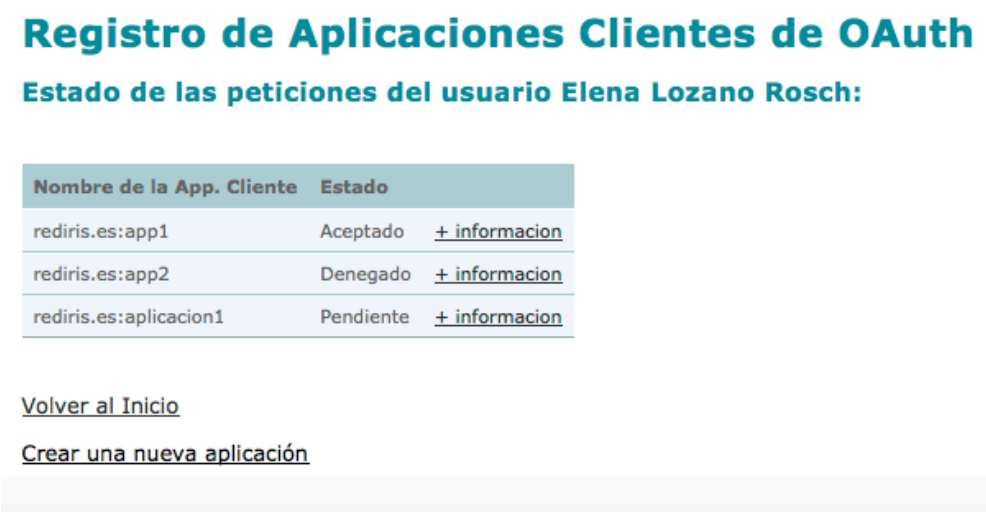


Figura 4.5: Interfaz de seguimiento de la petición de registro

PER y el nombre elegido. Un ejemplo de una aplicación que se ha registrado con el nombre 'aplicacion1' de la institución RedIRIS será: `rediris.es:aplicacion1`.

- Las aceptaciones de petición de registro se realizan de forma manual, ya que desde RedIRIS se debe comprobar quién y desde qué organización está registrando aplicaciones Clientes de OAuth (figura 4.6).

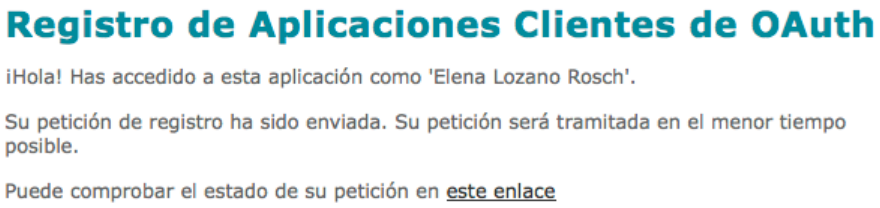


Figura 4.6: Aceptación de la petición de registro

- Las peticiones de registro pueden ser aceptadas (figura 4.7), denegadas (figura 4.8) o estar pendientes (figura 4.9). En la página de seguimiento de las peticiones, solo podrá visualizarse la información de configuración de OAuth en el caso de estar aceptadas.

Figura 4.7: Visualización de una petición de registro aceptada

Figura 4.8: Visualización de una petición de registro denegada

- Las peticiones podrán ser aceptadas, denegadas y eliminadas por una persona que formará parte del personal de RedIRIS.
- El interfaz permite revocar el alta de una aplicación y suprimir su registro de la base de datos permanentemente.
- El secreto que genera la aplicación para un cliente determinado se obtiene mediante un metodo resumen de una cadena fija determinada por RedIRIS, el identificador de tipo 'institucion:nombre', el identificador del solicitante aportado en la aserción obtenida tras la autenticación en el Sistema de Identidad de Rediris (SIR), el cual es opaco y no permite asociarlo al solicitante en cuestión.

El modelo de datos que se tiene en esta aplicación consiste en dos tablas con los parámetros:

- Tabla Peticiones:

Figura 4.9: Visualización de una petición de registro pendiente

- Identificador del solicitante
- Identificador de la aplicación
- Nombre de la aplicación
- Institución a la que pertenece
- Estado de la petición (aceptado, denegado, pendiente)

- Tabla Claves:

  - Identificador de la petición
  - Clave

Este modelo se ha implementado en una base de datos ligera (SQLite[22]) debido a la simplicidad del modelo de datos a almacenar y los bajos requerimientos técnicos que ésta tiene.

4.3.2. Cliente

Esta aplicación implementa una estructura de aplicación cliente de OAuth (comentada en capítulos anteriores), pero que básicamente consiste en los siguientes pasos:

- Inicio de la aplicación.
- Intercambio de credenciales del cliente.

- Redirección a la página de autenticación del servidor.
- Intercambio de las credenciales temporales y de la petición.
- Acceso a los recursos.

La aplicación cliente que se da como código de ejemplo está hecha de tal forma que una persona que no esté familiarizada con los conocimientos técnicos en identidad digital pueda implantarla en su institución. Para ello se ha realizado una clase que encapsule los procedimientos generales. En el *Apéndice 1: Manual de Implantación* se especifica la forma de desplegar el sistema de forma detallada y en el *Apéndice 2: Código de Aplicación Cliente* puede verse el código de esta clase.

En la actualidad, solo es posible elegir a las listas de correo a las que está suscrita una persona como ámbito de búsqueda ('*scope*'), pero se están planteando nuevos ámbitos. En un futuro no será problema añadirle nuevos alcances debido a la forma modular de la aplicación.

4.3.3. Servidor

La aplicación servidora es la parte más compleja de la solución y constituye la base de todo el proyecto. La estructura diseñada para ésta se basa en los puntos de acceso necesarios para el funcionamiento de OAuth y es la siguiente:

- TCREndpoint.php - *Temporal Credential Request Endpoint* - Página que comprueba las credenciales del cliente y asigna unas credenciales temporales.
- ROAEndpoint.php - *Resource Owner Authorization Endpoint* - Punto donde el Usuario es redirigido y permite o deniega el acceso a un recurso. Este punto está asociado al Servicio

de Identidad de RedIRIS (SIR)[33] mediante la instalación de un *phpPoA*[34][1] en el servidor. El SIR ofrece un hub de interconexión entre los servicios de identidad de las instituciones afiliadas y proveedores de servicio, a nivel nacional e internacional. Está basado en tecnologías de federación de identidades, de manera que:

- Los usuarios se identifican en los servidores locales de la institución, utilizando el procedimiento de identificación definido por la misma y sin que las credenciales sean expuestas fuera del dominio local.
- Los administradores de los servicios de identidad de las instituciones tienen control total sobre los procedimientos de identificación y los atributos asociados a cada usuario.
- Cada institución aplica de manera autónoma los mecanismos de control que considere necesarios para ofrecer a sus usuarios la posibilidad de decidir de manera informada acerca de la información personal susceptibe de ser transmitida.
- RedIRIS proporciona una conexión segura, fiable y conforme a estándares entre las instituciones y los proveedores de servicio.
- Los proveedores de servicio aplican de manera autónoma los mecanismos de control de acceso a los recursos que tienen bajo su control. Es importante tener en cuenta que los proveedores de servicio pueden ser tanto entidades ajenas al entorno académico (comerciales, gubernamentales, etc.) como dentro del mismo, ya pertenezcan a RedIRIS u otra red académica nacional (NREN).

La versión actual del SIR utiliza el protocolo de federación PAPI v.1 y soporta distintos protocolos de salida como PAPI v.1, SAML 1.1 y SAML 2 u OpenID, entre otros.

[1] phpPOA es una implementación en lenguaje php de un PoA PAPI. Un PoA es un Punto de Acceso (*Point of Access*) que controla quién puede entrar en un conjunto determinado de localizaciones web.

- ROAEndpointLogic.php - Una vez aceptado o denegado, realiza los pasos necesarios para que se de una respuesta negativa o una positiva, junto con las credenciales temporales autorizadas.
- TREndpoint.php - *Token Request Endpoint* - Punto que realiza el intercambio de unas credenciales temporales autorizadas por unas credenciales de token válidas para el acceso al recurso.
- PREndpoint.php - *Protected Resource Endpoint* - Dadas unas credenciales de token válidas, se permite el acceso al recurso.
- Dentro de la carpeta Utils
  - bdUtils.php Funciones que realizan conexión con una base de datos ligera, (SQLite) y permite el almacenamiento de las credenciales de forma temporal.
  - criptoUtils.php - Funciones que generan los elementos aleatorios de las credenciales.
  - ldapUtils.php - Funciones necesarias para la inicialización de la conexión con el LDAP del cual se obtienen los datos de los recursos.
  - papiUtils.php - Funciones necesarias para que el usuario pueda autenticarse en el servidor.

Los datos de las credenciales se almacenan en una base de datos ligera (SQLite[22]) debido a la simplicidad del modelo de datos a almacenar y los bajos requerimientos técnicos que ésta tiene. Éstos se estructuran en dos tablas:

- Tabla de Credenciales Temporales:
  - Identificador de la credencial
  - Identificador de la aplicación cliente
  - Secreto de la aplicación cliente

  - Si está autorizado o no por el usuario
  - Si se ha intercambiado ya o no por una credencial de token.
  - Timestamp en el que caducará la petición
  - Alcance de la aplicación. (Alcance de los recursos a los que se quieren acceder)
  - Verificador de la credencial
  - URL de callback a la que se deberá redirigir tras tener la autorización del usuario.

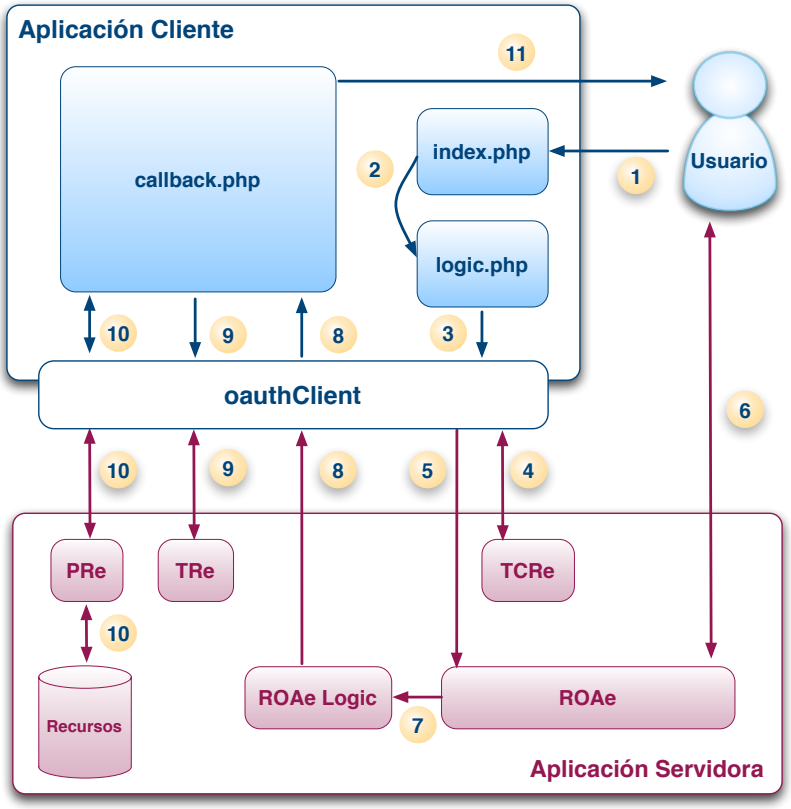
- Tabla de Credenciales de Token:

  - Identificador de la credencial
  - Identificador de la aplicación cliente
  - Secreto de la aplicación cliente
  - Identificador (sPUC) del usuario al que pertenecen los recursos.
  - Timestamp en el que caducará la petición
  - Alcance de la aplicación. (Alcance de los recursos a los que se quieren acceder)
  - Verificador de la credencial

En cuanto a la temporización de las credenciales, se podrá configurar desde el cliente; haciendo que este tiempo determinado se almacene en el campo *timestamp* de las tablas de credenciales. Cuando expira, se eliminan automáticamente de la base de datos.

En la figura 10 podemos observar la interacción de la aplicación Cliente con la aplicación Servidora. Podemos observar que la clase *oauthClient* es la que interactúa directamente con el servidor, actuando como un adaptador para la aplicación Cliente y que simplifica la implantación de la misma para quienes vayan a realizarla.

El código de éstas clases se adjunta en el *Apéndice 3: Código del Servidor*.



1. Entrada del usuario en la aplicación Cliente.
2. La aplicación Cliente redirige a donde se encuentre la lógica de OAuth.
3. Inicialización y configuración de la clase oauthClient.
4. Intercambio de credenciales del Cliente. Comprueba que está registrada previamente. Generación de las credenciales temporales.
5. Redirección del usuario al punto de autenticación y autorización del servidor.
6. Autorización del usuario sobre el acceso a sus recursos.
7. Autorización de credenciales temporales según lo establecido por el usuario.
8. Redirección del Usuario a la aplicación Cliente.
9. Intercambio de credenciales temporales autorizadas válidas por credenciales de token.
10. Petición de los recursos, pasando por la petición de la aplicación Cliente, el punto de entrada del Servidor y el sistema de almacenaje de los recursos en el Servidor.
11. Muestra por pantalla los recursos obtenidos debidamente formateados.

Figura 4.10: Interacción de Aplicación Cliente y Aplicación Servidora

Capítulo 5

Análisis temporal y de costes.

5.1. Estimaciones iniciales

Al comenzar el proyecto, se estableció una temporización aproximada, basándonos en los datos que teníamos inicialmente:

- El proyecto fin de carrera debía ocupar 540 horas como mínimo.
- El inicio del proyecto fue en noviembre de 2009.
- Desde diciembre de 2009, el tiempo de trabajo diario sería de 5 horas, de lunes a viernes.

Con esta información pasamos a dividir en partes el trabajo necesario para realizar el proyecto: Análisis de los requisitos, Documentación, Implementación y Realización de la presentación.

Análisis de los requisitos: Comprenderá las actividades de planificación, organización, investigación de tecnologías y herramientas, reuniones iniciales y de *brainstorming*.

Documentación: Actividades de desarrollo de la documentación escrita, manuales de usuarios, guía de instalación, gráficos y tablas explicativas, etc.

Implementación: Todo lo relacionado con la programación de la solución definida.

Presentación: Comprende el esfuerzo dedicado a la realización de la presentación, preparación de la documentación que tuviera que entregarse, grabación de CDs con el código, etc.

En estos siete meses y medio de trabajo, se llegaron a estimar los siguientes valores para cada una de las actividades:

- Análisis: Ocuparía la primera fase del proyecto, siendo el esfuerzo de mes y medio aproximadamente.
- Implementación: Sería la segunda fase del proyecto, con una duración de dos meses aproximadamente.
- Documentación: Tercera fase del proyecto, de mes y medio aproximadamente.
- Presentación: Última fase, con un esfuerzo de medio mes.

Debido a la realización de anteriores proyectos fin de carrera, sabíamos que estas etapas, aunque iban a estar más o menos repartidas en el tiempo, acabarían por solaparse, por lo que no pusimos fechas muy exactas en la temporización de cada una de las etapas.

5.2. Estimaciones finales

Durante la realización del proyecto, se fue completando una ficha con las horas dedicadas para cada etapa, para así poder comparar los valores estimados con los reales.

Las estimaciones finales obtenidas se han esquematizado en la siguiente tabla:

Etapa	Tiempo Estimado (en horas)	Tiempo Invertido (en horas)
Análisis	144	125
Documentación	144	200
Implementación	180	180
Presentación	72	60
Horas Totales	540	565

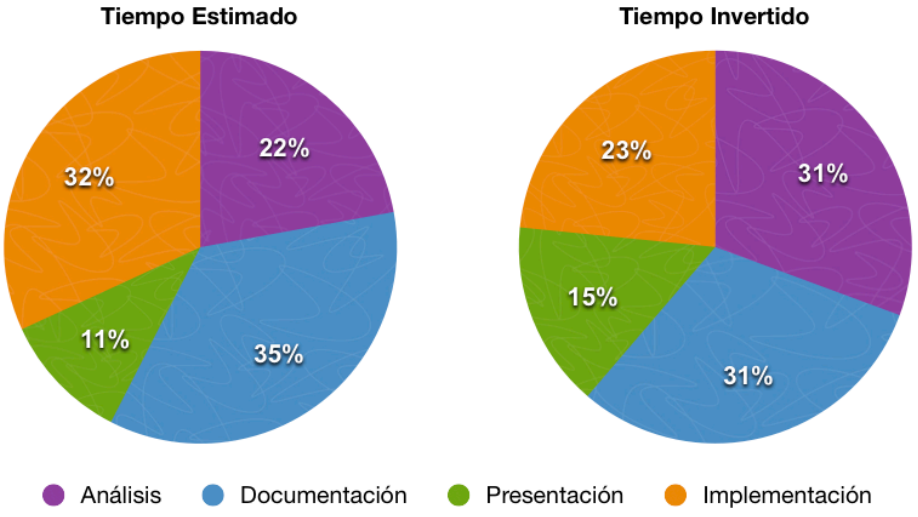


Figura 5.1: Gráficas comparativas de Tiempo Estimado y Tiempo Invertido

Como se puede observar en la gráfica 5.1, se estimó una duración menor de la documentación de la que fue realmente, con una diferencia de un cuatro por ciento. Tras un análisis pormenorizado del tiempo invertido en documentación, se encontró el motivo de esta diferencia, la necesidad de revisar la documentación a mitad del proyecto debido a cambios en el protocolo y la realización de la documentación en inglés.

Cabe destacar que se estimó un tiempo de implementación mayor del que requirió finalmente. Éste se vio reducido gracias a que aplicamos más esfuerzo al análisis y diseño de la solución, por lo que la implementación fue una tarea de aplicación directa.

En cuanto a la etapa de presentación, tomó más tiempo de lo estimado debido a la necesidad de hacer diagramas interactivos y a las horas de ensayo requeridas.

Aunque la definición de los objetivos iniciales se ven cumplidos en las 565 horas invertidas finalmente, durante el transcurso del proyecto se establecieron nuevos objetivos que pretenden que se realicen nuevas extensiones del proyecto y se implanten en otros casos de usos, por lo que esas horas de extensión también fueron registradas para el análisis temporal del proyecto.

En la figura 5.2 y en la tabla adjunta se puede observar el tiempo invertido en estas extensiones, en comparación con el total de horas invertidas.

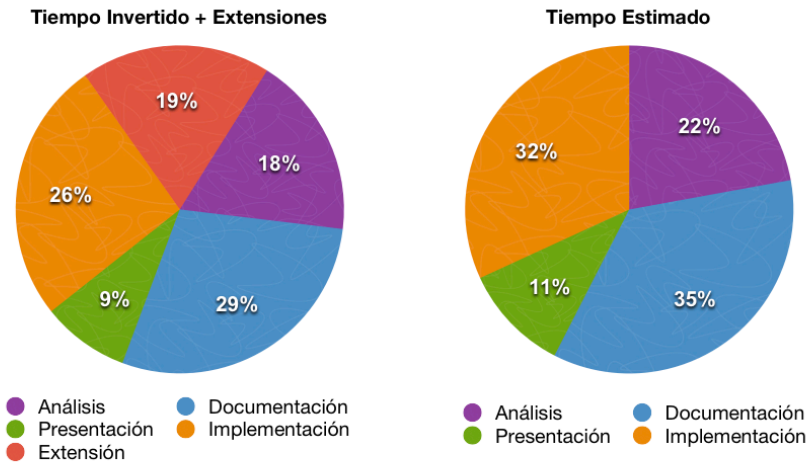


Figura 5.2: Gráficas comparativas de Tiempo Estimado y Tiempo Invertido con extensiones.

Etapa	Tiempo Estimado (en horas)	Tiempo Invertido (en horas)
Análisis	144	125
Documentación	144	200
Implementación	180	180
Presentación	72	60
Extensión	-	130
Horas Totales	540	685

Por último, en este análisis de costes temporales, mostramos la gráfica de ocupación de cada etapa dependiendo del momento en el que se realizaron (figura 5.3). Como podemos ver, aunque cada etapa está restringida a un momento determinado del desarrollo, se solapan muchas de ellas en el tiempo, tal y como habíamos previsto en las estimaciones iniciales.

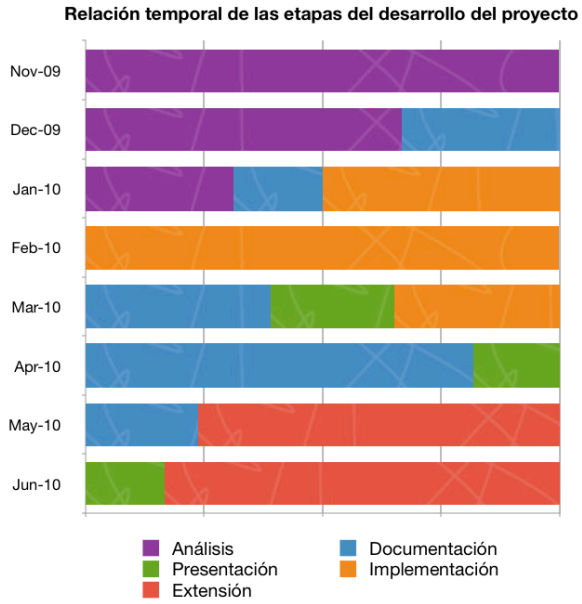


Figura 5.3: Relación Temporal de la ejecución de las etapas del desarrollo del proyecto

Capítulo 6

Conclusiones

Uno de los aspectos observados al realizar este proyecto ha sido la necesidad de documentarse e investigar antes de realizar ninguna implementación. Si bien es importante llevar a la práctica los conceptos asimilados, más importante es realizar un estudio exhaustivo de las tecnologías que se van a utilizar, para así poder tomar las decisiones correctas cuando sea necesario.

Además de esto se ha visto no sólo la necesidad de documentarse, si no la actualización permanente durante el transcurso de un proyecto, especialmente cuando las tecnologías que están utilizándose son tan novedosas.

Éste fue el caso ocurrido durante el desarrollo del proyecto, cuando modificaron el borrador del protocolo OAuth y posteriormente la versión del mismo, añadiendo nuevas funcionalidades y cambiando la nomenclatura utilizada en él.

Gracias a la continua actualización y revisión de la información utilizada al realizar este proyecto y a la lectura de la lista de correo del grupo de trabajo de OAuth, se pudo reestructurar la arquitectura en un estadio poco avanzado del desarrollo, ocasionando una baja repercusión en el proyecto.

Otro aspecto interesante del proyecto es la posibilidad que tecnologías innovadoras en identidad digital como OAuth ofrecen a un público menos especialista. Gracias a la simplicidad a la hora de implantar estas soluciones en el lado del cliente es posible que aplicaciones de bajo presupuesto puedan incorporar sistemas de acceso a recursos protegidos que con otro tipo de tecnología implicarían un esfuerzo mayor a los desarrolladores de las mismas.

Este motivo es uno de los cuales han propiciado el auge de una nueva generación de herramientas en identidad digital, la cual pretende implementar sistemas de autenticación y autorización basados en tecnologías que garanticen la seguridad de las comunicaciones, pero que no resulten excesivamente complejas ni ocupen muchos recursos.

Esta tendencia se ve reflejada en las investigaciones y desarrollos que se están realizando actualmente, como por ejemplo la realización de servicios web basados en tecnología REST[1] [31] frente a implementaciones mucho más complejas y pesadas como SOAP[2] [30].

Como ya comenté en la introducción, este servicio utiliza como caso base de implantación la herramienta de Listas de distribución de RedIRIS[3] permitiendo a los usuarios poder ver la información correspondiente a las listas que están suscritos; pero pretende ir más allá; ya que tal y como se ha realizado la arquitectura es fácilmente extensible para poder añadir la posibilidad de acceder a recursos de otra índole.

[1] Mecanismo que permite aprovechar las capacidades predefinidas de los protocolos que soportan las comunicaciones, como puede ser HTTP.

[2] Protocolo de comunicación basado en mensajes con formato XML[32]

[3] `(http://www.rediris.es/list`

Actualmente se están estudiando las posibilidades que se pueden obtener de esta aplicación en los siguientes proyectos de RedIRIS:

- Panel de Servicios: Panel en desarrollo que pretende ser el punto de entrada donde los PERs (Personas de Enlace con RedIRIS) pueden ver el estado de los servicios que ofrece RedIRIS, gestionar en cuáles de ellos participa, acceder a estadísticas y otros datos de los mismos. La arquitectura para integrar este panel con OAuth se basa en la de OpenSocial[37], una plataforma que integra contenidos de distintos servicios relacionados con redes sociales.

- BSCW [36] (*Basic Support for Cooperative Working*): plataforma de trabajo colaborativo que permite la colaboración en grupo de manera sencilla, con funcionalidades como integración con calendarios, creación de blogs, listas de correo, gestión de versiones de documentos, repositorios, etc. OAuth se utilizaría aquí como forma de poder acceder y ver el contenido de esta plataforma desde otros lugares, por ejemplo, el panel de servicios o cualquier sitio web que actuara como cliente de OAuth.

Además de este incremento de funcionalidad que se encuentra en estudio, el desarrollo de este servicio ha abierto otras puertas a la investigación y desarrollo.

Durante la realización del proyecto se continuó investigando acerca de OAuth y se encontró una extensión del protocolo que está definiéndose actualmente, la cual se denomina OAuth WRAP[4].

Esta extensión nació para intentar simplificar a OAuth, eliminando sus procesos de firma y reemplazándolas por tokens de corta duración sobre SSL/TLS[7]. No pretendía sustituir a OAuth, pero sí añadir nuevas características que completan y desarrollan elementos de los cuales carece el protocolo en su versión original.

[4] *OAuth Web Resource Authorization Profiles*[5]

Posteriormente a la publicación de OAuth WRAP, se vio que esta extensión era muy ventajosa, por lo que ha sido integrada en la segunda versión de OAuth[38], la cual se encuentra actualmente en *draft* en la IETF[5].

OAuth WRAP y por consiguiente OAuth versión 2, permite la delegación de la autorización de recursos protegidos a una o más autoridades de confianza. Los Clientes que desean acceder a un Recurso Protegido primero obtienen autorización de un Servidor de Autorización, enviando para ello unas credenciales y perfiles determinados.

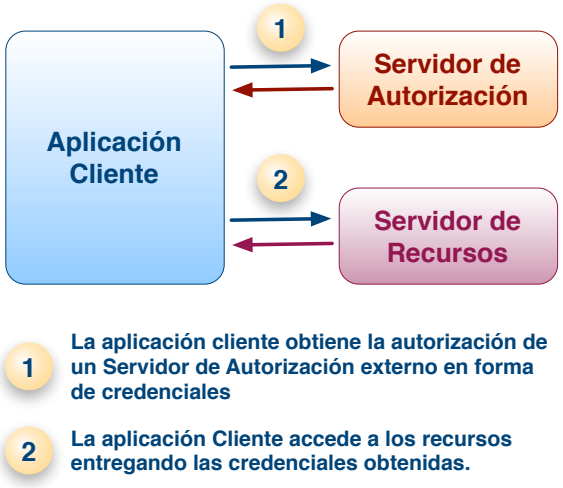


Figura 6.1: Flujo de Autorización de OAuth WRAP

Una vez autorizado, el Cliente obtiene unas credenciales firmadas digitalmente que podrán ser reemitidas en caso de ser necesario y que permitirá acceder a los recursos protegidos.

[5] *Internet Engineering Task Force*[39]

Actualmente nos encontramos trabajando en el desarrollo de un software de autenticación y autorización basado en uno de los flujos establecidos en OAuth2 adquiridos de OAuth WRAP, el de aserción. Éste consiste en que la aplicación Cliente sea autorizada con una aserción, ya sea SAML[6] u otro tipo de aserción que reconozca el Servidor de Autorización. El cliente presenta esta aserción al servidor y la intercambiará por las credenciales de acceso correspondientes.

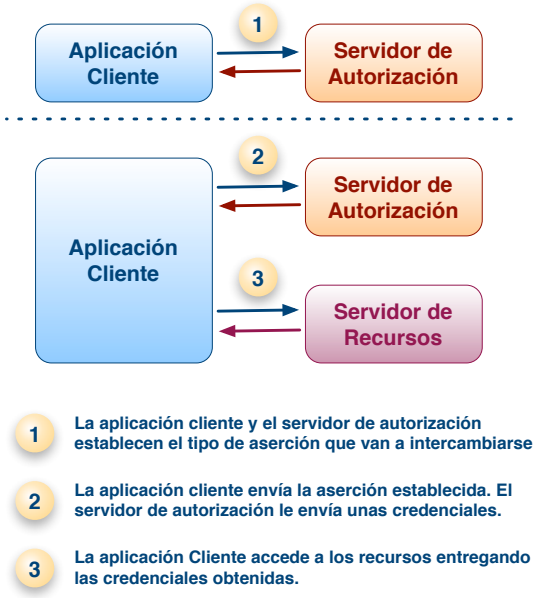


Figura 6.2: Flujo de Autorización de OAuth WRAP - Perfil de Aserción

[6] *Security Assertion Markup Language*[28]

Bibliografía

[1] **Página Oficial de OAuth** http://oauth.net

[2] **Blog oficial de OAuth** http://hueniverse.com/oauth/

[3] Borrador de IETF - **The OAuth Protocol: Web Delegation** http://tools.ietf.org/wg/oauth/draft-ietf-oauth-web-delegation/

[4] Borrador de IETF - **The OAuth 1.0 Protocol** http://tools.ietf.org/id/draft-hammer-oauth-10.txt

[5] Borrador de IETF - **OAuth Web Resource Authorization Profiles** http://tools.ietf.org/id/draft-hardt-oauth-01.txt

[6] Borrador de IETF - **Using OAuth for Recursive Delegation** http://tools.ietf.org/id/draft-vrancken-oauth-redelegation-00.txt

[7] Referencia sobre **TLS/SSL** http://www.ietf.org/dyn/wg/charter/tls-charter.html

[8] **Referencia del Algoritmo SHA1** http://www.faqs.org/rfcs/rfc3174.html

[9] **RFC del protocolo HTTP** www.ietf.org/rfc/rfc2616.txt

[10] **RFC de la sintaxis de los *Uniform Resource Identifier* (URI)** www.ietf.org/rfc/rfc3986.txt

[11] **Definiciones de códigos de estado del protocolo HTTP** http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

[12] **Especificación de HTML 4.0** http://www.w3.org/TR/1998/REC-html40-19980424/

[13] **Sistema de paquetes PHP PEAR** http://pear.php.net/

[14] **Extensión PHP escrita por John Jawed** http://php.net/oauth

[15] **Basic PHP Library por Andy Smith** http://oauth.googlecode.com/svn/code/php/

[16] **Simple Oauth Library por Cal Henderson** https://svn.iamcal.com/public/php/lib_oauth/lib_oauth.php

[17] **HTTP_OAuth pear package** http://pear.php.net/package/HTTP_OAuth

[18] **HTTP_Request2 pear package** http://pear.php.net/package/HTTP_Request2

[19] **Componente OAuth para CakePHP** http://cakebaker.42dh.com/downloads/oauth-component-for-cakephp/

[20] **Componente OAuth para Elgg** http://community.elgg.org/pg/plugins/jricher/read/230574/oauth

[21] **Componente OAuth para Zend** http://framework.zend.com/wiki/pages/viewpage.action?pageId=37957

[22] **SQLite** http://www.php.net/manual/en/book.sqlite.php

[23] **Página Oficial de PAPI** http://papi.rediris.es/

[24] **Página Oficial de Shibboleth** http://shibboleth.internet2.edu/

[25] **Página Oficial de OpenID** http://openid.net/

[26] **Página Oficial de Sun Access Manager** http://www.sun.com/software/products/opensso_enterprise/index.xml

[27] **Referencia de PHP** http://php.net

[28] **Página oficial de SAML** http://saml.xml.org/

[29] **Página oficial de JSON** http://www.json.org/

[30] **Página oficial de SOAP** http://www.w3.org/TR/soap/

[31] **Disertación original sobre REST** - *Roy Thomas Fielding* http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm#fig_5_8

[32] **Página oficial de XML** http://www.w3.org/TR/2006/REC-xml-20060816/

[33] **Página oficial del Servicio de Identidad de RedIRIS** http://www.rediris.es/sir/index.html.es

[34] **Página del proyecto phpPoA en la Forja de RedIRIS** https://forja.rediris.es/projects/phppoa/

[35] **Página de la Forja de RedIRIS** https://forja.rediris.es/projects/phppoa/

[36] **Página oficial del servicio BSCW de RedIRIS** http://bscw.rediris.es/

[37] **Página oficial de OpenSocial** http://wiki.opensocial.org/

[38] **Borrador de OAuth versión 2** http://tools.ietf.org/html/draft-ietf-oauth-v2-05

[39] **Página oficial de la IETF** http://www.ietf.org/

Apéndice A

Manual de Implantación

A.1. Descripción

OAuth (*Open Authorization; Autorización abierta*) es un estándar abierto que permite incluir seguridad en la autenticación para aplicaciones web y de escritorio.

A.1.1. Conceptos Básicos

A continuación se definen los elementos básicos que serán necesarios para la comprensión del protocolo OAuth.

Servidor (*Server*) Es el servidor HTTP donde se encuentran los recursos que están protegidos mediante OAuth y se encarga de denegar o permitir el acceso a los mismos. Es capaz de responder a peticiones de OAuth autenticadas.

Propietario del Recurso (*Resource Owner*) Entidad capaz de acceder y controlar recursos protegidos utilizando unas credenciales determinadas al autenticarse en el Servidor.

Cliente (*Client*) Aplicación web que utiliza OAuth para acceder al Servidor en nombre del Propietario del Recurso o en su propio nombre. Es un cliente HTTP que es capaz de hacer peticiones de OAuth autenticadas.

Recurso Protegido (*Protected Resource(s)*) Datos del Propietario del Recurso a los que desea acceder el Cliente.

Credenciales (*Credentials*) Son un par de identificador único y secreto compartido. Existen tres tipos de credenciales distintas: del Cliente, Temporales y de Token.

Credenciales del Cliente (*Client Credentials*) Identifican y autentican al cliente que hace la petición.

- Identificador del Cliente (*Client Identifier*) Valor que identifica al Cliente en el Servidor.

- Secreto compartido del Cliente (*Client Shared-Secret*)] Secreto que utiliza el Cliente y que lo acredita en el Servidor.

Credenciales Temporales (*Temporary Credentials*) Identifican y autentican a la petición de autorización. Son utilizadas por el Cliente para obtener autorización del Propietario del Recurso. Será una cadena alfanumérica única relacionada con un secreto del Cliente correspondiente.

Credenciales del Token (*Token Credentials*) Identifican y autentican al acceso concedido. Permite acceder a los Recursos Protegidos y es el paso posterior a la obtención de las credenciales temporales.

A.1.2. Flujo de Autorización de OAuth

A continuación se enumeran los pasos que realiza internamente el protocolo OAuth para funcionar de forma correcta.

1. **Obtención de las Credenciales Temporales**
   El objetivo de este paso es que el Cliente obtenga unas Credenciales Temporales. Para ello, la aplicación Cliente envía una petición a la dirección de destino dada por el Servidor para estos casos, denominada *Temporary Credential Request Endpoint (TCREndpoint)*.
   En ella se envían las Credenciales del Cliente, para que el Servidor pueda contrastarlas con las que tiene almacenadas y verificar la autoría de la petición.
   Una vez enviada la petición, el Servidor la recibe y verifica la firma y las Credenciales del Cliente que van en ella.
   En caso de ser verificada correctamente, el Servidor genera unas Credenciales Temporales que representarán al Propietario del Recurso en el Servidor para esa conexión concreta, difiriendo del nombre de usuario y contraseña originales.

2. **Autorización del Propietario del Recurso y validación de la petición**
   El objetivo de este segundo paso es que el Propietario del Recurso autorice las Credenciales Temporales.
   Para ello, el Cliente redirige al Propietario del Recurso a la URL que el Servidor tiene establecida para el tratamiento de las autorizaciones, denominada *Resource Owner*

Authorization Endpoint (ROAEndpoint).
Una vez redirigido el Propietario del Recurso en su navegador web a la aplicación servidora, el usuario aceptará o denegará el acceso a los recursos. La decisión tomada por el usuario se hará mediante la redirección del Propietario del Recurso de nuevo al Cliente (a la dirección de *callback* configurada).

3. **Intercambio de las Credenciales Temporales por las Credenciales del Token**
   En este paso, se intercambian las Credenciales Temporales por las Credenciales del Token. Para ello, el Cliente envía una petición a la URL del Servidor especificada para ello, denominada *Token Request Endpoint (TREndpoint)*.

4. **Acceso a los Recursos Protegidos**
   Como último paso para que el Cliente acceda a los Recursos Protegidos en nombre del Propietario de los Recursos, éste deberá enviar una petición a la dirección del servidor establecida para ello, denominada *Protected Resource Endpoint (PREndpoint)*

A.2. Requisitos

- Tener instalada alguna versión de PHP superior o igual a la 5.1.2.
- Tener instalado el sistema de paquetes PHP PEAR[13] en una versión igual o superior a la 1.4.0.
- Instalar el paquete PEAR HTTP_OAuth[17] con sus dependencias:
  - Paquete PEAR HTTP_Request2[18], en su versión 0.5.1 o superior.
  - Módulos de php: date y hash.

A.3. Implantación

1. Darse de alta en el sistema de registro de clientes OAuth.
2. Anotar los parámetros de acceso dados al realizar el registro de forma satisfactoria. Éstos serán:

   Identificador del Cliente Nombre identificador introducido al dar de alta la aplicación.

   Secreto del Cliente Clave alfanumérica que se obtiene al dar de alta la aplicación.

Callback Punto al que él servidor devolverá al usuario una vez aceptada o rechazada la autorización. En el ejemplo dado es *callback.php*.

time Tiempo que permanecerá activa la autorización.

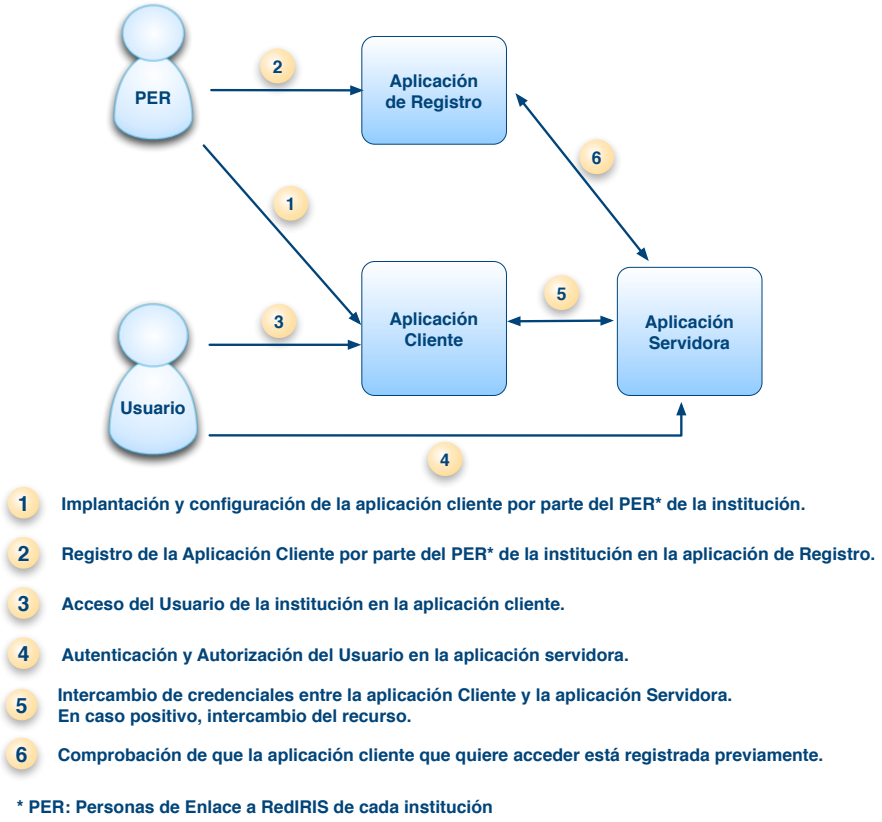


Figura A.1: Implantación - Relación de aplicaciones.

3. Desplegar la aplicación web de prueba en un servidor web con los requisitos especificados anteriormente configurados.

4. Modificar la página de índice dada en el código de ejemplo con el estilo que desee su organización.

5. Introducir en la página de índice algún sistema de login en caso de desearlo.

6. Modificar la página de lógica de la aplicación con los parámetros de acceso anotados en el segundo paso.

7. Modificar la página de callback para mostrar de forma deseada el resultado devuelto por el servidor.

A.4. Despliegue

En el caso de no utilizar la implementación de ejemplo dada, los pasos principales que serán necesarios para proveer acceso a los recursos protegidos mediante OAuth son los siguientes:

1. Instanciación de la clase *oauthClient*.
   ```
   $oauth = new oauthClient();
   ```

2. Configuración de los parámetros específicos de la aplicación:
   ```
   $oauth->configure($clientid, $clientSecret, $callback);
   ```

3. **El Cliente obtiene unas credenciales temporales del Servidor**
   Este paso es el que se realiza mediante la llamada a la función de la clase *oauthClient*.
   ```
   $oauth->preprocess();
   ```

4. **El Cliente pide autorización del propietario del recurso** Con estas credenciales temporales, el cliente redirige al usuario a la url del servidor que permitirá que el usuario autorice o deniegue el acceso al recurso.
   ```
   $oauth->redirect();
   ```

5. **El Propietario del Recurso valida la petición y da autorización para acceder al mismo**
   Una vez redirigido a la dirección de *callback*, el usuario deniega o acepta el acceso de la aplicación cliente al recurso. La respuesta del servidor devolverá unas credenciales temporales mediante perámetros GET: *Token*, *verificador* y *token secret*.

6. **El Cliente intercambia las credenciales temporales por las del token y accede al recurso protegido**
   Este proceso consiste en utilizar los parámetros enviados en la respuesta e intercambiarlos por unas credenciales de token autorizadas.
   ```
   $token = $_GET['oauth_token'];
   $verifier = $_GET['oauth_verifier'];
   $secret = $_GET['oauth_token_secret'];
   $oauth->postprocess($token, $verifier, $secret);
   ```

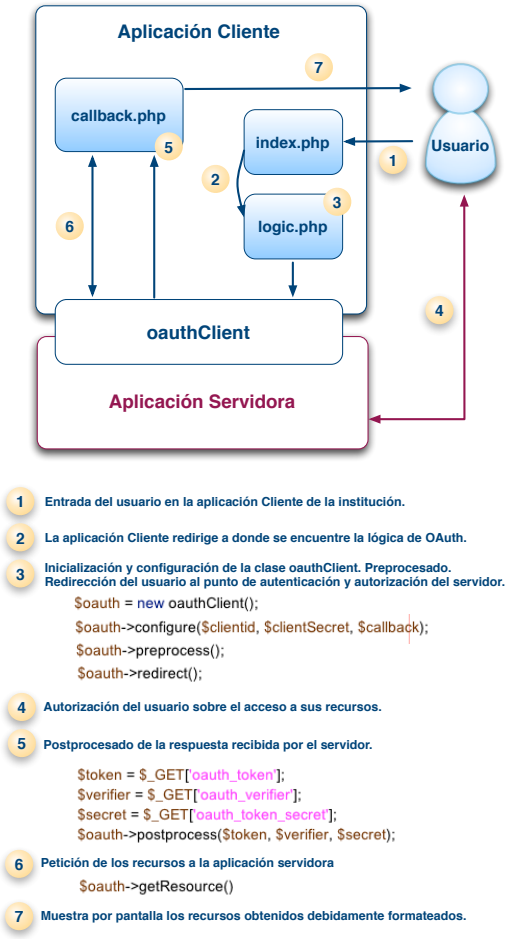


Figura A.2: Arquitectura de la aplicación cliente - Despliegue

7. **Acceso al recurso protegido:**
   `$oauth->getResource()`

8. **Procesado de la respuesta**
   La respuesta devuelta por el servidor es una respuesta JSON que podrá ser tratada de la forma que se desee. En el código de ejemplo se ofrecen algunas funciones de ejemplo.

A.5. Procedimiento a seguir por el Usuario

El procedimiento de uso para poder acceder a los recursos relativos a listas de correos de rediris desde su aplicación es el siguiente:

1. Introduce en el navegador la dirección de la aplicación Cliente de OAuth de tu institución. En esta página inicial, la institución podrá tener algún tipo de autenticación propia. (Figura A.3)

Figura A.3: Autorización del Usuario

2. Autenticación en el sistema de RedIRIS. Este sistema será el Servicio de Identidad de RedIRIS (SIR), el cual ofrece un hub de interconexión entre los servicios de identidad de las instituciones afiliadas y proveedores de servicio, a nivel nacional e internacional.

   - El usuario seleccionará su institución en la lista ofrecida.(Figura A.4)

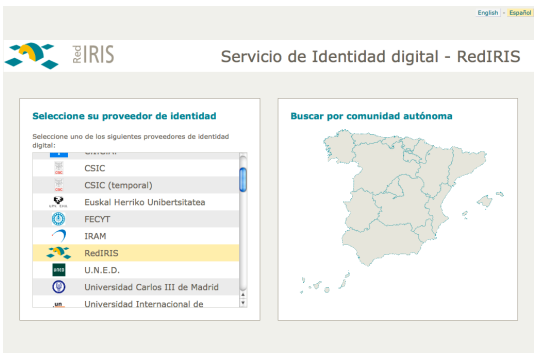


Figura A.4: Autenticación en el sistema de RedIRIS - Selección de institución.

   - El usuario aceptará la institución.(Figura A.5)
   - El usuario introduce sus credenciales. (Figura A.6)

Figura A.5: Autenticación en el sistema de RedIRIS - Aceptar institución.

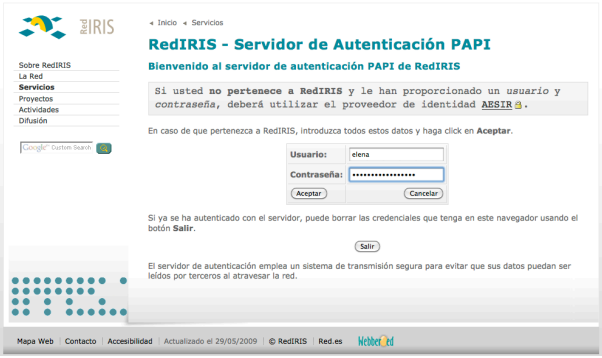


Figura A.6: Autenticación en el sistema de RedIRIS - Introducir Credenciales

3. Una vez introducida la autenticación, el usuario podrá aceptar o denegar el acceso a los recursos por parte de la aplicación cliente.(Figura A.7)

Figura A.7: Autorización del Usuario

4. En el caso de haberse aceptado el acceso, el sistema mostrará la información correspondiente al usuario. (Figura A.8)

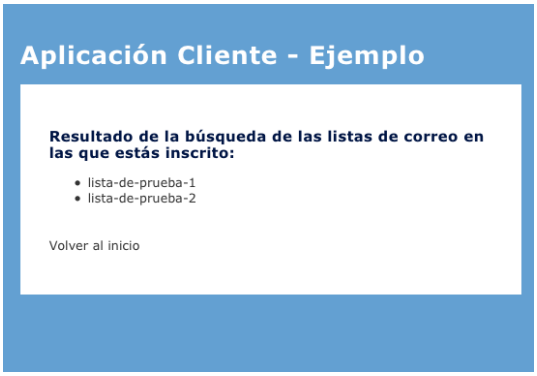


Figura A.8: Resultado del tratamiento de la información devuelta

Apéndice B

Código de Aplicación Cliente

```php
<?php
require_once 'HTTP/OAuth/Consumer.php';
class oauthClient {
    protected $clientId; // Client ID previously registered on rediris.es
    protected $clientSecret; // Client Secret previously provided by the
                        //server in rediris.es
    protected $tcrEndpoint;  //Temporary Credential Request Endpoint
    protected $roaEndpoint; //Resource Owner Authorization Endpoint
    protected $trEndpoint; //Token  Request Endpoint
    protected $prEndpoint; //Protected Resource  Request Endpoint
    protected $method; //Method to send the request: get or post
    protected $scope; //URL where the Resources are accesible
    protected $callback; //Callback URL of your client where the Server
                        // come back after authorization
    protected $httpRequest;
    protected $request;
    protected $consumer;
    protected $token;
    protected $secret;
    protected $verifier;
    protected $error;
    protected $timeTempCred;
    protected $timeTokenCred;

    function __construct() {
        $this->tcrEndpoint="http://test76.rediris.es/osr/TCREndpoint.php";
        $this->roaEndpoint="http://test76.rediris.es/osr/ROAEndpoint.php";
        $this->trEndpoint="http://test76.rediris.es/osr/TREndpoint.php";
        $this->method="GET";
        $this->prEndpoint="http://test76.rediris.es/osr//PREndpoint.php";
        $this->httpRequest=new HTTP_Request2;
        $this->httpRequest->setHeader('Accept-Encoding', '.*');
        $this->request=new HTTP_OAuth_Consumer_Request;
        $this->request->accept($this->httpRequest);
        $this->error = null;
         $this->timeTempCred=300;
    }

    function configure($clientid, $clientSecret, $callback, $time = 300) {
        $this->clientId = $clientid;
        $this->clientSecret = $clientSecret;
        $this->callback = $callback;
```

```
43          $this->consumer = new HTTP_OAuth_Consumer($this->clientId,
44                                      $this->clientSecret);
45          $this->timeTokenCred=$time;
46      }
47
48      // Action before the Authorization of the client.
49      function preprocess($scope) {
50          try {
51              $this->consumer->accept($this->request);
52              //Configurable params
53              $args = array();
54              $args['scope'] = $scope;
55              $args['timeTemp']=$this->timeTempCred;
56              $this->scope= $scope;
57              // Gets the Temp Credentials
58              $this->consumer->getRequestToken($this->tcrEndpoint,
59                      $this->callback, $args, $this->method);
60          } catch (HTTP_OAuth_Consumer_Exception_InvalidResponse $e) {
61              $this->error = "Respuesta Invalida";
62          } catch (Exception $e) {
63              $this->error = "Ocurri&oacute; un error al intentar obtener
64                      la autorizaci&oacute;n del usuario.";
65          }
66      }
67
68      // Redirection al ROAEndpoint
69      function redirect() {
70          header("Location:".$this->consumer->getAuthorizeUrl(
71                              $this->roaEndpoint));
72      }
73
74      function postprocess($token, $verifier, $secret) {
75          try {
76              $this->verifier=$verifier;
77              $this->token=$token;
78              $this->secret=$secret;
79              $args['scope']=$this->scope;
80              $args['timeToken']=$this->timeTokenCred;
81              $this->consumer = new HTTP_OAuth_Consumer($this->clientId,
82                      $this->clientSecret, $token, $secret);
83          } catch (Exception $e) {
84              $this->error =   "La aplicaci&oacute;n no ha sido autorizada a
```

```
85                              acceder a los recursos." ;
86              }
87          }
88          //Making the Request of the Resource to the Server
89          function getResource() {
90              try {
91                  //Optional params
92                  $args = array();
93                  $args['oauth_verifier']=$this->verifier;
94                  $response = $this->consumer->sendRequest($this->prEndpoint,
95                                  $args,$this->method);
96                  return $response->getBody();
97              }catch (HTTP_OAuth_Consumer_Exception_InvalidResponse $e) {
98                  $this->error = "Respuesta inv&aacute;lida.";
99              }catch(Exception $e) {
100                 $this->error = "Ocurri&oacute; un error inesperado.";
101             }
102         }
103
104         function getError(){
105             return $this->error;
106         }
107     }
108     ?>
```

Apéndice C

Código de Aplicación Servidora

El código de la aplicación servidora puede encontrarse en la Forja de RedIRIS[35], en la dirección `https://forja.rediris.es/projects/oauthclient`