Jabber/XMPP

Oscar Lage Serrano

oscar@servidata.es

15 de Junio de 2005

Esta obra está publicada bajo la licencia de Creative Commons:

Reconocimiento-NoComercial-CompartirIgual 2.1 España

http://creativecommons.org/licenses/by-nc-sa/2.1/es/

Indice

1. La Mensajería Instantánea (Instant Messaging) y el protocolo Jabber/XMPP ........ 1

2. Fundamentos técnicos de Jabber/XMPP .................................................................. 7

2.1. Modelo de mensajería....................................................................................... 7

2.1.1. Ventajas .................................................................................................... 8

2.1.2. Inconvenientes .......................................................................................... 9

2.1.3. Comunicaciones S2S (Server to Server) ................................................ 10

2.1.4. Encaminamiento de los paquetes Jabber/XMPP .................................... 10

2.2. Ejemplo de una sesión Jabber/XMPP............................................................. 11

3. Message protocol .................................................................................................... 15

4. Presence protocol.................................................................................................... 20

5. Groupchat protocol ................................................................................................. 24

6. Info/Query protocol ................................................................................................ 27

6.1. Registration protocol (jabber:iq:register) ....................................................... 31

6.2. Authentication protocol (jabber:iq:auth) ........................................................ 35

6.2.1. Anonymous Authentication.................................................................... 37

6.2.2. Plain Authentication ............................................................................... 37

6.2.3. Digest Authentication............................................................................. 38

6.2.4. Zero-knowledge Authentication............................................................. 38

6.3. Roster Protocol (jabber:iq:roster) ................................................................... 41

6.3.1. Roster reset ............................................................................................. 43

6.3.2. Roster update .......................................................................................... 44

6.3.3. Roster push ............................................................................................. 44

7. Autentificación SASL ............................................................................................ 46

8. Comunicación Server-to-Server (jabber:server)..................................................... 51

8.1. Dialback authentication .................................................................................. 52

9. Transporte entre Jabber/XMPP y otros servidores de MI ...................................... 55

10. Chatbots .............................................................................................................. 56

11. Jabber/XMPP como middleware........................................................................ 57

12. Sitios de interés................................................................................................... 58

13. Agradecimientos................................................................................................. 59

1. La Mensajería Instantánea (Instant Messaging) y el

protocolo Jabber/XMPP

3

En Jabber/XMPP

los mensajes pasan del cliente emisor al servidor de éste, y Iq

el servidor del

dominio los envía al servidor del dominio del usuario destino para así ser

enviados al cliente destinatario del mensaje.

En los sistemas cliente/servidor, el cliente muestra información al usuario

final y maneja sus peticiones, las peticiones del cliente son enviadas al servidor

que ofrecerá unos determinados servicios.

En Jabber/XMPP esta arquitectura es seguida para crear clientes simples,

ya que así la mayoría del procesamiento de la lógica del sistema de MI es

llevada a cabo por el servidor. Esto hace más fácil la programación e

implementación de clientes de MI Jabber/XMPP , y la complejidad del cliente

dependerá de las necesidades del cliente. Así para crear un cliente de MI de

Jabber/XMPP que simplemente envíe y reciba mensajes sólo habrá que

implementar esa pequeña parte del protocolo, y la mayor parte de la

programación del cliente estará entonces en el desarrollo de la interfaz del

mismo.

De esta forma las empresas sólo

encuentran beneficios a la hora de implantar un servidor de MI Jabber/XMPP.

Además de toda la arquitectura de dominios, Jabber/XMPP también tiene

otras particularidades como puede ser la distinción de un usuario (user) y una

conexión de éste (resource). Cada usuario de Jabber/XMPP es una

terminación lógica de mensajes en la red, que representará a una persona o

una cuenta de usuario. Sin embargo, al contrario de otros sistemas de MI, en

Jabber/XMPP un usuario puede estar conectado simultáneamente a su

servidor de dominio de Jabber/XMPP desde diferentes clientes pero con su

único identificador Jabber/XMPP (Jabber ID). Esto es útil, por ejemplo, si un

usuario se conecta desde su trabajo para recibir sus mensajes, pero cuando el

está fuera de su lugar de trabajo el podría conectarse desde el teléfono para

seguir hablando, aunque el ordenador del trabajo siga conectado con su

cuenta. El usuario podrá seguir enviando y recibiendo mensajes desde su

teléfono, y cuando decida dejar de hacerlo se desconectará, por lo que todos

los siguientes mensajes seguirán llegando al ordenador del trabajo para que

pueda ver lo que le han escrito en el tiempo en el que no ha estado conectado

desde el teléfono, e incluso podrá enviar mensajes a su conexión del trabajo

para recordar algo que deba hacer cuando vuelva al trabajo, ya que el mensaje

quedará en el ordenador del trabajo. Esto es posible gracias a los “resources“

de Jabber/XMPP, o diferentes conexiones, que representan un particular punto

final de la conexión.

En la mayoría de los casos se envían paquetes a los usuarios, y éstos son

recibidos por la conexión actual del cliente. El servidor de Jabber/XMPP debe

de administrar el reenvío de estos paquetes a la conexión más apropiada del

usuario en cada momento. Si por ejemplo yo quiero enviar un mensaje al

usuario oscar que pertenece al dominio jabber.es, lo envío y el servidor recibe

el paquete. El servidor mirará si el usuario “oscar” se encuentra conectado, si

no es así almacenará el paquete para enviárselo cuando se conecte. Si el

usuario tiene una conexión abierta, el paquete le será enviado inmediatamente,

pero si tiene varias, el servidor deberá decidir a cuál de las conexiones reenviar

el paquete, es decir, cuál de las conexiones es la principal o actual conexión de

ese cliente para que pueda recibir el mensaje lo antes posible. Imaginemos que

yo tengo dos clientes Jabber/XMPP conectados con mi Jabber ID, uno en el

móvil y otro en el ordenador del trabajo. Los clientes usarán diferentes nombres

de conexión: “móvil” y “trabajo”. El servidor detectará esta conexión, y

determinará que “móvil” es mi conexión por defecto si está disponible, y

reenviará el paquete a esa dirección.

5

Cada usuario puede crear una lista de las conexiones y priorizarlas.

Además podrá comunicarse entre sus conexiones ya que sabe que la dirección

de la conexión del trabajo será:oscar@jabber.es/trabajo mientras que la del

móvil será: [oscar@jabber.es/móvil](mailto:oscar@jabber.es/móvil).

Además el protocolo nos permite enviar paquetes a un servidor en concreto,

estos paquetes no irán destinados a un usuario del servidor, si no al servidor.

Por ejemplo podríamos enviar un mensaje a “jabber.es” directamente, como lo

haremos cada vez que deseamos autentificarnos en el servidor, cambiar

nuestros datos o nuestros contactos. Además esto es usado por los servidores

de cada dominio para reenviar paquetes de sus usuarios a usuarios de otros

dominios. Además es posible enviar paquetes a una conexión del servidor en

concreto: “jabber.es/admin”.

La compacta y sencilla estructura de los Jabber ID los hace fáciles de

memorizar y usar. La única pega de este formato es la fácil confusión entre

Jabber IDs y cuentas de correo electrónico. Los administradores de dominios

Jabber/XMPP pueden simplificar esto utilizando los mismos nombres de Jabber

ID y de e-mail para sus usuarios.

Precence:

Jabber/XMPP como ya hemos dicho contiene el protocolo de presencia

permite a los usuarios configurar otros estados como “salí a comer” o “me he ido de paseo”.

Rosters

Jabber/XMPP además permite crear listas de contactos (rosters), que

permiten al usuario tener una lista de otros usuarios y conocer su estado

actual. Los contactos en Jabber/XMPP son guardados por el servidor de

dominio Jabber/XMPP del cliente, para que así estén siempre disponibles

desde donde quiera que se conecte el usuario.

Además permite confirmar o cancelar una suscripción de otro usuario. Esto

permite proteger la privacidad del usuario y determinar quién tiene permiso

para conocer el estado de cada usuario.

El sistema de presencia de Jabber/XMPP es tan flexible que permite la

aplicación del mismo para su uso en aplicaciones en las que haría falta

simplemente un servicio de presencia. Por ejemplo, con un detector de

movimientos de una alarma del hogar un usuario podría estar charlando con

sus amigos y tener como contacto a ese detector, que estaría unido a un

ordenador conectado como un usuarios Jabber/XMPP, así podría saber si hay

alguien en su jardín. Podrían así conectarse un montón de sensores de una

alarma y conocer el estado de cada puerta y ventana de la casa simplemente

uniéndolos todos a un ordenador conectado a un cliente de presencia

Jabber/XMPP. Así podríamos conocer el estado de la ventana de nuestra

habitación desde el trabajo.

6

Pero Jabber/XMPP también tiene sus problemas debido a que es un

protocolo hoy por hoy inmaduro, y en pleno desarrollo. La mayoría de la

documentación oficial del estándar Jabber/XMPP está incompleta o sin

actualizar y la única respuesta a las preguntas sobre el protocolo es probar el

comportamiento del servidor de referencia de Jabber/XMPP.

Además el protocolo sufre de ineficiencias directamente relacionadas con

su naturaleza XML, ya que los formatos binarios de datos reducen el ancho de

banda necesario para su transporte por la red, así como su procesamiento

tanto en el cliente como en el servidor. Además no disponen de la detección de

errores y corrección.

Otro problema de Jabber/XMPP es que al haber nacido tras la

popularización del MSN Messenger y de otros sistemas de MI no consigue

arrastrar a la gente hacia él. Por ello se está realizando ingeniería inversa con

los protocolos de éstos proveedores de MI para así poder crear puentes entre

Jabber/XMPP y todos los demás sistemas, así desde Jabber/XMPP los

usuarios podrían conectarse con todos los demás usuarios de otros sistemas,

así esto no sería un problema a la hora de cambiarse a Jabber/XMPP.

2. Fundamentos técnicos de Jabber/XMPP

Primero vamos a analizar el modelo de mensajería de Jabber/XMPP para

más adelante centrarnos en los tres protocolos fundamentales de

Jabber/XMPP: message, presence e info/query.

2.1. Modelo de mensajería

El cliente Jabber/XMPP es quien interactúa directamente con el usuario.

Es el programa que muestra las respuestas del servidor y que recoge las

peticiones del cliente y las envía al servidor para que las trate. Normalmente

mensajes que el usuario quiere que lleguen a otro usuario y que el servidor

deberá localizar al otro usuario y hacerle llegar el paquete.

La conexión es la realizada entre el cliente y el servidor, por la que

viajarán los paquetes XML.

Por último, pero los más importantes, los paquetes son las peticiones,

mensajes, etc. que envían o reciben los usuarios, en forma de fragmentos de

XML correctamente formados. El protocolo Jabber/XMPP es quien especifica el

formato de estos paquetes.

El servidor Jabber/XMPP, como todo servidor, es arrancado en un

ordenador y se mantiene esperando conexiones de usuarios en un puerto. El

estándar de Jabber/XMPP establece el puerto 5222 como el puerto estándar de

los servidores Jabber/XMPP y el 5223 era usado en antes de SASL para

conexiones cliente/servidor SSL seguras. Se pueden utilizar otros puertos, pero

la mayoría de los clientes Jabber/XMPP intentarán conectarse por defecto a

estos dos puertos, por lo que deberán ser reconfigurados para que se conecten

a puertos alternativos.

El cliente Jabber/XMPP comienza un flujo “stream” mandando la etiqueta

XML <stream:stream> al servidor, esto representa el inicio de una sesión del

cliente con el servidor. Cuando un servidor Jabber/XMPP acepta una conexión

de un cliente responderá con <stream:stream> para confirmar el inicio de la

sesión. Si hubiera un error en la sesión se enviaría la etiqueta <stream:error>

explicando el problema y cerrando la conexión por parte del servidor.

Una vez que se ha establecido el flujo entre el cliente y el servidor pueden

enviarse paquetes de acuerdo con los múltiples protocolos de Jabber/XMPP.

Primero la autentificación: permite al servidor verificar

que el cliente es quien dice ser para poder actuar bajo el Jabber ID bajo el que

se autentifica y enviar y recibir mensajes.

El estándar de Jabber/XMPP no especifica el conjunto de protocolos

disponibles para un usuario sin autentificar, pero como mínimo es normal que

los usuarios puedan utilizar los protocolos de autentificación y registro.

Tanto el servidor como el cliente pueden cerrar el flujo de datos en

cualquier momento enviando una etiqueta de cierre </stream:stream>. En ese

momento, cada uno podrá cerrar la conexión y terminar la sesión

Jabber/XMPP. Es conveniente dejar a la otra entidad terminar de enviar un

paquete que esté en curso antes de cerrar la conexión.

Tal y como hemos explicado esta sería una conexión típica de

Jabber/XMPP:

o Conectarse con el puerto 5222 del servidor.

o Enviar una etiqueta de sesión abierta <session:session>.

o Esperar la respuesta del servidor <session:session>.

o Usar el protocolo de autentificación o para iniciar con su Jabber ID.

o Enviar y recibir paquetes de acuerdo a los protocolos Jabber/XMPP.

o Enviar la etiqueta de cierre </session:session> para terminar la

sesión.

o Cerrar la conexión.

Esta simple conexión cliente/servidor tiene sus ventajas e inconvenientes.

2.1.1. Ventajas

El modelo cliente/servidor distribuido de Jabber/XMPP tiene muchas

ventajas. Una de las más importantes es el uso de un modelo simple y

conocido por todos los desarrolladores de software para las comunicaciones de

red. El correo electrónico por ejemplo utiliza esta misma arquitectura de

conexión. Este tipo de arquitectura sólo permite dos tipos de escenarios,

9

cliente/servidor y servidor/servidor. Para todos los servidores que no tengan

implementados los mecanismos de comunicación entre servidores, sólo se

dará la primera situación.

La seguridad y privacidad de los clientes es muy buena, debido en gran

medida a que cada cliente sólo trata con su servidor, así los demás usuarios no

tendrán ninguna referencia de dónde está ubicado realmente el cliente dentro

de la red, lo único que conocerán es su Jabber ID, y la dirección del servidor al

que ese cliente se conecta. Además el cliente nunca aceptará conexiones

como es el caso del servidor. Esto aumenta realmente la seguridad con

respecto a las conexiones punto a punto que se realizan en otros tipos de

clientes como Gnutella.

El modelo de clientes ligeros beneficia a los desarrolladores de clientes

que pueden desarrollar clientes con costes más bajos y económicos. Como

consecuencia el desarrollador podrá centrarse en aspectos importantes de cara

al cliente como son la interfaz e integración con otras aplicaciones, así como un

fácil mantenimiento del cliente.

Esta arquitectura permite al servidor del dominio centralizar el control de

Jabber/XMPP, se pueden crear políticas de seguridad y aplicarlas sin tener que

modificar los clientes o tener que controlar conexiones directas entre los

clientes, ya que como se ha dicho todo el tráfico pasa por el servidor. Así, por

ejemplo, se podrían limitar el ancho de banda a ciertos clientes en ciertas

horas, se podrían denegar el envío de archivos en horas punta de la red

empresarial, dar acceso a los clientes a comunicarse con ciertos dominios

Jabber/XMPP… las posibilidades en este campo son infinitas, y siempre habría

que configurar únicamente el servidor.

2.1.2. Inconvenientes

Como todo, la arquitectura cliente/servidor también tiene sus

inconvenientes. Como la privacidad de cara a alguien que tenga acceso al

servidor, ya que en él se almacenarán todos los mensajes que se reciban para

las cuentas Jabber ID de los usuarios que no estén conectados. Pero los

usuarios de Jabber/XMPP no están tampoco a merced del servidor

Jabber/XMPP, se pueden enviar mensajes codificados dentro de un mensaje

de Jabber/XMPP tradicional. Los algoritmos de cifrado habituales pueden

proteger los mensajes de los usuarios ante los “curiosos”.

Otro gran problema de la arquitectura cliente/servidor es que si en la red

empresarial sólo existe un servidor de Jabber/XMPP y éste permanece inactivo

durante un tiempo, los clientes no podrán comunicarse entre ellos. Este

problema tiene una fácil solución que es la instalación de un grupo de

servidores de Jabber/XMPP.

Además de la gran seguridad de las conexiones cliente/servidor, en las que

otro cliente nunca puede averiguar tu dirección, esto tiene también su gran

inconveniente. Conlleva a un cuello de botella que sólo se podrá ir

10

solucionando ampliando el servidor de Jabber/XMPP, o añadiendo nuevos

servidores, y ampliando el ancho de banda de las conexiones con el servidor.

2.1.3. Comunicaciones S2S (Server to Server)

Jabber/XMPP permite la instalación de varios dominios de Jabber/XMPP,

así como la comunicación entre el servidor responsable de cada dominio con el

resto de servidores. Los paquetes enviados por los clientes emisores son

enviados por el servidor de su dominio al servidor del dominio del destinatario

de los paquetes, y éste a su vez se los envía a los clientes destinatarios. Esta

comunicación entre servidores se denomina Server-to-Server (S2S).

El protocolo S2S es esencialmente una copia del protocolo de los clientes,

la conexión y el inicio del flujo de datos también se realiza enviando un stream

de inicio abierto. El servidor emisor actuará como cliente en representación de

sus clientes enviando los mensajes de éstos a los servidores destinatarios.

Dividiendo la red Jabber/XMPP en diferentes dominios se permite

centralizar o descentralizar las comunicaciones Jabber/XMPP dentro de la red

empresarial, con todas las ventajas e inconvenientes que ello supone. Se

podría llegar a dos extremos, en los que en uno habría millones de usuarios

conectados a un único servidor central, o al extremo opuesto en que habría

prácticamente un servidor por cada usuario. El primer extremo sería ideal de

cara a la seguridad y nefasto por el cuello de botella que ello generaría… El

otro modelo sería muchísimo menos seguro, pero la conexión entre los

usuarios del mismo dominio tendría un ancho de banda fuera de lo común en

una red de MI. Como todo el mundo sabe los extremos no son buenos nunca,

lo ideal es la organización de los clientes y servidores en una red con el fin de

que la seguridad tampoco implique quedarse sin ancho de banda para las

conexiones, siempre hay un término medio y con ello hay que “jugar”.

2.1.4. Encaminamiento de los paquetes Jabber/XMPP

Lo primero que debemos de entender es que los paquetes son enviados a

un destinatario lógico, es decir, a un cliente y no a un equipo remoto como

sería el caso de una conexión punto a punto. Es responsabilidad del

encaminamiento del servidor Jabber/XMPP asegurarse que los paquetes llegan

al usuario destinatario donde quiera que se encuentre en la red, o si no está

conectado almacenarlos para enviárselos cuando se autentifique.

También hay que darse cuenta que la MI transcurre a través del espacio y

del tiempo, los paquetes recorren la red y el servidor debe saber cuando

recogerlos y guardarlos y cuando reenviarlos a su destinatario. Para ello debe

saber si un usuario se encuentra conectado, y si es así, saber desde dónde

está conectado para hacerle llegar los paquetes. Además los paquetes deben

llegar al destinatario lo antes posible si el usuario está conectado, y si no, como

ya se ha dicho, almacenarlos y hacérselos llegar en cuanto sea factible.

El servidor se basará en el atributo “to” del mensaje para identificar el

destinatario del mismo. En este atributo el cliente emisor deberá haber puesto

el Jabber ID del destinatario, que es todo lo que el servidor debe conocer para

hacer llegar ese paquete a su fin. Cogerá la parte del Jabber ID del nombre del

servidor (la parte que sigue al carácter ‘@’) para saber si el destinatario

pertenece a su dominio, si es así extraerá el nombre (la parte que precede al

carácter ‘@’) y lo buscará entre sus usuarios, si el cliente está conectado se lo

enviará, si no es así, lo guardará para enviárselo en cuanto sea posible. Si el

destinatario no pertenece a su dominio, se conectará con el servidor

responsable del dominio, que deberá tener el nombre del dominio para ser

localizado, y le enviará el mensaje actuando en representación de su cliente.

2.2. Ejemplo de una sesión Jabber/XMPP

Gracias a los esfuerzos del protocolo para facilitar su lectura usando

mensajes XML, y la sencillez de su estructura se puede seguir de una manera

fácil una sesión simple de comunicación Jabber/XMPP. Además para un mejor

entendimiento de cada protocolo hay que tener una idea general de cómo

funciona realmente Jabber/XMPP.

Para ello vamos a mostrar una conexión entre dos usuarios Jabber/XMPP,

que seguirá este sencillo esquema:

Oscar:

1. Conexión con el servidor “jabber.es”.

2. Enviar la etiqueta de inicio de sesión.

3. Registrarse como usuario “oscar” y contraseña “oscarpass”.

4. Autentificarse como oscar en el servidor.

Iñaki:

5. Conectarse con el servidor “jabber.es”.

6. Enviar la etiqueta de inicio de sesión.

7. Registrarse como usuario “iñaki” y contraseña “iñakipass”.

8. Autentificarse como Iñaki en el servidor.

Oscar:

9. Enviar un mensaje al usuario “iñaki”.

10. Actualizar la presencia a “available”.

Iñaki:

11. Actualizar la presencia a “available”.

12. Recibir el mensaje de “oscar”.

13. Enviar un mensaje a “oscar”.

12

Oscar:

14. Recibir el mensaje de “iñaki”.

15. Cerrar la sesión.

Iñaki:

16. Cerrar la sesión.

Antes de empezar a desarrollar la conexión entre el servidor y estos dos

usuarios hay que recordar que se puede reproducir íntegramente esta sesión

abriendo la consola del sistema operativo y ejecutando la herramienta telnet

para conectarse al servidor que el usuario desee. Para ello bastará con escribir

en la línea de comandos: “telnet jabber.es 5222” donde jabber.es es el

servidor al que nos queremos conectar, y 5222 es el puerto por defecto de los

servidores Jabber.

Los pasos 1-4 y 5-8 son idénticos, lo único que varía es el nombre y la

contraseña del usuario, así que sólo se mostrará la sesión de oscar. Se

mostrará en negrita lo que el usuario escribe, y lo demás será la respuesta del

servidor:

o Paquetes 1,5:

Nos conectamos haciendo telnet al servidor de jabber.es en el puerto 5222

**% telnet jabber.es 5222**

Trying jabber.es...

Connected to jabber.es.

Escape character is '^]'.

o Paquetes 2,6:

Enviamos el tipo de xml que vamos a utilizar y la etiqueta <stream:stream>

con la información opcional del tipo de etiqueta. El servidor nos devuelve la

misma etiqueta y nos informa del id de la sesión y de su nombre

Jabber/XMPP.

**<?xml version='1.0'?>**

**<stream:stream xmlns:stream='http://etherx.jabber.org/streams'**

**xmlns='jabber:client'**

**to='jabber.es'>**

<?xml version='1.0'?>

<stream:stream xmlns:stream='http://etherx.jabber.org/streams'

id='3C0FB73C'

xmlns='jabber:client'

from='jabber.es'>

o Paquetes 3,7:

Damos de alta el usuario oscar con la contraseña oscarpass con un

mensaje iq de tipo set. El tipo de petición (query) es “jabber:iq:registrer”

para que el servidor sepa que el usuario lo que quiere es registrar su cuenta

y no autentificarse.

**<iq type='set' id=’reg\_id’>**

**<query xmlns='jabber:iq:register'>**

**<username>oscar</username>**

**<password>oscarpass</password>**

13

**</query>**

**</iq>**

<iq type='result'/>

o Paquetes 4,8:

Lo mismo que en el punto anterior, pero el tipo de petición (query) esta vez

sigue el protocolo de ‘jabber:iq:auth’ que es el de autentificación de un

usuario, nos estamos “logeando”.

**<iq type='set' id=’auth\_id’>**

**<query xmlns='jabber:iq:auth'>**

**<username>oscar</username>**

**<password>pass</password>**

**</query>**

**</iq>**

<iq type='result'

id='client\_auth\_ID'/>

o Paquete 9:

Oscar envía el mensaje al usuario Iñaki, con el título y el cuerpo.

**<message to='iñaki@jabber.es'>**

**<subject>Hola</subject>**

**<body>Este es el cuerpo del mensaje</body>**

**</message>**

o Paquete 10:

Oscar envía una actualización de su presencia a ‘available’ para hacer

saber al servidor que ya se encuentra disponible. Y el servidor le da la

bienvenida.

**<presence type='available'/>**

<message from='jabber.es’

to='oscar@jabber.es'>

<subject>Welcome!</subject>

<body>Welcome to Jabber! </body>

</message>

o Paquetes 11 y 12:

Iñaki actualiza su presencia enviando simplemente la etiqueta presence, lo

que el servidor interpreta por defecto que el cliente está “available”. A lo

cual el servidor responde con un mensaje de bienvenida y con el mensaje

que oscar le había enviado y que tenía almacenado para cuando se

conectara.

**<presence/>**

<message from=**'** jabber.es**'**

to=’iñaki@jabber.es'>

<subject>Welcome!</subject>

<body>Welcome to Jabber! </body>

</message>

<message to='iñaki@jabber.es'

14

from='oscar@jabber.es'>

<subject>Hola</subject>

<body>Este es el cuerpo del mensaje</body>

</message>

o Paquete 13:

Iñaki envía un mensaje a oscar como respuesta.

**<message to='oscar@jabber.es'>**

**<body>gracias por tu mensaje</body>**

**</message>**

o Paquete 14:

Oscar recibe el mensaje.

<message to='oscar@jabber.es'

from='iñaki@jabber.es'

<body>gracias por tu mensaje</body>

</message>

o Paquetes 15 y 16:

Y los dos usuarios se desconectan enviando la etiqueta de cierre de la

sesión.

**</stream:stream>**

Connection closed by foreign host.

Como puede verse la comunicación en Jabber/XMPP es simple y muy

intuitiva para el usuario gracias a su formato XML. Se puede entender

fácilmente la comunicación con solo unas básicas nociones de lo que se está

realizando.

3. Message protocol

Hay seis tipos de mensajes:

* normal: que serían mensajes parecidos a los del correo electrónico.
* chat: mensajes persona a persona que serían los mensajes

utilizados en una conversación entre dos personas.

* groupchat: mensajes enviados a un grupo de personas
* headline: que serían los mensajes de marquesina.
* error: para los mensajes de error.
* jabber:x:oob: para las conexiones directas entre clientes para envío
* de archivos.

Los cinco primeros son los tipos más normales de mensajes en los

sistemas Jabber/XMPP. Los mensajes “jabber:x:oob” se denominan Out-ofband

messages, y facilitan un mecanismo para intercambio de datos

directamente entre dos usuarios, estos mensajes usan el servidor

Jabber/XMPP para intercambiar datos de la conexión entre los dos clientes,

normalmente el usuario que va a servir un fichero enviaría un mensaje de este

tipo al otro cliente con la IP y el puerto al que se debe conectar el cliente que va

a descargarse el fichero.Se pueden enviar etiquetas de oob dentro de la

extensión x de un mensaje normal, o empaquetadas dentro de un paquete del

tipo Info/Query.

El protocolo de mensajes es extremadamente sencillo, los paquetes son

enviados por un usuario a un receptor. Por defecto, no se reciben

confirmaciones de que el mensaje ha sido recibido por el destinatario para

reducir el tráfico en el servidor, además si el receptor no se encuentra

disponible, el servidor guardará el mensaje hasta que se conecte, a este

protocolo se le llama “store and forward”.

Un mensaje básico consiste en la etiqueta <message> con el típico from,

to y el id del paquete como atributos. Además este tipo de paquetes soporta

cuatro subelementos:

o <subject>: indicando el título o tema del mensaje, como en los emails.

o <thread>: identificador generado por el cliente para identificar la

conversación a la que pertenece el mensaje.

16

o <body>: el mensaje en sí iría en esta etiqueta.

o <error>: si ocurriera algún error, esta etiqueta se encontraría en el

mensaje.

Así un mensaje típico sería el siguiente:

<message from='oscar@jabber.es/trabajo'

to='iñaki@jabber.es/casa'

id='messageid1'>

<thread>threadid\_01</thread>

<subject>Título del mensaje</subject>

<body>Cuerpo del mensaje</body>

</message>

Pero muchos de estos campos no son obligatorios ya que por ejemplo el id

y el thread son para un manejo más fácil de los mensajes en los clientes, no

todos los mensajes tienen por qué tener títulos y además el campo from no es

necesario ya que para evitar que una persona envíe mensajes poniendo como

origen un Jabber ID que no es el suyo el servidor reescribirá el campo from del

paquete antes de enviarlo a su destinatario, así pues, si nos hemos conectado

como el usuario “usuario1” y enviamos este paquete, a Iñaki no le llegará en el

campo from oscar@jabber.es/trabajo, si no usuario1@jabber.es.

Así pues un mensaje podría ser mucho más corto y ocupar menos ancho de

banda en su transmisión, como el siguiente ejemplo enviado por oscar:

<message to='iñaki@jabber.es'>

<body>Hola!</body>

</message>

Y al destinatario le llegaría:

<message from='oscar@jabber.es/trabajo'

to='iñaki@jabber.es'>

<body>Hola!</body>

</message>

El tipo por defecto de mensajes es el normal message. Estos mensajes

son normalmente creados y enviados usando interfaces similares a las usadas

en las aplicaciones de correo electrónico. Como el correo electrónico, estos

mensajes son enviados a usuarios, sin la necesidad de que el destinatario esté

conectado.

Los mensajes que no contienen el tipo son considerados como normales.

Pero por supuesto se puede poner el tipo a normal.

Los usuarios usan los mensajes de tipo chat para enviarse entre ellos

mensajes instantáneos. Estos mensajes suelen ser cortos, ya que se utilizan

para llevar a cabo una conversación entre dos personas. Estos mensajes son

normalmente mostrados en una interfaz de tipo chat, en la que el usuario

puede ir viendo lo que escribe el y lo que escribe su contacto.

17

Los mensajes de tipo chat deben de llevar puesto obligatoriamente el

campo del tipo con el valor chat porque si no serían considerados como

normales, al igual que el resto de los mensajes.

Estos mensajes son para conversaciones entre dos clientes únicamente,

para conversaciones entre más de dos clientes se usan los mensajes

groupchat.

Los mensajes del tipo groupchat son similares a los mensajes del tipo

chat, pero están diseñados para mantener conversaciones entre un grupo de

personas sobre un tema en concreto. Por ello cuando un cliente envía un

mensaje al grupo, todos los clientes que se hayan unido al grupo recibirán el

mensaje, aunque también se podrá escribir mensajes de este tipo privados

dirigidos a un usuario en concreto. Para formar parte de un grupo es necesario

unirse a él con un sobrenombre nickname que será mostrado al resto de los

usuarios. Esto está pensado para ocultar la identidad real de los participantes

ya que si un usuario se conecta al grupo “usuarios-xp@conference.jabber.es”

como “pepe”, a los demás usuarios no les aparecerá su verdadero Jabber

ID:”oscar@jabber.es” lo que realmente les aparecerá será “usuariosxp@

conference.jabber.es/pepe” y todos los mensajes que yo envíe al grupo

llegarán con ese identificativo, al igual que si escribo a un usuario en concreto

del grupo le llegará este mismo identificativo y me deberé dirigir a él en el

mismo formato, ya que sólo conozco al grupo al que pertenece y su nickname.

Así un mensaje que envía el usuario del grupo usuariosxp@

conference.jabber.es con nick pepe aunque sea privado le llegará al

usuario Iñaki que se ha conectado con el nick de paquito de la siguiente

manera:

o Mensaje enviado:

<message from='oscar@jabber.es/trabajo'

to=’usuarios-xp@conference.jabber.es/paco'

id='messageid4'

type='groupchat'>

<thread>threadid\_04</thread>

<body>Texto del mensaje</body>

</message>

o Mensaje recibido:

<message from=’usuarios-xp@conference.jabber.es/pepe'

to='iñaki@jabber.es/casa'

id='messageid4'

type='groupchat'>

<thread>threadid\_04</thread>

<body>Texto del mensaje</body>

</message>

Como puede verse el servidor ha cambiado el usuario destino por el

Jabber ID original del destinatario, y en el mensaje que ha llegado a iñaki, el

origen del mensaje también ha sido cambiado para ocultar su identidad. Si se

18

hubiera querido enviar un mensaje al grupo entero sería igual, salvo el campo

to que no llevaría /paco porque iría a todo el grupo.

Los mensajes del tipo headline message están diseñados para mostrar

información en la barra de estado o en otras partes de la interfaz de usuarios.

Los usan comúnmente los chatbots para informar de noticias, alertas del

tiempo… a los usuarios que se den de alta en estos servicios automatizados.

No requieren del las etiquetas <thread> o <subject>.

Otro tipo de mensajes son los mensajes de error, cuando un cliente envía

un mensaje y se produce un error, ya sea que el cliente al que va dirigido no

existe, o que sencillamente el mensaje ha sido rechazado por el cliente

destinatario.

Estos son los errores más comunes del núcleo de Jabber:

400 – Petición errónea.

401 – Desautorizado.

402 – Servicio de pago.

403 – Prohibido.

404 – No encontrado.

405 – No permitido.

406 – No aceptable.

407 – Registro requerido.

408 – Timeout.

409 – Conflicto.

500 – Error interno del servidor.

501 – No implementado.

502 – Error del servidor remoto.

503 – Servicio no disponible.

504 – Timeout de servidor remoto.

Estos códigos son válidos en Jabber-core, pero han cambiado en XMPP.

Por último están los Out-of-band messages, que no son realmente un

mensaje convencional de Jabber/XMPP. En cambio, es una extensión de los

mensajes que es enviada dentro de un mensaje normal.

Un mensaje de este tipo contiene normalmente información, normalmente

una URL, que el cliente desea usar para realizar una transferencia de datos

punto-a-punto sin pasar por el servidor. Los cliente normalmente lo suelen

implementar arrancando un servidor Web o un FTP separadamente o como

parte del cliente Jabber/XMPP. Por lo tanto están dando a otro cliente la

información de su IP y el puerto que han abierto para que el otro cliente se

descargue el archivo.

Esto es ideal para el intercambio de ficheros entre clientes ya que con ello

consigues reducir el tráfico que pasa por el servidor, pero como siempre que un

cliente revela su IP tiene sus riesgos, si la persona a la que el usuario envía

19

esta información es un usuario malintencionada podría utilizar la IP con otros

fines muy distintos a los de descargarse el fichero que el cliente le está

ofreciendo.

4. Presence protocol

Entre otras cosas la MI se diferencia del correo electrónico en la

posibilidad que tienen los usuarios de detectarse los unos a los otros y saber

cuando un usuario se encuentra conectado. Además hoy en día se va más allá

y podemos saber si aunque esté conectado al sistema de MI en ese instante

está ocupado haciendo algo y debemos esperar para hablarle o si está

conectado pero se ha ido, no está frente al ordenador. Además Jabber/XMPP

nos ofrece la posibilidad de poner tantos estados diferentes como desee el

usuario, así siempre se sentirá más identificado con el estado en el que pone

su sistema de MI.

Esto es muy importante porque si un usuario se encuentra conectado,

porque está esperando a que alguien se conecte porque tiene que charlar con

él, pero está muy ocupado trabajando, lo que menos necesita es que le lleguen

constantemente mensajes a su aplicación y le molesten, por ello pondrá su

estado a ocupado, y sus contactos deberán respetarlo y no molestarlo a no ser

que sea algo importante… Este es uno de los múltiples servicios que se

pueden encontrar al protocolo de presencia.

Además Jabber/XMPP se ha preocupado de la intimidad de los usuarios,

y si un usuario quiere agregar a otro en su lista de contactos, y recibir su

presencia, deberá pedírselo al servidor, y si el cliente que va a ser “espiado”

acepta, entonces podrá ver su estado. Esto es así porque si un usuario no

quiere que una persona sepa cuando está conectado, cuando está

comiendo… está en tu derecho a la intimidad, y será él quien decida quién

puede conocer su estado actual.

El protocolo de presencia es usado principalmente en dos contextos:

o Presence update: actualización de la presencia debido a un cambio de

estado del usuario.

o Presence subscription management: permite a los usuarios

subscribirse a las actualizaciones de presencia de otros usuarios y

controlar quien está accediendo a su propia presencia.

En ambos casos el servidor de Jabber/XMPP actúa como árbitro entre el

emisor de la actualización de presencia y los destinatarios de la misma. El

servidor tiene la obligación de hacer llegar el paquete de actualización de

presencia a todos los contactos del cliente que lo generó, pero sólo a los

contactos que el cliente emisor ha confirmado que le gustaría que recibieran su

presencia.

El servicio de actualización de presencia usa un simple mensaje

unidireccional del emisor al servidor de su dominio, es el servidor el que tendrá

que copiar y reenviar el mensaje de presencia a todos los clientes del emisor.

El servidor mira la lista de contactos del emisor para conocer quienes son sus

contactos, esta lista de contactos en Jabber/XMPP se llama Roster (siempre esta en el server esta lista).

Los clientes enviarán y

recibirán mensajes de suscripción de presencia de dos tipos:“subscribe” y

“unsubscribe”. Estos mensajes deben de ir llegando a su destinatario, pero sin

embargo el servidor deberá escuchar y actualizar la presencia de su cliente

según estos mensajes. Este tipo de suscripción de presencia también es usada

en los mensajes de grupo que veremos más adelante.

Los mensajes de presencia tienen los atributos “to”, “from” y “type” como

viene siendo habitual para todos los tipos de mensajes Jabber/XMPP. Estos

nos sirven para redirigir los paquetes y determinar su tipo.

Estos son los diferentes tipos de presencia:

o available: mensaje de actualización que indica que el usuario está listo

para recibir mensajes.

o unavailable: mensaje de actualización que indica que el usuario no está

disponible para recibir mensajes.

o subscribe: mensaje de petición de suscripción de presencia, el usuario

que lo envía desea suscribirse a la presencia del destinatario.

o unsubscribe: mensaje de cancelación de suscripción de presencia, el

usuario que lo envía desea cancelar su suscripción a la presencia del

destinatario.

o subscribed: respuesta que recibirá un usuario que ha realizado una

petición de suscripción, que indica el estado actual de la suscripción.

o unsubscribed: respuesta que recibirá un usuario que ha realizado una

petición de suscripción y le ha sido negada, o una petición de

cancelación de la suscripción.

o error: mensaje estándar de error de Jabber/XMPP que indica problemas

en la presencia.

o probe: petición Servidor-a-Servidor que envía toda la información de

presencia de un servidor a otro.

El protocolo de presencia permite además cuatro sub-etiquetas más en el

paquete de presencia:

o <status>: texto libre para que el usuario explique su estado actual.

22

o <priority>: prioridad numérica del mensaje, los números más altos

tienen más prioridad. Sólo se admiten números enteros positivos.

o <error>: paquete estándar de error de Jabber/XMPP.

o <show>: Uno de los cuatro estados estándar que los clientes pueden

usar para modificar su presencia “available”. Los clientes Jabber/XMPP

usarán normalmente el estado show para mostrar iconos de presencia

estándar, alertas sonoras y otras cosas. Si el estado show no está

indicado, el usuario se encuentra en estado normal o online. Los estados

estándar para show son:

o chat: el usuario está intentando hablar con alguien.

o away: el usuario está fuera del cliente Jabber/XMPP por un corto

periodo de tiempo.

o xa (extended away): el usuario está fuera por un periodo

prolongado de tiempo.

o dnd (do not distrub): el usuario no desea recibir mensajes.

Un ejemplo de actualización de presencia es el siguiente:

<presence from='oscar@jabber.es/trabajo'

to='jabber.es'

type='available'>

<status>Estoy aburrido, que alguien me hable</status>

<priority>10</priority>

<show>chat</show>

</presence>

Normalmente se están mostrando los campos opcionales en todos los

ejemplos para hacerlos más claros, pero un ejemplo mucho más corto de

presencia podría ser simplemente <presence/>, en el que el cliente

simplemente indicaría a sus contactos que está presente. Pero un ejemplo

mucho más típico y compacto que el primero sería el siguiente:

<presence>

<status> Estoy aburrido, que alguien me hable </status>

<show>chat</show>

</presence>9

El efecto sería prácticamente el mismo que el primero, salvo la prioridad

del mensaje, y como puede observarse este ejemplo consumiría bastante

menos ancho de banda de la red empresarial que el primero. Y en ambos

casos a mis contactos les llegaría el mismo mensaje:

<presence from='oscar@jabber.es/trabajo'

to='iñaki@jabber.es'>

<status>Estoy aburrido, que alguien me hable</status>

<show>chat</show>

</presence>

Esto es debido a que como en el protocolo de mensajes, el servidor

reescribirá el atributo “from” y añadirá el campo “to” para cada uno de los

contactos que deban recibir la presencia.

23

Los paquetes de suscripción de presencia usan un protocolo de peticiónrespuesta.

Este sería un ejemplo de una petición-respuesta de suscripción:

**<presence from='oscar@jabber.es/trabajo'**

**to='iñaki@jabber.es/casa'**

**type='subscribe'/>**

<presence from='iñaki@jabber.es/casa'

to='oscar@jabber.es/trabajo'

type='subscribed'/>

En el primero Oscar envía una petición de suscripción a la presencia de

Iñaki, y recibió la confirmación de suscripción por parte de Iñaki. La respuesta

no suele ser así de rápida, ya que Iñaki no tiene por qué estar conectado en

ese momento, y hasta que no se conecte no recibirá la petición de Oscar, en

ese momento deberá confirmarla o rechazarla, y entonces le llegará la

confirmación a Oscar, que si lo llevamos a casos extremos puede que tampoco

se encuentre conectado y la recibirá la próxima vez que se conecte. Si Iñaki

hubiera rechazado la suscripción en lugar de una confirmación “subscribed” el

tipo hubiera sido “unsubscribed”.

La presencia es usada por Jabber/XMPP con varias finalidades, entre las

cuales la más popular y más usada es la de dar a conocer a otros usuarios el

estado actual de cada cliente y saber cuando están ocupados o libres para

hablar entre ellos.

Además los mensajes de tipo groupchat usan la presencia de una forma

muy simple. Se juntan para crear grupos de conversaciones, darse de alta y de

baja de ellos…

5. Groupchat protocol

Existen cuatro acciones fundamentales que un usuario debe ser capaz de

hacer en un groupchat:

o Unirse al grupo: enviando una presencia de tipo “available” al grupo.

o Enviar mensajes a todo el grupo: enviando un mensaje de tipo

groupchat al Jabber ID del grupo deseado, al que previamente el

usuario se ha debido de unir.

o Enviar mensajes a un único miembro del grupo (mensajes privados):

enviando un mensaje de tipo groupchat a una persona del grupo,

poniendo tras el Jabber ID del grupo el carácter ‘/’ y el nickname de

la persona.

o Abandonar el grupo: enviando un mensaje “unavailable” de presencia

al grupo.

Como se puede ver, para unirse y abandonar un grupo se usa el protocolo

de presencia, y para enviar mensajes, ya sean a todo el grupo, o mensajes

privados se usa el protocolo de mensajes, pero con el tipo “groupchat”.

Para unirse a un grupo, se usa una actualización de presencia

especificando el nickname que el usuario va a utilizar a lo largo de la sesión. En

el siguiente ejemplo nos conectaremos con el nick “pepe” al grupo xpusers@

groups.jabber.es:

<presence to='xp-users@groups.jabber.es/pepe'

from='oscar@jabber.es/trabajo'/>

El servidor de groupchat asociará mi Jabber ID con el nickname pepe en

su lista interna de usuarios. Una vez que me añada a lista me enviará el

siguiente mensaje confirmándome mi suscripción al grupo:

<presence to='oscar@jabber.es/trabajo'

from='xp-users@groups.jabber.es/pepe'/>

Además enviará un mensaje de información al grupo advirtiendo de mi

incorporación al mismo, que por supuesto yo también recibiré:

<message to='oscar@jabber.es/trabajo'

from='xp-users@groups.jabber.es'

type='groupchat'>

<body>pepe has joined xp-users</body>

</message>

Como se puede ver en el atributo from del mensaje el emisor del mismo

es el grupo en sí, ya que no tiene el nickname de ningún usuario. Si Iñaki se

une al grupo la secuencia sería la siguiente:

25

Iñaki se une al grupo:

<presence from='iñaki@jabber.es/casa'

to='xp-users@groups.jabber.es/paco'/>

Iñaki recibe del servidor:

<presence to='iñaki@jabber.es/casa'

from='xp-users@groups.jabber.es/pepe'/>

<presence to='iñaki@jabber.es/casa'

from='xp-users@groups.jabber.es/paco'/>

<message to=’iñaki@jabber.es/casa'

from='xp-users@groups.jabber.es'

type='groupchat'>

<body>paco has joined xp-users</body>

</message>

En el que podemos observar que el servidor le informa a iñaki de cada

usuario que está conectado al grupo, en este caso pepe y paco que es el, y

envía el mensaje al grupo entero de información sobre un nuevo usuario.

Oscar por su parte recibirá:

<presence to='oscar@jabber.es/trabajo'

from='xp-users@groups.jabber.es/paco' />

<message to='oscar@jabber.es/trabajo'

from='xp-users@groups.jabber.es'

type='groupchat'>

<body>paco has joined xp-users</body>

</message>

Recibe la presencia de un usuario nuevo, y el mensaje informativo del

suceso.

Si yo soy oscar, no se la verdadera identidad de paco, tan solo se que su

nickname es paco, pero puedo enviarle un mensaje directamente a él porque el

servidor del grupo se encargará de reenviárselo a Iñaki.

<message to='xp-users@groups.jabber.es/paco'

from='oscar@jabber.es/trabajo'

type='groupchat'>

<body>Hola paco!!!</body>

</message>

Por supuesto el servidor hará llegar el mensaje a iñaki(paco), pero deberá

ocultar también mi identidad:

<message to='iñaki@jabber.es/casa'

from='xp-users@groups.jabber.es/pepe'

type='groupchat'>

<body> Hola paco!!!</body>

</message>

26

Como vemos ninguno de los dos puede saber el verdadero Jabber ID del

otro, sólo conocen el grupo al que ambos se encuentran conectados, y el

sobrenombre que el otro utiliza en el grupo.

Veamos ahora como llegaría un mensaje que envía Iñaki (paco) a todo el

grupo:

<message to='xp-users@groups.jabber.es’

from='iñaki@jabber.es/casa'

type='groupchat'>

<body>Hay alguien ahí?</body>

</message>

El servidor del grupo recibirá el mensaje y lo reenviará a cada miembro del

grupo sustituyendo el origen del mensaje por el nickname de Iñaki, y el

destinatario que será el cliente que lo deba recibir en cada caso, veamos como

les llega el mensaje a Iñaki y a Oscar:

<message to='iñaki@jabber.es/casa'

from='xp-users@groups.jabber.es/paco'

type='groupchat'>

<body> Hay alguien ahí?</body>

</message>

<message to='oscar@jabber.es/trabajo'

from='xp-users@groups.jabber.es/paco'

type='groupchat'>

<body> Hay alguien ahí?</body>

</message>

Si Iñaki se debe de ir, y quiere abandonar el grupo enviará un mensaje de

presencia “unavailable” al grupo:

<presence to='xp-users@groups.jabber.es/paco'

from='iñaki@jabber.es/casa'

type='unavailable'/>

Iñaki recibirá la confirmación del grupo de su marcha del mismo:

<presence to='iñaki@jabber.es/casa'

from='xp-users@groups.jabber.es/paco'

type='unavailable'/>

Y Oscar recibirá la actualización de la presencia y un mensaje informando

de la marcha de paco:

<presence to='oscar@jabber.es/trabajo'

from='xp-users@groups.jabber.es/paco'

type='unavailable'/>

<message to='oscar@jabber.es/trabajo'

from='xp-users@groups.jabber.es'

<body>paco has left</body>

</message>

6. Info/Query protocol

Este tipo de mensajes como bien indica su nombre son de peticiónrespuesta,

así que cuando un cliente envía un mensaje IQ siempre deberá

recibir una respuesta del destinatario, aunque sea te tipo error.

Un usuario en Jabber/XMPP representa a una persona. Los clientes

Jabber/XMPP actúan en representación de un determinado usuario, y un

usuario en particular puede tener varios clientes activos simultáneamente.

Como ya se ha mencionado con anterioridad los clientes que actúan en

representación de un usuarios son llamados “resources”, y se indica al final del

Jabber ID del usuario.

El usuario es representado en el sistema Jabber/XMPP por una cuenta de

usuario almacenada y gestionada en el servidor Jabber/XMPP del dominio. Los

clientes se autentifican con el servidor Jabber/XMPP usando la información de

autentificación. Una vez el usuario se ha autentificado, el cliente puede

descargar o actualizar la información de su cuenta de usuario almacenada en

el servidor.

Este protocolo se centra fundamentalmente en los procedimientos para

crear y almacenar la información de un usuario en el servidor.

Aunque la mayoría del tráfico de un sistema de MI Jabber/XMPP está

compuesto de mensajes y presencias, el mayor esfuerzo a la hora de

implementar un cliente o servidor Jabber/XMPP radica en la administración de

las cuentas de usuarios. Jabber/XMPP ha implementado un protocolo de

petición/respuesta para gestionar estas peticiones.

IQ es un protocolo sencillo y ampliable de petición-respuesta que permite

a los usuarios hacer peticiones, y almacenar o cambiar datos usando un

sencillo protocolo. IQ es simplemente el conductor de esas peticiones y

respuestas, y gestiona qué datos son necesarios de acuerdo con las

necesidades particulares de cada servidor.

IQ es totalmente ampliable por los desarrolladores, que encuentran en

este servicio de IQ el lugar idóneo para sus ampliaciones de gestión. La

mayoría de las peticiones IQ son entre un cliente y un servidor. Sin embargo,

hay algunos protocolos de IQ que van estrictamente de un cliente a otro, como

el protocolo de petición de versión del cliente, en que un cliente le pedirá al otro

su versión del programa.

El protocolo IQ tiene dos participantes, el que genera la petición y el que

la trata. Pueden darse dos situaciones, una comunicación entre un cliente y un

servidor, en la que el cliente será quien envíe los paquetes IQ, o entre dos

cliente, en cuyo caso un cliente realiza una petición a otro cliente el cual le

deberá responder.

28

El protocolo comienza con una petición de un cliente a otro cliente o

servidor que gestionará su petición. El paquete IQ será encaminado por el

sistema Jabber/XMPP como cualquier otro paquete hasta ser entregado a su

destinatario. Pero al contrario de un mensaje Jabber/XMPP, el paquete no será

almacenado si el receptor del paquete no se encuentra disponible. Además los

paquetes IQ que se dirigen a un usuario no pueden ser enviados al Jabber ID

de un cliente, hay que especificar el cliente concreto de ese Jabber ID, es decir,

habrá que especificar el “resource”. A no ser que el destinatario sea un

servidor, que en tal caso el paquete si puede ir dirigido al servidor o incluso no

mencionar el destinatario si es el servidor de tu dominio. Ya se comentó

anteriormente que el servidor adopta como suyos todos los paquetes que no

tienen destinatario.

Los paquetes IQ pueden ser de dos tipo, “set” o “get”, el primero como se

sobreentiende es para modificar datos que posee el destinatario del paquete, y

el segundo para solicitarlos. El destinatario tratará el paquete y enviará una

respuesta siempre, en el primer caso responderá si se han podido modificar

correctamente o si ha habido algún tipo de error, y en el segundo caso nos

devolverá la información que le hemos solicitado o un mensaje de error.

El formato de un paquete IQ sería el siguiente:

<iq type=**'set|get|result|error'**

to=**'jid\_destino'**

from=**'jid\_origen'**

id=**'unica'>**

<query xmlns=**'iq extension namespace'>**

**<query\_field1/>**

**<query\_field2/>**

</query>

</iq>

El atributo “type” indica el tipo de petición que se va a enviar, que será uno

de los siguientes tipos:

o get: el emisor solicita información al destinatario.

o set: el emisor solicita la actualización de los datos que tiene el

destinatario.

o result: respuesta de una solicitud IQ anterior.

o error: respuesta de error en el procesamiento de una petición IQ

anterior.

Como puede observarse los dos primeros tipos serían los que utilizaría el

cliente que emita la petición IQ y los dos últimos serían las dos posibles

respuestas de quien reciba la petición. También hay que fijarse que se pueden

dar varias peticiones/respuestas <query> dentro de un mismo mensaje iq. Esto

es sobre todo más habitual en la respuesta que el la pregunta, ya que si por

ejemplo un cliente solicita a su servidor la lista de contactos, el cliente enviará

una única petición, pero el servidor responderá con un <query> por cada

29

contacto que ese cliente tenga almacenado en el servidor, es algo similar a una

respuesta de una sentencia SQL que devuelve múltiples resultados de acuerdo

a la petición que le hemos hecho.

Hay que mencionar que un paquete iq no tiene porqué tener

obligatoriamente sub-elementos <query>. Muchas veces las respuestas no

contienen ningún elemento <query>. Pero los paquetes iq pueden tener

también otros elementos como pueden ser los <vCard>.

Los emisores de peticiones IQ no tienen que esperar a la respuesta para

poder enviar otra petición IQ, tampoco deben esperar que las respuestas a sus

peticiones lleguen en el mismo orden en que las enviaron, para distinguir las

respuestas se guiarán de la etiqueta id que será igual tanto en la petición como

en la respuesta. Esta identificación la genera el cliente al hacer la consulta y el

destinatario la copiará en la respuesta, ya sea una respuesta de error o un

resultado.

Como puede verse en el ejemplo la etiqueta <iq> por si sola contiene

además de los atributos típicos de Jabber/XMPP elementos <query>. Esto

permite a los servidores tratar a todos los paquetes iq de la misma forma, sin

tener que conocer el contenido de las peticiones, si es que no van dirigidas a

ellos. Esto es muy importante, ya que así pueden implementarse peticiones

que no pertenezcan al protocolo, y serán reconducidas perfectamente por

todos los servidores de Jabber/XMPP, aun desconociendo el contenido de las

peticiones, por ello es muy importante este formato en el que Jabber/XMPP

encapsula las peticiones dentro del paquete iq. Esto abre las puertas a los

desarrolladores a la hora de ampliar el protocolo Jabber/XMPP y adaptarlo a

sus necesidades, y libera a los servidores de tener que conocer todos los tipos

de peticiones aunque no las tuvieran que tratar nunca.

Los paquetes <query> que van dentro de los paquetes <iq> establecen un

namespace XML usando el atributo xmlns. Cada namespace establece una

extensión IQ con sus propias etiquetas y atributos, y un protocolo para su

procesamiento. Así que si nos encontramos que el atributo xmlns contiene

“jabber:iq:foobar” la petición seguirá el protocolo foobar que pertenece a una

extensión de iq, y para tratarlo el servidor o cliente destinatario deberá conocer

también dicha extensión, si no devolverá una respuesta de error.

Normalmente los elementos <query> contienen simplemente hijos que

representan etiquetas de datos. En peticiones IQ de tipo get las etiquetas

rellenas servirán al destinatario para realizar la búsqueda y las vacías serán los

campos que pide el cliente, así que en la respuesta al cliente los campos que

envió vacíos deberían de volver rellenos.

Si por ejemplo realizamos una extensión que definimos como

my:query:namespace y se trata de una búsqueda entre los contactos del

usuario, un ejemplo de esa petición sería:

<iq type='get' to='jabber.es' from='yo@jabber.es'>

<query xmlns="my:query:namespace">

<name-first>Igor</name-first>

30

<name-last/>

<email/>

</query>

</iq>

Y su respuesta:

<iq type='result' to='yo@jabber.es' from='jabber.es'>

<query xmlns='my:query:namespace'>

<name-first> Igor </name-first>

<name-last>García</name-last>

<email>IgorGarcia@yahoo.es</email>

</query>

<query xmlns='my:query:namespace'>

<name-first> Igor </name-first>

<name-last>Sánchez</name-last>

<email>igorSanchez@hotmail.com</email>

</query>

</iq>

La mayoría de los patrones de petición-respuesta básicos siguen este

modelo. La mayor diferencia entre las diferentes extensiones del protocolo IQ

son los campos y su significado para quien lo envía y para quien procesa la

petición. Las extensiones de IQ se organizan según las necesidades de cada

aplicación.

El diseño del protocolo IQ permite una rápida y eficiente forma de agregar

tus propias extensiones, siempre que tanto el emisor como el receptor

entiendan la extensión y la tengan implementada para poder tratarla

adecuadamente. Pueden perfectamente operar dentro de una red estándar de

Jabber/XMPP dos clientes que conozcan una extensión propia y necesiten

usarla, ya que la extensión va empaquetada dentro del estándar IQ y será

reenviada correctamente por el servidor o servidores Jabber/XMPP por las que

tenga que pasar.

Existen actualmente más de 15 extensiones de IQ dentro del protocolo

Jabber/XMPP y otras 10 en fase de desarrollo, que dentro de poco serán parte

del estándar. Pero cualquier desarrollador puede implementar sus propias

extensiones según sus necesidades.

Pasemos a ver a fondo las extensiones IQ más importantes.

6.1. Registration protocol (jabber:iq:register)

El primer paso para poder tener usuarios en el servidor es dejar a los

usuarios crear sus cuentas en el servidor mediante esta extensión del protocolo

IQ. Esto no ocurrirá así en un servidor de uso privado de una red empresarial,

en el que uno de los factores importantes es la seguridad. Pero para cualquier

servidor público de Jabber/XMPP es necesario implementar el protocolo de

registro para que cualquier persona que lo desee pueda registrarse y unirse a

la comunidad de usuarios Jabber/XMPP.

Esta extensión es usada para tres propósitos:

o Crear cuentas de usuario.

o Actualizar la información de las cuentas de usuario.

o Borrando cuentas de usuario.

Como ya hemos dicho este protocolo es usado especialmente en los

servidores públicos de Jabber/XMPP que proveen del servicio de MI

Jabber/XMPP al público en general. Obviamente, no en todas las situaciones

los administradores de los servidores Jabber/XMPP dejarán al público crear

sus propias cuentas de usuarios sin permiso. La situación es parecida a la de

los servidores de correo, nuevamente, como todo el mundo sabe existen

servidores de correo que cobran por sus servicios, o que ofrecen sus servicios

como un complemento adicional a otro servicio principal, en estos servidores

un usuario cualquiera no puede darse de alta y disfrutar de una cuenta de

correo. Sin embargo, existen otros servidores de correo, llamémoslos públicos,

en los que cualquier persona puede dar de alta una cuenta de correo una vez

introducidos unos datos personales mínimos. La situación es muy parecida con

los servidores Jabber/XMPP.

En los servidores públicos, y en su más pura esencia, las cuentas de

usuario no son más que unos repositorios de credenciales que los clientes

usan para autentificarse con el servidor. Además de estos datos básicos,

muchos servidores asocian otros datos a la cuenta de usuario como pueden

ser el e-mail, el nombre completo del usuario y su teléfono, así como otros

datos que el servidor quiera recoger de sus usuarios.

Además muchos usuarios también almacenan “vCards” a su cuenta de

usuario. Las “vCards” son como tarjetas de negocio, en formato XML, que

contienen información de contacto sobre una persona.

Los servidores tienen muchas formas de almacenar la información de un

usuario. La forma de almacenar estos datos está fuera del alcance del

protocolo Jabber/XMPP y por ello cada servidor lo realizará de la forma que

mejor se adapte a sus necesidades. Una forma muy utilizada, y que también la

utiliza el servidor de referencia de Jabber/XMPP (jabberd) es usar ficheros XML

32

con los datos de cada usuario dentro de un directorio. Otras implementaciones

pueden ser por ejemplo usar una base de datos o hacer persistente los datos

mediante tecnologías como J2EE o Hibernate.

Si el servidor no tiene implementado este protocolo de registro se deberán

desarrollar utilidades para el tratado de las cuentas de usuario. Otros

servidores tendrán implementado este protocolo y además tendrán otras

herramientas para el manejo de las cuentas de usuario. O este trabajo puede

hacerse “a mano” sin ser automatizado como en jabberd que además de

implementar el protocolo, el administrador puede crear cuentas de usuario

simplemente creando ficheros XML con el formato adecuado y dejándolos en la

carpeta de “spool” correspondiente donde se encuentran el resto de los

usuarios.

Aunque el manejo y almacenamiento de las cuentas de usuario puede ser

algo complicado, la implementación del protocolo de registro no es así.

El protocolo de registro normalmente suele ser junto con el de

autentificación el único protocolo que un usuario sin autentificar puede usar de

un servidor Jabber/XMPP después de iniciar la conversación con la etiqueta

<stream:stream>.

En general el protocolo implica una consulta “get” para comprobar el tipo

de registro implementado en el servidor, y los datos necesarios para el registro,

y un “set” para el registro en sí, en el que se deberán enviar al menos los datos

obligatorios como son el nombre de usuario y la contraseña. Hacer la petición

”get”, antes de registrarse no es obligatorio, pero dará información al cliente de

que campos son soportados por el servidor en el registro, ya que el envío de

campos no implementados por el servidor no está implementado en el

protocolo, y se desconoce su respuesta.

El servidor normalmente borrará los campos no soportados o los

almacenará aunque nunca los utilizará. En cualquiera de los casos el cliente

estará malgastando ancho de banda. Por esa razón es recomendable que se

pida al servidor que campos necesita antes de registrar a un usuario.

La prueba de registro implica el envío de la etiqueta “get” en un mensaje

vacío:

<iq type='get' id='register\_get\_id'>

<query xmlns='jabber:iq:register'/>

</iq>

El servidor responderá con un paquete IQ de respuesta que contendrá las

etiquetas vacías de los campos que soporta, o un error IQ informando que no

soporta el protocolo de registro. La respuesta típica sería la siguiente:

<iq type='result' id='register\_get\_id'>

<query xmlns='jabber:iq:register'>

<username/>

<password/>

33

<hash/>

<email/>

<phone/>

</query>

</iq>

El servidor puede indicar cero o más etiquetas de registro como en este

caso. Los clientes generalmente crearán un formulario con un campo de texto

seguido tras una etiqueta que informará de lo que hay que rellenar en el campo

de texto.

Hay que fijarse que campos como <username>, <password> o <hash>

son estándar y siempre pertenecerán a la petición. Y el cliente siempre los

reconocerá como campos especiales.

El cliente utilizará la contraseña para generar los credenciales de

autentificación que van dentro de los campos <password>, <hash>,

<sequence> y <token>.

La aplicación cliente cogerá la información que el usuario le proporcione y

generará un paquete de registro que será enviado al servidor:

<iq type='set' id='register\_set\_id'>

<query xmlns='jabber:iq:register'>

<username>Nombre</username>

<password>Password</password>

<email>Nombre@server.com</email>

<phone>94 444 55 66</phone>

</query>

</iq>

En el caso de que en la prueba del servidor, éste haya contestado que

admite el campo hash significará que admite el protocolo de “zero-knowledge

authentification” entonces el cliente deberá generar los credenciales de los que

se ha hablado anteriormente:

<iq type='set' id='register\_set\_id'>

<query xmlns='jabber:iq:register'>

<username>Nombre</username>

<hash>23ea323be3231</hash>

<sequence>100</sequence>

<token>9823cd2323fa</token>

<email>Nombre@server.com </email>

<phone>94 444 55 66</phone>

</query>

</iq>

Si todo ha ido correctamente el servidor creará la cuenta de usuario y

devolverá una respuesta vacía:

<iq type='result' id='register\_set\_id'/>

Si no es así, el servidor devolverá un mensaje de error IQ normal

indicando por qué se ha producido el fallo. Normalmente será debido a que el

nombre de usuario que se intenta registrar ya está usado en el servidor, o si el

usuario estuviera intentando modificar algún dato de la cuenta de usuario sin

autentificarse previamente también devolvería un mensaje de error.

34

35

6.2. Authentication protocol (jabber:iq:auth)

El protocolo de autentificación de Jabber/XMPP es, como el de registro,

una extensión del protocolo IQ. Es uno de los protocolos únicamente dedicados

a la seguridad en Jabber, permite a los usuarios demostrar a su servidor que

son quien dicen ser, aunque muchas veces el cliente no tiene la certeza de que

el servidor sea quien dice ser, ya que dentro de una red un cliente puede

interceptar sus mensajes y engañarle, pero esto ya no es cosa del protocolo

Jabber/XMPP, si no fallos que derivan directamente de las direcciones IP v.4.

Este sistema de autentificación pertenece al núcleo del protocolo Jabber,

es decir, a los inicios del protocolo, desde que salió el estándar Jabber/XMPP

la autentificación se realiza mediante SASL, aunque siguen existiendo

servidores que admiten este tipo de autentificación.

El sistema de autentificación y acceso a un servidor es simple: los clientes

no autentificados tienen una serie de permisos, y los clientes autentificados

tienen un completo acceso al uso de todos los protocolos de Jabber/XMPP que

estén implementados en el servidor del dominio al que pertenecen.

Desafortunadamente, este simple modelo de autentificación está muchas

veces limitado. Por ejemplo en la actualidad no hay una forma estándar de

restringir el acceso a los grupos de conversación… Además si un administrador

de un sistema Jabber/XMPP desea restringir los mensajes a ciertos dominios

Jabber/XMPP, limitar el tamaño de transferencias de ficheros, el tamaño de los

paquetes u otras funciones administrativas, se verá ante la ausencia en el

protocolo Jabber/XMPP de estándares sobre este tipo de funciones

administrativas, todo este tipo de técnicas quedan de momento fuera del

protocolo y deberán ser diseñados e implementados por el administrador del

sistema y los desarrolladores del servidor.

Otro problema es que Jabber/XMPP no distingue entre rangos o grupos

de usuarios, todos los usuarios tienen según el protocolo el mismo derecho a

usar todos los protocolos implementados en el servidor, sin embargo esto

debería ser diferente. Se están juntando dos conceptos diferentes como son el

de autentificación y autorización, una persona autentificada no debería estar

autorizada a hacer todo lo que deseara. Sin duda este aspecto se retomará

cuando el protocolo esté en plena madurez.

Quitando estos pequeños inconvenientes que pueden ser solucionados

por los desarrolladores aunque no estén tratados en el conjunto del protocolo

Jabber/XMPP, el protocolo de autentificación de Jabber/XMPP aporta garantías

suficientes para cualquier sistema de MI y sus actividades. Además el protocolo

nos ofrece cuatro niveles diferentes de autentificación:

o Anonymous.

o Plain.

o Digest.

o Zero-knowledge.

36

Como el protocolo de registro, el protocolo de autentificación también

consta de dos fases diferentes, una de consulta en la que el cliente podrá

conocer cuáles de los tres tipos de autentificación que propone el estándar

están implementados en el servidor, que como resultado nos devolverá datos

de autentificación usados para el método zero-knowledge, si es que el servidor

lo soporta. La segunda fase consiste en la autentificación en sí. El cliente

enviará una petición “set” que contendrá los datos necesarios para la

autentificación según el tipo de autentificación que el usuario elija.

La prueba de autentificación utiliza una petición “get” en la que se deberá

especificar el nombre del usuario que se pretende autentificar. El servidor

retornará los métodos de autentificación disponibles para ese usuario en

concreto y los campos que el usuario podría utilizar para autentificarse.

Un ejemplo típico de prueba de autentificación sería el siguiente:

<iq type='get' id='auth\_get\_id'>

<query xmlns='jabber:iq:auth'>

<username>oscar</username>

</query>

</iq>

La presencia o ausencia de determinados campos en la contestación del

servidor indicará que tipos de autentificación soporta el servidor:

<iq type='result' id='auth\_get\_id'>

<query xmlns='jabber:iq:auth'>

<username>oscar</username>

<resource/>

<password/>

<digest/>

<token>33ab323</token>

<sequence>99</sequence>

</query>

</iq>

La presencia de <password> indica que admite la autentificación en modo

texto de la contraseña, es decir, autentificación “plain”. El campo <digest>

indica que admite la autentificación tipo “digest”. Finalmente la presencia de

<token> y de <sequence> conteniendo datos indica que es posible una

autentificación del tipo zero-knowledge. Es posible recibir un resultado que no

contenga ninguno de esos campos, en tal caso, el servidor en cuestión sólo

soportaría usuarios anónimos, por lo tanto, sin autentificar.

Una vez que el cliente ya conoce los diferentes métodos con los que

puede autentificarse deberá seleccionar el más adecuado según su criterio. Es

importante que sólo se envíen los campos de un tipo de autentificación en la

petición “set” de autentificación. El resultado del envío de múltiples tipos de

autentificación simultáneos no está contemplado en el estándar de

Jabber/XMPP. Sin embargo el protocolo de autentificación de Jabber/XMPP no

contempla la forma de salir del usuario en el que te has autentificado, para

finalizar la sesión se deberá enviar la etiqueta de cierre </stream:stream> para

37

cerrar la sesión con el servidor, y en el caso de querer conectarse como otro

usuario diferentes se deberá volver a conectar con el servidor para su posterior

autentificación.

6.2.1. Anonymous Authentication

Si el servidor admite usuarios anónimos bastará con el envío de la

petición “set” sin ningún otro tipo de datos:

<iq type='set' id='auth\_set\_id'>

<query xmlns='jabber:iq:auth'/>

</iq>

El servidor responderá con un error si no soporta usuarios no

autentificados. Pero si soporta usuarios anónimos devolverá como resultado

del “set” un nombre aleatorio:

<iq type='result' id='auth\_set\_id'>

<query xmlns='jabber:iq:auth'>

<resource>nombreAleatorio</resource>

</query>

</iq>

El usuario podrá entonces enviar mensajes, donde su Jabber ID estará

formado por el nombre del servidor seguido del carácter ‘/’ y del recurso

aleatorio que ha devuelto en la autentificación anónima, es decir:

Jabber ID: jabber.es/nombreAleatorio

6.2.2. Plain Authentication

Este tipo de autentificación es el primero que provee de algún tipo de

seguridad. Su principal ventaja es su extrema sencillez a la hora de su

implementación. Funciona enviando dentro del XML de autentificación la

contraseña en formato de texto sin codificar, del usuario a autentificar.

<iq type='set' id='auth\_set\_id'>

<query xmlns='jabber:iq:auth'>

<username>oscar</username>

<resource>trabajo</resource>

<password>pass</password>

</query>

</iq>

El principal problema de este tipo de autentificación es que la contraseña

es enviada de forma abierta al servidor. Es fácil que si alguien está escuchando

la red pueda sacar la información de nuestro nombre de usuario y contraseña y

utilizarla para suplantar nuestra identidad en el sistema Jabber.

38

6.2.3. Digest Authentication

Para evitar el envío de contraseñas en texto sin codificar, mediante este

tipo de autentificación el servidor enviará un identificador de sesión en el

campo “id” junto con la etiqueta <stream:stream> de inicio de sesión. Para

generar la autentificación de tipo digest el cliente concatenará el identificador

de la sesión con la contraseña del usuario. La cadena resultante será

codificada usando el algoritmo SHA-1. El texto en hexadecimal y minúsculas

será enviado en la etiqueta <digest> en la petición de autentificación:

<iq type='set' id='auth\_set\_id'>

<query xmlns='jabber:iq:auth'>

<username>oscar</username>

<resource>trabajo</resource>

<digest>139ab93c13f31</digest>

</query>

</iq>

En java, por ejemplo, este algoritmo de autentificación se implementaría

de una forma sencilla utilizando el paquete java.security.MessageDigest.

El único inconveniente de este tipo de autentificación es que la contraseña

del usuario debe ser enviada durante el protocolo de registro como texto sin

codificar. Además el servidor guardará la contraseña en formato de texto sin

codificar. Un fallo en la seguridad del servidor puede poner en problemas la

seguridad de las contraseñas de sus usuarios. Sin embargo la contraseña sólo

viajará por la red empresarial o Internet durante el registro, luego irá siempre

codificada.

Todos los problemas serán solucionados con la siguiente forma de

autentificación.

6.2.4. Zero-knowledge Authentication

Es el formato más seguro y más complicado soportado por los protocolos

Jabber/XMPP, muchas veces lo podemos encontrar escrito como “0k”. El

método de autentificación 0k es complejo y su adopción tanto en servidores

como en clientes ralentizará el proceso de autentificación.

Este tipo de autentificación ya no guarda la contraseña del usuario en el

servidor. De hecho, la información que el servidor guarda son las credenciales

que sólo servirán para una única autentificación del cliente. El servidor irá

creando nuevas credenciales de un solo uso.

La técnica para generarlas utiliza cuatro tipos de información:

o La contraseña del usuario: usada por el cliente para generar llaves

de tipo 0k. La contraseña es almacenada en el cliente y nunca será

39

enviada al servidor. La llave será creada con la combinación de la

contraseña del usuario y el token.

o Token: información generada aleatoriamente usada para crear la

llave de 0k. El token es almacenado en el servidor. Separando la

contraseña y el token entre el cliente y el servidor respectivamente

se consigue que la llave creada sea única para el conjunto

cliente/servidor.

o Sequence: número que decrece automáticamente indicando que

llave del conjunto de llaves está siendo usada en el momento.

o Hash: llave particular en el conjunto de llaves identificada por el

número de secuencia explicado anteriormente.

Inicialmente el cliente generará todas las piezas que se usarán en el

protocolo de registro, para hacerlo el cliente:

1. Creará un mensaje de tipo digest con SHA-1 de la contraseña

del usuario para crear el hashA. La salida se convertirá a

hexadecimal en minúsculas.

2. Generará un token aleatorio.

3. Generará el digest de la combinación de los resultados de los

dos puntos anteriores para crear el has0 que será pasado a

hexadecimal en minúsculas.

4. Selecciona arbitrariamente un número de secuencia.

5. Digest del hash-n para crear el hash-(n+1) y convertirlo en

hexadecimal. Así hasta generar la secuencia m que será el

número del paso 4.

El cliente envía el token, la secuencia, y el hash al servidor en el

protocolo de registro si éste soporta autentificación 0k. Para autentificarse, el

cliente seguirá dos pasos.

El primero consistirá en enviar una prueba de autentificación y el servidor

devolverá el token, y el número de secuencia menos uno. El cliente coge este

token, y el número de secuencia y genera un nuevo hash(m-1). El cliente para

ello sigue los mismos pasos que los explicados anteriormente excepto que

utiliza el token y el número de secuencia dados, y se los envía al servidor.

El servidor coge el hash-(m-1) y genera el hash m mediante digest.

Compara el enviado por el cliente y el que ha generado él, si coinciden el

usuario habrá sido autentificado. Si no coinciden devolverá un mensaje IQ de

error.

Si todo es correcto el servidor decrementa el número de secuencia y lo

almacena. La próxima vez que el cliente se autentifique, el servidor enviará el

token M-2 al cliente. El proceso continuará hasta que el número de secuencia

llegue a cero. El cliente deberá de volver a usar el protocolo de registro para

actualizar el token y el número de secuencia antes de que éste llegue a cero.

40

Hay que fijarse que el servidor no puede predecir qué hash n-1 saldrá del

hash n. Es realmente una llave de un solo uso. Si alguien robara la llave y ve

que funciona esa llave sólo le servirá para esa vez, porque no se podrá volver a

autentificar ya que no podrá adivinar la siguiente llave para el siguiente número

de secuencia.

Este tipo de autentificación tiene muchas ventajas:

o Las contraseñas nunca son enviadas por la red.

o Las contraseñas nunca son almacenadas en el servidor.

o Las contraseñas robadas dejan de ser válidas tan pronto como son

usadas.

o La mayoría del procesamiento es para el cliente, liberando al

servidor que podría generar si no un cuello de botella en el sistema.

La única vulnerabilidad del sistema es la no actualización de los

credenciales antes de que el número de secuencia llegue a cero. La

actualización de los mismos sólo es posible una vez el cliente se haya

autentificado.

Realmente sugerimos que únicamente los clientes autentificados con una

conexión segura (SSL) puedan actualizar las credenciales, si no es así un

atacante utilizando la técnica de “man in the middle” podría enviar nuevos

credenciales al servidor una vez que el usuario se haya autentificado robando

la cuenta de usuario.

Como puede verse la implantación del sistema 0k conlleva un gran

esfuerzo para los desarrolladores pero puede merecer la pena según el

sistema que se quiera implantar sobre Jabber/XMPP. Creemos que no tendría

sentido implantarlo en una simple aplicación de MI, pero sí en otro tipo de

aplicaciones que requieran este tipo de seguridad para sus conexiones

Jabber/XMPP. Este protocolo fue adoptado por Jabber/XMPP recientemente, y

esperemos que dentro de un tiempo todo haya avanzado lo suficiente para que

no sea un gran gasto de procesamiento y que se aplique en cualquier tipo de

aplicación Jabber/XMPP.

41

6.3. Roster Protocol (jabber:iq:roster)

Para no tener que enviar los cambios de presencia entre todos los

usuarios del sistema Jabber/XMPP ha creado el concepto de suscripción de

presencia. Como su nombre indica, la suscripción de presencia determina los

suscriptores que recibirán las actualizaciones de presencia de cada usuario.

Los suscriptores pedirán una suscripción a un usuario, y el usuario aceptará o

denegará dicha suscripción. Cada usuario se suscribirá a los usuarios que

desee y podrá aceptar que dichos usuarios u otros usuarios se suscriban a los

cambios de su presencia.

Para organizar y administrar las suscripciones de cada usuario,

Jabber/XMPP ha definido unas estructuras de datos estándar conocidas como

Jabber/XMPP roster. Jabber/XMPP roster no es mas que una lista de otros

usuarios identificados por su Jabber ID. El protocolo de suscripción de

presencia permite a los usuarios suscribirse a la presencia de otros usuarios,

sea cual sea su dominio de Jabber. Los servidores usan el protocolo de

suscripción de presencia para sincronizar los rosters para sus usuarios tanto

fuera como dentro de su dominio.

Los diferentes tipos de relaciones de suscripción tratados por el roster son

los siguientes:

o to: el usuario está interesado en recibir las actualizaciones de

presencia del suscriptor.

o from: el suscriptor está interesado en recibir las actualizaciones de

presencia del usuario.

o both: el usuario y el suscriptor tienen un mutuo interés en recibir las

presencias entre ellos.

o none: ni el usuario ni el suscriptor tienen interés por recibir

presencias entre ellos.

Además de la información básica de suscripción, el roster permite al

usuario almacenar información estándar del interfaz de ese suscriptor. Esta

información incluye un sobrenombre puesto por el usuario a su suscriptor y el

grupo o grupos al que pertenece el suscriptor a la hora de mostrarlo en la

interfaz.

Una de las cosas más importantes es que toda esta información está

almacenada y administrada por el servidor. Esto simplifica notablemente la

implementación y permite al usuario tener disponible dicha información allí

donde se encuentre. Cualquier cambio realizado en el roster en uno de los

clientes será automáticamente actualizado en los demás clientes iniciados con

el mismo Jabber ID. El protocolo roster fue desarrollado para permitir a los

clientes su administración.

42

A pesar de la estrecha relación entre el roster y la presencia, son

conceptos diferentes. Digamos que el roster es todo el conjunto de contactos

relacionados con nuestra cuenta Jabber. Además podemos guardar datos de

cada contacto como un sobrenombre puesto por el usuario o el grupo o grupos

al que pertenece para poder así buscar todos los contactos de un grupo, y no

tener que recorrernos todos los contactos. Como cada usuario tendrá su propio

roster, si enviamos una actualización de presencia al servidor, éste buscará en

nuestro roster todos los contactos que tengan los tipos de suscripción “both” o

“form” para reenviarles sólo a ellos nuestra presencia.

Como todo se almacena en el servidor, el cliente comenzará nada más

autentificarse pidiendo todo el roster al servidor con un roster reset. Mostrará

entonces todos los contactos del roster en la aplicación. Si más adelante el

usuario desea hacer cambios en algún contacto del roster enviará una roster

update al servidor y se quedará esperando un roster push del servidor para

confirmar la actualización a todos los clientes abiertos con el mismo Jabber ID,

ya que si uno de ellos realiza un cambio, éste se tiene que reflejar en el resto.

Los cambios del roster pueden suceder en cualquier momento desde que

un usuario se autentifica, por lo tanto los clientes deben estar preparados para

recibir del servidor roster push y actualizar los contactos que muestran en ese

momento. El servidor enviará un roster push cuando:

o Una actualización del roster cambie los atributos del mismo, y se

deba actualizar los clientes mostrados o alguno de sus atributos.

o Cuando por algún cambio en el tipo de suscripción de presencia se

deba crear o borrar un contacto.

Como se ha visto el protocolo de suscripción de presencia tiene grandes

efectos sobre el roster, tantos que una actualización del tipo de suscripción

puede hacer desaparecer de la pantalla del usuario un contacto, ya que si el

contacto no desea ser visto por el usuario y envía una cancelación de la

suscripción de presencia poniéndola a “none” el usuario debe de dejar de ver el

estado de ese contacto lo antes posible.

Los clientes sólo pueden modificar el apodo de sus contactos y el grupo o

grupos a los que pertenecen mediante el protocolo roster. Además, como

hemos visto pueden influir en el roster mediante el protocolo de suscripción de

presencia.

El diseño de las suscripciones de presencia y los roster de Jabber/XMPP

facilitan el desarrollo de los clientes Jabber/XMPP, ya que los clientes no deben

almacenar esos datos, ni tampoco se deben de preocupar de cómo

modificarlos o administrarlos, lo único que deben hacer es realizar peticiones al

servidor y será éste quien se encargue de su procesamiento. Además el

servidor se debe encargar de que cuando el usuario se desconecte o conecte

todos sus suscriptores reciban una actualización de presencia.

43

Aun así los clientes son libres de enviar de por sí actualizaciones de

presencia a otros usuarios. Sin embargo si esto se realiza, el cliente deberá

gestionar que actualiza la presencia de todos sus contactos, mientras que si lo

hace a través del roster, sólo le enviará la actualización de presencia al

servidor y será éste quien lo gestione.

El protocolo roster es una extensión del protocolo IQ, existen tres tipos

básicos de protocolos roster:

o Roster get: usado por los clientes para obtener una copia del roster

almacenado en el servidor.

o Roster update: usado por los clientes para actualizar el roster

almacenado en el servidor.

o Roster push: actualizaciones asíncronas del roster que el servidor

envía a los clientes.

Vamos a ver mejor cada tipo del protocolo roster.

6.3.1. Roster get

Permite a los clientes obtener una copia completa del roster almacenado

en su servidor Jabber/XMPP. Para ello se debe enviar un mensaje IQ de tipo

get vacío especificando el protocolo “jabber:iq:roster”.

<iq type='get' id='roster\_get\_id'>

<query xmlns='jabber:iq:roster'/>

</iq>

Y el servidor contestará con una copia completa del roster.

<iq type='result' id='roster\_get\_id'>

<query xmlns='jabber:iq:roster'>

<item jid='sub1ID' name='nickname1'

subscription='both'>

<group>Personal</group>

<group>Trabajo</group>

</item>

<item jid='sub2ID' name='nickname2'

subscription='from'>

<group>Trabajo</group>

<group>Familia</group>

<group>Equipo de Futbol</group>

</item>

</query>

</iq>

Hay que tener en cuenta que el paquete <query> puede tener cero o más

paquetes <item>. Cada paquete <item> representa una suscripción de un

contacto, y ese contacto pertenecerá a cero o más grupos, identificados por

cero o más paquetes <group>. La etiqueta name es opcional aunque

44

generalmente siempre se utiliza para que el cliente muestre un nombre en los

contactos en lugar del Jabber ID del contacto.

6.3.2. Roster update

Los clientes pueden modificar el roster enviando un paquete IQ de tipo set

conteniendo dentro del paquete <query> el item a actualizar.

<iq type='set' id='roster\_set\_id'>

<query xmlns='jabber:iq:roster'>

<item jid='sub1ID' name='Pepe'>

<group>Trabajo</group>

<group>Familia</group>

<group>Equipo de Baloncesto</group>

</item>

</query>

</iq>

El servidor responderá con un paquete IQ de tipo respuesta indicando que

la actualización se ha realizado correctamente o un paquete IQ de error en

caso contrario. El servidor enviará los cambios a todos los clientes

autentificados usando el siguiente protocolo.

6.3.3. Roster push

Es enviado por el servidor a todos los clientes que se han conectado con

el mismo Jabber ID tras una modificación en el roster por alguno de ellos o por

la modificación del tipo de suscripción de presencia de alguno de los contactos.

Son los únicos mensajes IQ que no obtienen respuesta, sólo van del servidor a

los clientes y éstos no tienen que responder. El roster push para la

actualización anterior sería la siguiente.

<iq type='set'>

<query xmlns='jabber:iq:roster'>

<item jid='sub1ID' name='Pepe' subscription='both'>

<group>Trabajo</group>

<group>Familia</group>

<group>Equipo de Baloncesto</group>

</item>

</query>

</iq>

Si un cliente borra su suscripción del roster usando el protocolo de

suscripción, el roster item de esa suscripción sería borrado del servidor. El

servidor enviaría un roster push a todos los clientes indicando lo acontecido

con un mensaje IQ de tipo set y poniendo el atributo subscription con el valor

“remove”. El cliente entonces borraría ese item de entre los contactos que está

mostrando.

45

Por ejemplo si yo me desuscribo de “sub1ID” y el hace lo mismo conmigo

yo recibiría el siguiente roster push del servidor.

<iq type='set'>

<query xmlns='jabber:iq:roster'>

<item jid='sub1ID' subscription='remove'/>

</query>

</iq>

46

7. Autentificación SASL

XMPP incluye autentificación mediante SASL (Simple Authentication and

Security Layer), como ya se ha comentado antes en los inicios del protocolo

Jabber la autentificación se realizaba mediante el protocolo jabber:iq:auth, pero

hoy en día SASL es quien se lleva la palma. SASL provee a Jabber/XMPP de

un método generalizado para la autentificación.

Para ello se han aplicado ciertas normas en la autentificación:

o Si la autentificación SASL se da entre dos servidores, la

comunicación no se establecerá hasta que se aseguren de la

auténtica DNS del otro servidor, véase la comunicación entre

servidores (Server-to-Server) del siguiente apartado.

o Si quien quiere autentificarse soporta SASL deberá incluir el atributo

“version” con el valor “1.0” por lo menos, en la cabecera del stream

inicial.

o Si el servidor soporta SASL deberá informar de sus tipos de

autentificaciones con la etiqueta <mechanisms/> en la contestación

de la etiqueta de inicio de sesión, si es que el cliente soporta la

conexión SASL.

o Durante la negociación SASL, ninguno de los dos deberá enviar

algún carácter en blanco como separación entre elementes, esta

prohibición ayuda a asegurar la precisión a nivel de byte.

o Cualquier carácter XML contenido en los elementos XML deberá

estar codificado usando base64.

El proceso de autentificación mediante SASL sería el siguiente:

1. La entidad que pida una autentificación SASL deberá incluir el

atributo “version” el la etiqueta de inicio de sesión enviada al servidor

con el valor “1.0” como mínimo.

2. Cuando el servidor recibe la etiqueta de inicio de sesión con el

atributo “version” deberá comunicar los tipos de autentificación SASL

que implementa, cada uno de ellos irá dentro de un hijo del tipo

<mechanisms/>.

3. El cliente deberá seleccionar uno de los mecanismos enviando el

elemento <auth/> con un valor adecuado para el mecanismo de

autentificación SASL elegido. El elemento contendrá caracteres XML

(en la terminología de SASL, la respuesta inicial “initial response”). Si

el cliente debe responder con un elemento vacío responderá con el

carácter ‘=’, que indicará que la respuesta no contiene datos.

47

4. Si fuera necesario, el servidor enviará el elemento <challenge/> al

cliente que contendrá datos en formato XML, esto dependerá del tipo

de autentificación SASL que el cliente haya elegido.

5. El cliente responderá al “desafío” enviando la etiqueta <response/> al

servidor.

6. Si fuera necesario el servidor enviaría más elementos “challenge” y el

cliente respondería a los mismos.

Esta serie de challenge/response continuaría hasta que ocurriera una de

estas tres cosas:

o Que el cliente que quería autentificarse aborte la autentificación

enviando la etiqueta <abort/> al servidor. En cuyo caso el servidor

dejará al cliente enviar un número configurable de peticiones más,

normalmente dos, antes de cerrar la conexión TCP. Así el cliente

podrá volver a autentificarse sin necesidad de reiniciar la sesión

como pasaba con el protocolo Jabber original.

o Que el servidor responda con la etiqueta <failure/>, con la que

comunicaría al cliente que la autentificación ha fallado. Como en el

caso anterior le dejará enviar un limitado número de peticiones más

para que si lo desea vuelva a intentarlo.

o Que el servidor responda con la etiqueta <sucess/>, ocn la que

comunicaría al cliente que la autentificación se ha realizado

correctamente, y además contendría datos en formato XML

dependiendo del tipo de autentificación SASL. Una vez realizada la

autentificación el cliente deberá enviar una etiqueta vacía de inicio de

sesión, sin necesidad de cerrar la sesión anterior, a la que contestará

el servidor y comenzará la conexión.

La autentificación SASL puede generar diferentes errores:

o <aborted/>: autentificación abortada, explicada anteriormente.

o <incorrect-encoding/>: los datos enviados por el cliente no pueden

ser procesados por el servidor debido a que la codificación base64 es

incorrecta.

o <invalid-authzid/>: el authzid dado por el cliente es inválido debido a

que está mal formado o a que el cliente no tiene permisos.

o <invalid-mechanism/>: el cliente tiene o solicita un mecanismo que no

está implementado en el servidor.

o <mechanism-too-weak>: el mecanismo está considerado como débil

por la política de seguridad del servidor.

48

o <not-authorized/>: la autentificación ha fallado debido a que el cliente

no tiene credenciales válidos.

o <temporary-auth-failure>: la autentificación ha fallado debido a un

error temporal del servidor.

Veamos un ejemplo de autentificación SASL entre un cliente y un

servidor :

1. El cliente envía la etiqueta de inicio de sesión al servidor:

<stream:stream

xmlns='jabber:client'

xmlns:stream='http://etherx.jabber.org/streams'

to=‘ejemplo.com’

version='1.0'>

2. El servidor responde al cliente con la etiqueta de inicio:

<stream:stream

xmlns='jabber:client'

xmlns:stream='http://etherx.jabber.org/streams'

id='c2s\_234'

from=‘ejemplo.com’

version='1.0'>

3. El servidor informa al cliente de los mecanismos disponibles:

<stream:features>

<mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>

<mechanism>DIGEST-MD5</mechanism>

<mechanism>PLAIN</mechanism>

</mechanisms>

</stream:features>

4. El cliente selecciona el mecanismo deseado:

<auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl'

mechanism='DIGEST-MD5'/>

5. El servidor envía el challenge al cliente en base64:

<challenge xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>

cmVhbG09InNvbWVyZWFsbSIsbm9uY2U9Ik9BNk1HOXRFUUdtMmhoIixxb3A9ImF1

dGgi

LGNoYXJzZXQ9dXRmLTgsYWxnb3JpdGhtPW1kNS1zZXNzCg==

</challenge>

El challenge decodificado es:

realm="somerealm",nonce="OA6MG9tEQGm2hh",\

qop="auth",charset=utf-8,algorithm=md5-sess

49

Aunque el servidor podría haber respondido con un error:

<failure xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>

<incorrect-encoding/>

</failure>

</stream:stream>

6. El cliente envía la respuesta en base64:

<response xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>

dXNlcm5hbWU9InNvbWVub2RlIixyZWFsbT0ic29tZXJlYWxtIixub25jZT0i

T0E2TUc5dEVRR20yaGgiLGNub25jZT0iT0E2TUhYaDZWcVRyUmsiLG5jPTAw

MDAwMDAxLHFvcD1hdXRoLGRpZ2VzdC11cmk9InhtcHAvZXhhbXBsZS5jb20i

LHJlc3BvbnNlPWQzODhkYWQ5MGQ0YmJkNzYwYTE1MjMyMWYyMTQzYWY3LGNo

YXJzZXQ9dXRmLTgK

</response>

La respuesta decodificada es:

username="somenode",realm="somerealm",\

nonce="OA6MG9tEQGm2hh",cnonce="OA6MHXh6VqTrRk",\

nc=00000001,qop=auth,digest-uri="xmpp/example.com",\

response=d388dad90d4bbd760a152321f2143af7,charset=utf-8

7. El servidor envía otro challenge codificado al cliente:

<challenge xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>

cnNwYXV0aD1lYTQwZjYwMzM1YzQyN2I1NTI3Yjg0ZGJhYmNkZmZmZAo=

</challenge>

El challenge decodificado es:

rspauth=ea40f60335c427b5527b84dbabcdfffd

El servidor también podría haber respondido con un error:

<failure xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>

<temporary-auth-failure/>

</failure>

</stream:stream>

8. El cliente responde:

<response xmlns='urn:ietf:params:xml:ns:xmpp-sasl'/>

9. El servidor le dice al cliente que la autentificación ha sido

correcta:

<success xmlns='urn:ietf:params:xml:ns:xmpp-sasl'/>

El servidor podría haber respondido con error:

<failure xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>

<temporary-auth-failure/>

</failure>

50

</stream:stream>

10. El cliente envía una nueva etiqueta de inicio de sesión:

<stream:stream

xmlns='jabber:client'

xmlns:stream='http://etherx.jabber.org/streams'

to=‘ejemplo.com’

version='1.0'>

11. El servidor responde al cliente con una cabecera de sesión con

más opciones (o si no con la etiqueta “features” vacía):

<stream:stream

xmlns='jabber:client'

xmlns:stream='http://etherx.jabber.org/streams'

id='c2s\_345'

from=‘ejemplo.com’

version='1.0'>

<stream:features>

<bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'/>

<session xmlns='urn:ietf:params:xml:ns:xmpp-session'/>

</stream:features>

51

8. Comunicación Server-to-Server (jabber:server)

Hasta ahora sólo se han mencionado las comunicaciones entre servidores

sin llegar a profundizar en cómo propone el protocolo Jabber/XMPP su

utilización. El conjunto de protocolos que las implementa no se deberían

implementar en un servidor privado que una intranet empresarial, si lo que

realmente se pretende es que los usuarios del sistema Jabber/XMPP

únicamente puedan hablar con sus compañeros, es decir, con los demás

clientes de su dominio.

En una red empresarial con similares propósitos, pero con más de un

dominio de redes Jabber/XMPP de la empresa, habría que implementar este

protocolo pero siempre habría que amoldarlo para que el servidor hiciera un

buen uso del mismo y sólo se conectara con otros servidores de la empresa, y

no con servidores públicos en donde puedan tener contactos los trabajadores

de la empresa.

Las comunicaciones entre clientes y servidores se basan siempre en el

conjunto de protocolos jabber:client, mientras que este tipo de comunicaciones

entre servidores se basarán en el conjunto jabber:server.

Existen dos grandes diferencias entre las comunicaciones entre clientes y

servidores y las comunicaciones entre servidores. Mientras que en las primeras

la comunicación es bidireccional las comunicaciones entre servidores son

unidireccionales, es decir, aunque en realidad se establezcan conexiones

mediante socket’s bidireccionales este tipo de comunicaciones sólo permiten

enviar paquetes al servidor que ha iniciado la conversación. El servidor

receptor sólo enviará de vuelta sus paquetes XML y mensajes de error. Si los

servidores necesitarían enviar paquetes en ambas direcciones deberían crear

dos conexiones diferentes.

La segunda gran diferencia es que en las comunicaciones entre

servidores los campos to y from deben estar perfectamente rellenados, ya que

el servidor destino no conoce al cliente que ha enviado ese paquete al servidor

que se lo está mandando, por lo tanto será el servidor del dominio del cliente el

responsable de rellenar cualquiera de estos campos.

Así, si el usuario conectado al servidor del dominio a.com quisiera enviar

un mensaje al usuario b del dominio Jabber/XMPP b.com el mensaje original

del cliente podría ser:

<message to='b@b.com'>

<body>Aupa!</body>

</message>

El servidor a.com recibe el mensaje, y lo edita para agregar el Jabber ID

del emisor para poder conectarse con el servidor b.com y podérselo enviar.

52

<message to='b@b.com' from='a@a.com'>

<body>Aupa!</body>

</message>

El servidor b.com recibe el mensaje, encuentra al usuario b en línea y

depende de la implementación del servidor podría volver a modificar el mensaje

para quitarle el destinatario y ahorrar ancho de banda en el envío del mensaje:

<message from='a@a.com'>

<body>howdy</body>

</message>

Cuando un servidor Jabber/XMPP actúa en representación de cualquiera

de sus usuarios en una conexión S2S la seguridad en la conexión es algo muy

importante. Cualquier servidor podría usurpar la identidad de un servidor local y

hacerse con los mensajes de todos los usuarios que pertenezcan al dominio

que estaría usurpando en una red. Por ello existe la necesidad de que exista

un protocolo de autentificación también en las conexiones entre servidores

S2S.

8.1. Dialback authentication

La autentificación entre servidores Jabber/XMPP es mucho más

complicada que entre los clientes y su servidor. En las autentificaciones entre

un cliente y su servidor, sólo los clientes con cuenta de usuario en ese servidor

pueden acceder a autentificarse. Sin embargo en las conexiones S2S, los

servidores Jabber/XMPP van a tener que autentificarse muchas veces con

servidores con los que nunca han tenido un contacto y que no conocen.

Además, como el número de servidores Jabber/XMPP podría llegar a ser

potencialmente grande, los servidores no pueden tener una cuenta para cada

uno de los otros servidores.

Este protocolo ha sido creado para crear un simple mecanismo de

autentificación para asegurarse de que cada servidor es quien dice ser, sin

conocerlo previamente. Para ello el servidor que se quiere autentificar envía

una clave al servidor en el que se quiere autentificar, éste para comprobar que

el servidor que desea autentificarse es quien dice ser se conectará al dominio

del servidor que dice ser y lo confirmará con la clave que ha recibido,

supuestamente de ese mismo servidor. Si es así el servidor le enviará una

contestación informando que todo es correcto y que está autentificado, y este a

su vez le enviará una contestación al servidor que está autentificándose con él,

aceptando su conexión.

Esto es igual que en la vida real cuando una persona quiere identificar a

un agente, si nos dice su nombre y número de placa y se lo preguntamos a su

central a través de su radio, esto no sería seguro porque si no fuera realmente

un agente podríamos estar hablando con un amigo suyo, sin embargo si

llamamos a la central desde nuestro teléfono móvil y nos lo confirman sabemos

que quien nos lo está confirmando es realmente alguien desde la central.

53

En el protocolo se confía en el servidor de DNS, que será quien nos va a

decir la dirección real de ese servidor, y la utilizaremos para preguntarle si el se

está intentando autentificar con nosotros o ha sido un impostor.

Aun así este protocolo no es del todo seguro ya que la persona que

intentara suplantar a otro servidor podría atacar al servidor de DNS para así

registrar su IP en lugar de la del servidor real del domino, o incluso suplantar al

DNS con técnicas de “Man in de Middle”.

La autentificación entre el servidor origen.com y el servidor destino.com

sería de la siguiente forma, origen.com se conectaría a destino.com enviando

la etiqueta de inicio de sesión e informando de que soporta la extensión

jabber:server:dialback de autentificación:

<stream:stream xmlns:stream='http://etherx.jabber.org/streams'

xmlns='jabber:server'

to='destino.com'

from='origen.com'

xmlns:db='jabber:server:dialback'>

El servidor destino.com responde con la etiqueta de inicio de sesión,

informando que él también soporta la autentificación “dialback” y asigna un

identificador a la sesión:

<stream:stream xmlns:stream='http://etherx.jabber.org/streams'

xmlns='jabber:server'

to='origen.com'

from='destino.com'

xmlns:db='jabber:server:dialback'

id='4208ab093e'>

Así, origen.com enviará el mensaje con la clave de autentificación como

respuesta ya que ambos servidores implementan el protocolo:

<db:result to='destino.com'

from='origen.com'>0283cd322312</db:result>

Con esta clave destino.com ya puede iniciar el protocolo de autentificación

y así deberá buscar a origen.com en su servidor de DNS y conectarse a la

dirección que éste le indique, para así iniciar la conversación con el origen.com

real:

<stream:stream xmlns:stream='http://etherx.jabber.org/streams'

xmlns='jabber:server'

to='origen.com'

from='destino.com'

xmlns:db='jabber:server:dialback'>

El servidor origen.com soporta el protocolo de autentificación por lo que

responderá con la etiqueta de comienzo de sesión y asignará un identificador

para la sesión:

<stream:stream xmlns:stream='http://etherx.jabber.org/streams'

xmlns='jabber:server'

to='destino.com'

from='origen.com'

xmlns:db='jabber:server:dialback'

54

id='403a33b093e'>

El servidor destino.com procederá a enviar la clave que recibió del

supuesto origen.com:

<db:verify to='origen.com'

from='destino.com'

id='5423ef'>

0283cd322312

</db:verify>

El servidor origen.com determinará si existe una sesión abierta con

destino.com y si la clave es correcta. Si es así, devolverá la confirmación a

destino.com:

<db:result to='destino.com'

from='origen.com'

type='valid'

id='5423ef'/>

El servidor destino.com enviará la confirmación de autentificación a la

primera sesión que había iniciado origen.com para autentificarse, por lo que la

conexión se podrá realizar y origen.com quedaría así autentificado.

<db:result to='origen.com'

from='destino.com'

type='valid'/>

Si origen.com no reconocería la clave respondería a destino.com con una

autentificación incorrecta:

<db:result to='destino.com'

from='origen.com'

type='invalid'

id='5423ef'/>

Y éste a su vez no aceptaría la conexión inicial de origen.com ya que

sería un impostor:

<db:result to='origen.com'

from='destino.com'

type='invalid'/>

La clave y su formato no están especificados en las especificaciones del

protocolo Jabber/XMPP. El servidor de referencia “jabberd” utiliza una

codificación SHA-1 de tipo digest de partes de los nombres de los servidores

para generar y verificar este tipo de claves. Así pues cada servidor podría tener

una forma de generar este tipo de claves, pero la autentificación sería correcta

entre dos servidores que implementaran diferentes tipos de contraseñas, ya

que quien la genera y la confirma es el mismo servidor, y no importa como se

haga.

55

9. Transporte entre Jabber/XMPP y otros servidores de

MI

Debido a que Jabber/XMPP es un protocolo libre basado en el paso de

paquetes en formato XML los sistemas Jabber/XMPP están preparados para el

uso como un sistema genérico de transporte de MI. Su simple diseño ha sido

explotado por servidores Jabber/XMPP para conectarse con otros sistemas de

MI como pueden ser el MSN Messenger y Yahoo! Messenger.

El servidor Jabber/XMPP de referencia jabberd utiliza módulos llamados

transports que proveen un puente entre Jabber/XMPP y los demás sistemas de

MI. Los transportes tratan a cada sistema de MI propietario como un dominio

Jabber/XMPP con sus propios clientes identificándolos por su “Jabber ID”, que

en realidad sería el nombre del usuario en el sistema externo. Al enviar un

mensaje Jabber/XMPP a uno de esos módulos esos Jabber IDs especiales

hacen que sean transportados por el módulo de transporte.

Los módulos de transporte conectan con el sistema de MI correspondiente

y actúan como clientes de ese servidor para intercambiar mensajes y presencia

entre los dos sistemas.

Por ejemplo, si se envía un mensaje a usuario@messenger.server.com el

servidor reconocerá que messenger.server.com es un usuario de otro sistema y

se lo enviará al módulo de transporte para que procese el paquete. El

transporte iniciará la sesión en la red MSN Messenger y enviará el mensaje al

interesado. Cada transporte deberá parsear los mensajes Jabber/XMPP a

mensajes con el formato del sistema de MI correspondiente.

En los casos en que el sistema esté bien documentado como puede ser el

caso de IRC esto es algo sencillo y ya está realizándose sin ningún problema.

El problema radica en los sistemas de MI propietarios como el MSN Messenger

a los que este intercambio de mensajes no les conviene para mantener atados

a sus clientes, por lo que tratan de cambiar el protocolo de manera regular para

así tener que volver a realizar trabajos de ingeniería inversa los desarrolladores

de los módulos de transporte de cada servidor para poder adaptarlos a los

cambios y que sigan funcionando correctamente.

56

10. Chatbots

Un chatbot es una aplicación que autónomamente se comporta como otro

usuario del sistema Jabber/XMPP. Cuando un usuario envía mensajes a un

chatbot, éste automáticamente genera sus propias respuestas. Los

desarrolladores crean chatbots para resolver muy diversas necesidades,

algunas son para mero ocio como pueden ser los chatbot que cuentan chistes.

Pero sin embargo los chatbot pueden dar servicios mucho más útiles

como noticias meteorológicas. Otros proveen de un traductor a los clientes, que

envían mensajes en su idioma y el chatbot responde con las traducciones. Así

los usuarios pueden ir traduciendo la conversación que están estableciendo

con otra persona que habla un idioma que no conocen.

Los chatbots amplían las posibilidades de Jabber/XMPP hasta límites

insospechados, y son simples programas automatizados que hacen uso del

protocolo nativo de Jabber/XMPP, sin necesidad de ningún otro cambio en el

protocolo para sus actuaciones. Esta es una grandeza de Jabber/XMPP, de la

que ningún otro sistema de MI puede presumir.

Por lo general no tienen interfaz, simplemente se comunican con otros

clientes que reclaman sus servicios vía Jabber/XMPP. Estos programas

pueden utilizarse además como:

o Soporte al consumidor: los chatbots pueden resolver dudas y

problemas frecuentes en muchos ámbitos comerciales, dando una

respuesta rápida y eficaz a problemas usuales y catalogados por el

proveedor.

o Servicios de información: pueden proveer de información a otros

usuarios. La información puede ser de muchos tipos como alertas

del tiempo, ofertas comerciales, noticias…

o Servicios de búsqueda: estos servicios pueden ser del tipo de los

buscadores de Internet habituales como google, guías telefónicas,

bases de datos corporativas, diccionarios, traductores, búsquedas

de artículos…

Como puede observarse nos ofrecen muy útiles y variados servicios que

pueden llegar a convertirse en herramientas imprescindibles para los usuarios

del sistema.

57

11. Jabber/XMPP como middleware

Jabber/XMPP puede verse como un desafortunado competidor de otros

sistemas MOM. Sin embargo, los sistemas Jabber/XMPP pueden tener sus

propios beneficios respecto a los tradicionales sistemas MOM, como pueden

ser:

o Económico: las empresas pueden ver una solución económica en

Jabber/XMPP para el paso de mensajes entre sus aplicaciones.

Especialmente para aplicaciones que no sean criticas.

o XML: como Jabber/XMPP está basado en XML es un maravilloso

middleware para infraestructuras basadas en XML.

o Abierto: el protocolo Jabber/XMPP y muchas de sus

implementaciones son abiertas. Los beneficios del open-software

están perfectamente desarrollados en www.opensource.org.

o Simple: tanto las grandes como las pequeñas empresas pueden

entender y usar fácilmente Jabber/XMPP con un corto aprendizaje.

o Ligero: Jabber/XMPP es un protocolo ligero en una red comparado

con sus competidores. Así el software que lo implementa es

compacto y eficiente.

El uso de Jabber/XMPP como MOM es una cuestión abierta en la

comunidad Jabber/XMPP. Existen dos grupos en www.jabber.org: JAM y

Jabber/XMPP RPC. Ambos están interesados en la integración de

Jabber/XMPP como middleware.

Los desarrolladores de Java están interesados en la integración con JMS

de Jabber/XMPP para su uso como middleware. Además Jabber/XMPP sería

un middleware excelente entre las tecnologías Microsoft .NET y los servidores

J2EE de Java.

58

12. Sitios de interés

o JabberEs, comunidad hispana de Jabber: www.jabberes.org

o Jabber Community: www.jabber.org

o Jabber Inc.: www.jabber.com

o Jabber Powered: www.jabberpowered.org

59

13. Agradecimientos

La realización de este documento ha sido posible gracias a la inestimable

colaboración de Luís Peralta que ha colaborado con entusiasmo en las tareas

de revisión y corrección del manual. Tengo que agradecer también la

colaboración de Badlop en las tareas de revisión.

Tengo que agradecer también a Natalia Vicandi su esfuerzo por animarme

siempre a conseguir lo que me propongo. También la gratificante ayuda de mis

compañeros Oihane Méndez, Itxaso Dieguez e Iñaki Larrañaga. Además

agradecer los sabios consejos de Javier Vicente y Amaia Méndez.