ISSN 0798 1015

REVISTA ESPACIOS

HOME Revista ESPACIOS ÍNDICES A LOS AUTORES

Vol. 38 (Nº 59) Año 2017. Pág. 6

Arquitectura de integración basada en socket para sistemas distribuidos

Integration architecture for distributed systems based on socket

Oiner GÓMEZ-BARYOLO 1; Alleiny MACHADO-SOSA 2; Alexis CABRERA-MONDEJA 3

Recibido: 05/08/2017 • Aprobado: 03/09/2017

Contenido

RESUMEN:
Las nuevas tendencias de la ingeniería de software apuntan al desarrollo de sistemas distribuidos implementados en diferentes plataformas. En el presente artículo se propone una arquitectura de integración basada en socket para sistemas distribuidos que requieren comunicación en tiempo real. Está basada en herramientas y tecnologías libres que facilitan la integración sobre el protocolo TCP entre plataformas móviles y web. La propuesta se implementa en un estudio de caso que facilita la comprensión de la arquitectura y valida su funcionamiento.
Palabras clave Arquitectura de integración, socket, plataformas móviles, plataformas web

ABSTRACT:
The new trends in software engineering point to the development of distributed systems implemented in different platforms. This article proposes a socket - based integration architecture for distributed systems that require real - time communication. It is based on free tools and technologies that facilitate integration between mobile and web platforms through the TCP protocol. The proposal has been implemented for a case study that facilitates the understanding of the architecture and the validation of its operationkey words
Keywords Integration architecture, socket, mobile platforms, web platforms

1. Introducción

Con el desarrollo intelectual del ser humano va avanzando a pasos agigantados la tecnología, en estos tiempos se presenta una era donde la computación no es consumida des de un escritorio sino ya acepta la realidad de que un ser humano se mantiene en constante movimiento (Freitas, 2015). Con esto empieza a surgir una era móvil donde las personas cada vez más realizan sus actividades diarias no solo desde una laptop sino cada vez se hace más frecuente la presencia de dispositivos celulares inteligentes y de tablets, por su fácil acceso y portabilidad (Nielson Avelino de Santana, 2016). Entre las tecnologías más utilizadas con este fin se puede resaltar la web y las aplicaciones móviles nativas, integradas por bibliotecas

adaptables a cualquier resolución de pantalla y navegador. Para ello se integran lenguajes y tecnologías como el HTML5, CSS3, JavaScript, Java, Swift, sockets y otras que agilizan el proceso de desarrollo y flexibilizan el uso de los sistemas (Vera & Rodríguez, 2016).

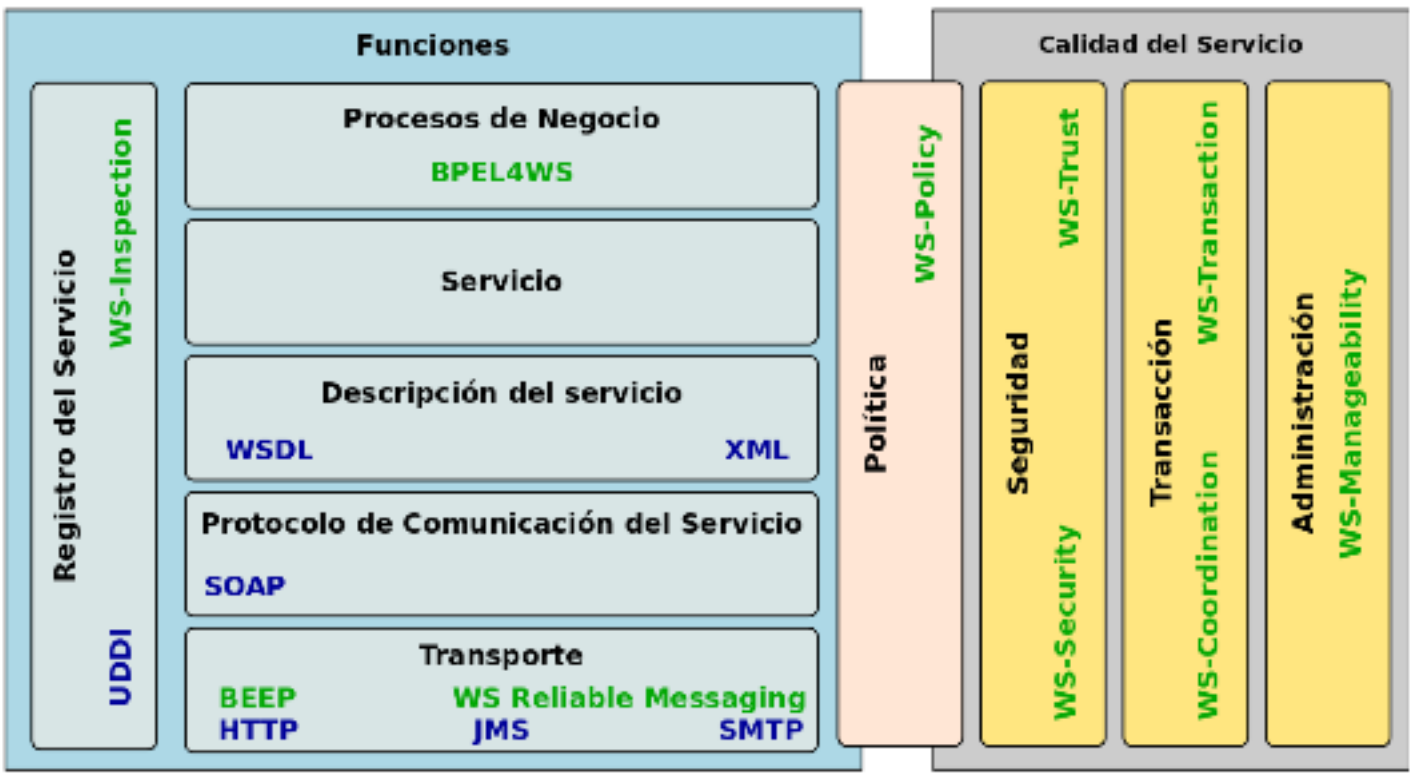
Los cambios producidos están relacionados en gran medida con el desarrollo de las Tecnologías de la Información y las Comunicaciones (TIC) y su utilización en las diferentes esferas de la sociedad (Balbinot, 2011). Como parte del desarrollo de las TIC y de la necesidad de alcanzar una cultura de gestión de información y conocimiento, se fortalece cada día la relación entre la gestión de información, las tecnologías y los Sistemas de Información (SI). Desde el punto de vista práctico un SI está compuesto por un conjunto de procesos, datos, modelos, tecnologías y personas que responden a una estructura coherente en función del propósito de una organización. Los SI están compuestos por componentes, muchos de ellos desarrollados en diferentes plataformas y alojados en distintos lugares (Merchán & Hernández, 2016).

1.1. Arquitectura orientada a servicios

Para lograr la comunicación y el intercambio de información entre los diferentes componentes se implementan protocolos y arquitecturas de comunicación como los Servicios Web basados en el Protocolo Simple de Acceso a Objetos (SOAP por sus siglas en inglés), también basados en la Transferencia de Estado Representacional (REST por sus siglas en inglés), comunicación basada en Socket, entre otras (Sánchez, Clemente, Prieto, Conejero, & Echeverría, 2017).

La W3C define los Servicios Web como: "*un conjunto de aplicaciones o de tecnologías con capacidad para interoperar en la Web. Estas aplicaciones o tecnologías intercambian datos entre sí con el objetivo de ofrecer unos servicios. Los proveedores ofrecen sus servicios como procedimientos remotos y los usuarios solicitan un servicio llamando a estos procedimientos a través de la Web*" (W3C, 2007). La Figura: 1 refleja el modelo de madurez de SOA para la integración de sistemas distribuidos:

Figura 1
Modelo de madurez de SOA [4]



Los servicios web permiten la comunicación a través de mensajes que adoptan el Lenguaje de Marcado Extensible (XML por sus siglas en inglés) que responden al Lenguaje de Descripción de Servicios Web (WSDL por sus siglas en inglés) y son enviados sobre el protocolo SOAP (OASIS, 2006). A continuación, se refleja especificación de un servicio web aplicando el estándar WSDL:

<? xml version=**'1.0'** encoding=**'UTF-8'**?>

```
<definitions name="" targetNamespace="" xmlns:typens="" xmlns:xsd="" xmlns:soap="" xmlns:soapenc="" xmlns:wsdl="" xmlns="">
   <message name="">
      <part name="" type=""/>
   </message>
   <portType name="">
      <operation name="">
         <documentation>  </documentation>
         <input message=""/>
         <output message=""/>
      </operation>
   </portType>
   <binding name="" type="">
      <soap:binding style="" transport=""/>
      <operation name="">
         <soap:operation soapAction=""/>
         <input>
            <soap:body namespace="" use="" encodingStyle=""/>
         </input>
         <output>
            <soap:body namespace="" use="" encodingStyle=""/>
         </output>
      </operation>
   </binding>
   <service name="">
      <port name="" binding="">
         <soap:address location=""/>
      </port>
   </service>
</definitions>
```

Como es posible apreciar el servicio web descrito anteriormente, está implementado en una clase y un método determinado que deben cumplir determinados parámetros para que puedan ser publicados en el estándar WSDL. Una vez publicado la ruta del WSDL estarán listos para ser consumido desde diferentes lenguajes, a continuación, se muestra un ejemplo usando lenguaje PHP:

```
//Creando un cliente de soap bajo el estándar <wsdl>
   $wsdl = 'URL del WSLD';
   $soapCliente = new SoapClient($wsdl, array());
//Consumo del servicio web
   $respuesta = $soapCliente->nombre_mensaje();
```

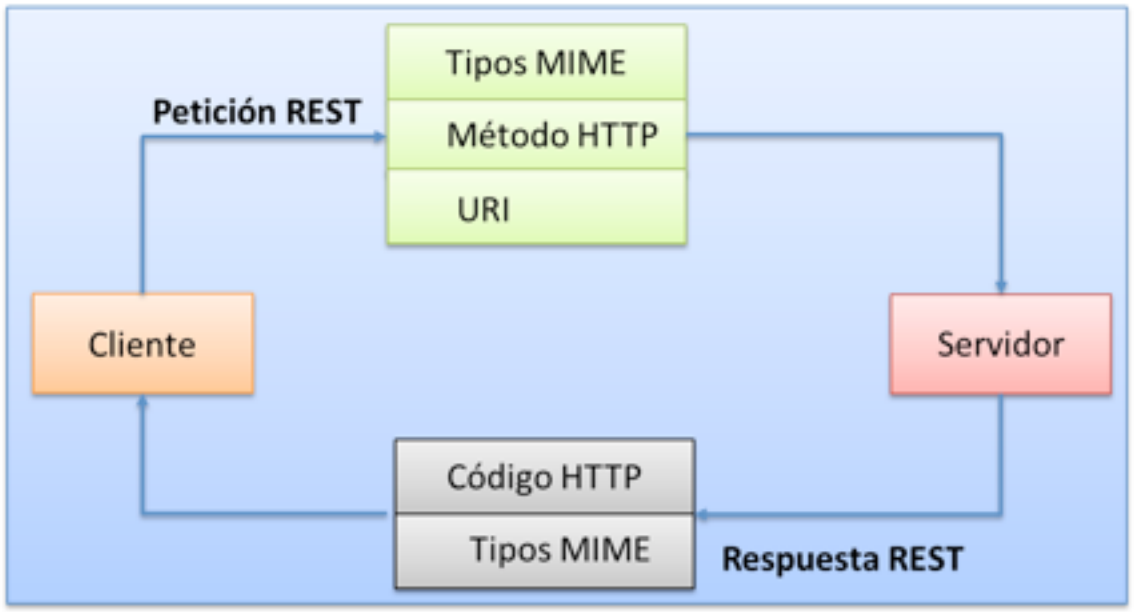
La principal limitante de esta tecnología radica en su complejidad de implementación, debido a que los sistemas deben ser desarrollados bajo esta concepción desde el inicio y tener como

base una infraestructura tecnológica sólida. En muchos casos la comunicación se torna lenta, producido por el nivel de procesamiento que se necesita para validad y encapsular los mensajes en los estándares definidos. Los mensajes intercambiados se quedan a nivel de la capa de negocio, requiriendo la integración con otras tecnologías de mensajería instantánea como el Protocolo extensible de mensajería y comunicación de presencia (XMPP por sus siglas en ingles), protocolo IRC, sockets, etc.

1.2. Arquitectura REST

La arquitectura de Transferencia de Representación de Estado (REST por sus siglas en inglés) es otra de las alternativas más implementadas en la actualidad desde el año 2000 que fue publicada por Roy Fielding. Los principios fundamentales sobre los que se basa REST son: el dinamismo de las interfaces, la robustez en la comunicación entre sus componentes, la independencia de la plataforma y la compatibilidad con otras arquitecturas y tecnologías. Lo logra identificando dinámicamente los recursos que interactúan y realizando su representación mediante el uso de URLs o mensajes autodescriptivos que son interpretados por métodos como el POST o GET. En la Figura: 2 se especifica el flujo de información en la informatización de un proceso utilizando el protocolo REST:

Figura 2
Flujo de comunicación REST (elaboración propia)



Para gestionar los recursos, el protocolo HTTP provee los siguientes métodos con los que se debe ejecutar las operaciones (Echeverría, Macías, Pavón, Conejero, & Figueroa, 2015):

- GET: Para consultar y leer recursos
- POST: Para crear recursos
- PUT: Para editar recursos
- DELETE: Para eliminar recursos
- PATCH: Para editar partes concretas de un recurso

La publicación y consumo de los servicios REST requiere la implementación de URLs únicas para cada recurso:

- Generación de pares de URLs por recurso
  - */recurso*
  - */recurso/{id}*
- Por cada relación de *uno-a-muchos* generarun par de URLs bases
  - /recurso_uno/*{id}/recurso_mucho*
  - /recurso_uno/*{id}/recurso_mucho/{id}*

<lista-recursos xmlns='**HTTP://ejemplo.org/mi-ejemplo-ns/**'>

<recurso-ref href=**"URI del primer recurso"**/>

<recurso-ref href="**URI del recurso #2**"/> Nombre completo </recurso>

.

.

<recurso-ref href="**URI del recurso #N**"/>Nombre completo</recurso>

</lista-list>

Las potencialidades de REST lo convierten en una de las opciones más utilizadas en la actualidad para la publicación y consumo de servicios web en la nube. No obstante, al ejecutarse sobre el protocolo HTTP presenta vulnerabilidades que impiden su implementación en escenarios que requieran políticas de control de acceso robustas. Si se desea aplicar en sistemas de procesamiento y visualización en "*tiempo real*" o baja latencia, será necesaria la integración con protocolos y tecnologías como XMPP, IRC, Socket, etc.

Los antecedentes descritos permiten evidenciar que cada una de las opciones analizadas se aplicará en función de la complejidad, infraestructura y robustez que se desee alcanzar. La mayor limitante que presentan ambas propuestas, radica en la ausencia de protocolos nativos de actualización de datos en los navegadores web y aplicaciones móviles. Para ello se requiere la integración con otras tecnologías para que cumplan esta función, complejizando el desarrollo, despliegue y mantenimiento de los sistemas.

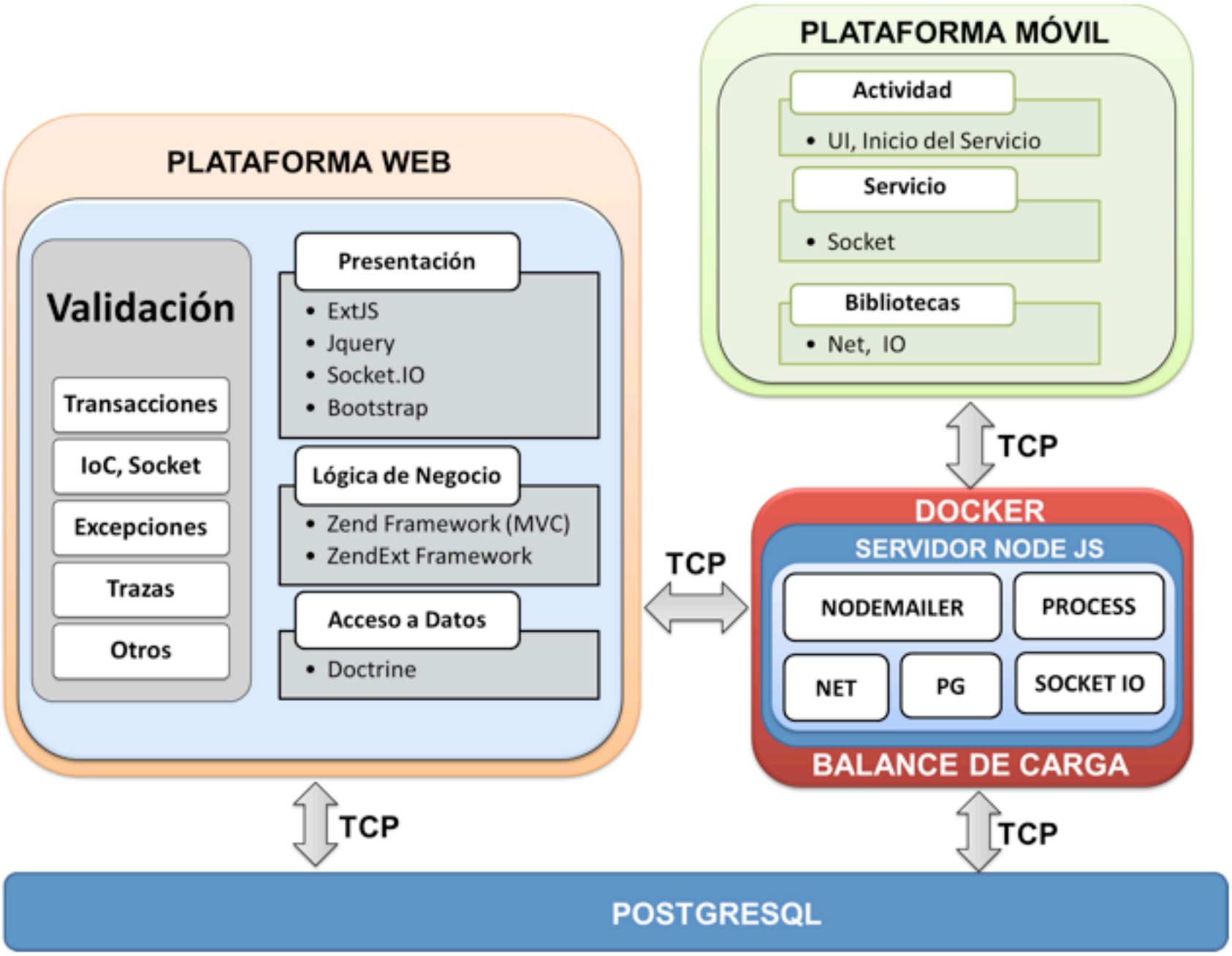
Por lo expuesto anteriormente, el objetivo del presente trabajo radica en el desarrollo de una arquitectura de integración para sistemas distribuidos basada en socket, que permita enviar, recibir y actualizar datos en componentes web y móviles cercano al tiempo real.

---

2. Arquitectura de integración basada en socket

La Arquitectura de software es una práctica que se ha convertido en un factor de vital importancia para lograr que los sistemas de software tengan un alto nivel de calidad. Poseer una buena Arquitectura de software es de suma importancia, ya que ésta es la base de todo sistema de software y determina cuáles serán los niveles de calidad asociados al sistema. La arquitectura propuesta está integrada por un conjunto de componentes reutilizables que provean la estructura genérica y el comportamiento para una familia de abstracciones, logrando una mayor estandarización, flexibilidad, integración y agilidad en el proceso de mantenimiento y desarrollos futuros. La Figura: 3 muestra los componentes que integran la arquitectura propuesta y la relación entre ellos.

Figura 3
Arquitectura de integración basada en socket (elaboración propia)



Como es posible apreciar la arquitectura está formada por tres componentes fundamentales que interactúan entre sí a través de socket utilizando el protocolo TCP. Existen dos componentes que interactúan con el gestor de base de datos PostgreSQL para consultar, insertar, modificar y eliminar datos. En las siguientes secciones se describen con detalle cada uno de los componentes de la arquitectura y la forma en que se comunican.

2.1. Plataforma web

Para alcanzar calidad y eficiencia en el proceso de desarrollo de este tipo de SI, es necesario implementar mecanismos que fomenten la reutilización de componentes con el objetivo de ganar en eficiencia y productividad en la industria del software. Una de las alternativas más utilizadas con este fin es la reutilización de un Marco de Trabajo (MT) que cumpla con los elementos tecnológicos y arquitectónicos definidos para la ejecución del proyecto. En el desarrollo del software, un MT es una estructura coherente de soporte definida sobre la cual se pueden organizar y desarrollar SI. Los MT pueden incluir programas, bibliotecas y lenguajes interpretados que garanticen la integración y comunicación entre los diferentes componentes que integran la solución (Rowlands & Bawden, 1999). La arquitectura incorpora en su diseño un MT en el componente Web como se ilustra en la Figura: 4 .

Figura 4
Componente Web

Este MT Web está desarrollado sobre lenguajes como: PHP, JavaScript, CSS, HTML, entre otros. Su diseño arquitectónico está formado por cuatro capas fundamentales: Capa de presentación, capa de negocio, capa de acceso a datos y una capa vertical que implementa componentes que pueden ser instanciados desde cualquiera de las capas anteriores. En cada una de las capas integra otros MT o bibliotecas que facilitan el desarrollo, tales como: Bootstrap, ExtJs, JQuery, Zend Feamework, Doctrine, entre otros.

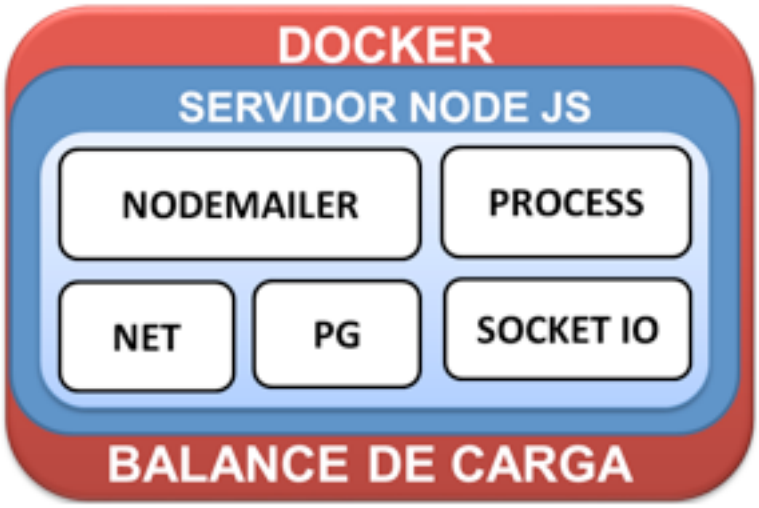
Como parte de la arquitectura de integración, provee diferentes mecanismos, uno de ellos es a través de Socket. Su implementación parte de dos escenarios fundamentales:

- Actualización de datos en “tiempo real” en el navegador Web: puede ocurrir en el momento que se transmita algún mensaje desde la Plataforma Móvil o que se genere algún evento desde el propio Servidor de Node Js que deba ser reflejado en la Web. Para lograrlo el Servidor Node Js crea un Socket Server, instanciando la biblioteca Socket IO que mantiene conectados uno o varios clientes Web que implementan la biblioteca Web Socket para su conexión. En el momento que se genere un evento que deba ser enviado a la Plataforma Web, el Servidor Node Js escribe el mensajea cada uno de los sockets Web conectados.
- Actualización de datos en “tiempo real” en la Plataforma Móvil: se produce cuando se realiza una modificación en la Plataforma Web que debe ser notificada a la Plataforma Móvil. Para ello existen dos formas, la primera de ellas es realizarlo desde el propio navegador Web por medio de Web Socket una vez que se confirme que la operación fue salvada satisfactoriamente en la base de datos y la segunda radica en realizarlo desde el propio Servidor Web por medio de la biblioteca nativa de PHP para el trabajo con socket una vez que se reciba confirmación del gestor de base de datos de que se realizó con éxito la operación.

2.2. Servidor Node JS

El núcleo del servidor de mensajería está implementado sobre un Servidor Node JS que se ha embebido Docker, un contenedor virtual que se puede ejecutar en cualquier servidor Linux. Este componente provee interfaces de comunicación por medio de socket, que permiten el intercambio de datos entre la Plataforma Móvil y la Plataforma Web. La Figura: 5 refleja algunas de las bibliotecas que se instancian en el Servidor Node JS.

Figura 5
Servidor Node JS



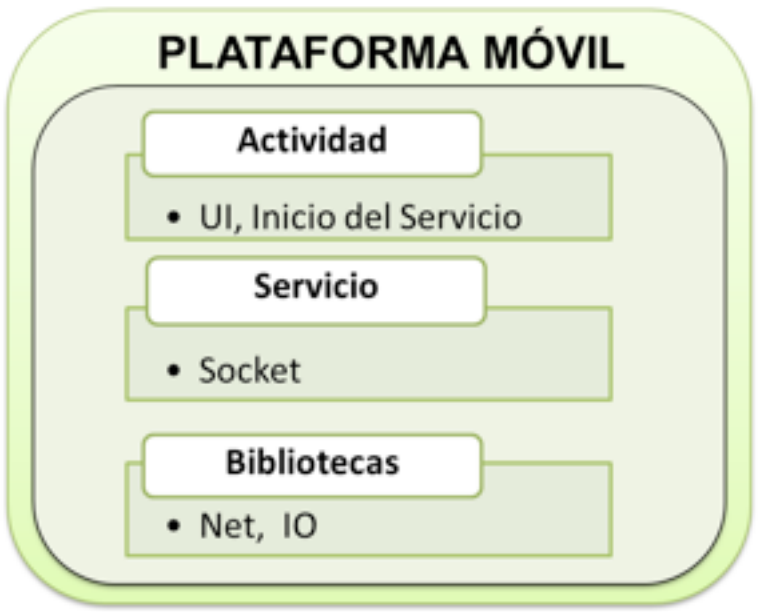
El Servidor Node JS a través de PG implementa un pool de conexiones que permite crear varias instancias de forma dinámica para el acceso al gestor de base de datos y de esta forma balancear la carga de peticiones que impactan positivamente en el rendimiento del sistema. Crea distintas instancias para recibir mensajes de la Plataforma Web, recibir y enviar mensajes a la Plataforma Móvil como se describe a continuación:

- Crea instancias de socket ejecutando la biblioteca NET específicamente para crear socket servidores a los que se conectan las instancias de la Plataforma Móvil y la Plataforma Web, de esta forma recibe mensajes desde el servidor de la Plataforma Web. Permite además recibir y enviar mensajes a la Plataforma móvil en "*tiempo real*".
- Crea instancias de Socket IO para que se conecten los navegadores Web para recibir mensajes desde el Servidor Node JS.

2.3. Plataforma Móvil

Las plataformas móviles como IOS y Android, implementan bibliotecas con clases básicas para conexión a servidores remotos de Internet e intercambio de paquetes. Sin embargo, estas clases no garantizan de manera independiente, una comunicación continua y tolerante a fallos de enlace en ambientes móviles, ya que son propensos a pérdidas de cobertura, cambios automáticos entre redes WIFI, GSM, así como otros factores dependientes de la arquitectura del sistema móvil. Por tal motivo deben ser considerados factores como el ciclo de vida de sus componentes para de esta forma garantizar conexión permanente, además implementar clases específicas que garanticen reconexión y autenticación automáticas. La Figura: 6 muestra la arquitectura estándar de la Plataforma Móvil, la misma puede ser aplicada en el desarrollo de soluciones tanto para IOS como para Android.

Figura 6
Plataforma Móvil



3. Estudio de caso

En esta sección se describe un estudio de caso donde se aplica la arquitectura propuesta en el desarrollo de un sistema de envío y recepción de mensajes en "*tiempo real*". Como se había especificado en la sección anterior, la arquitectura integra tres componentes fundamentales: Plataforma Web, Servidor Node JS y Plataforma Móvil.

3.1. Implementación de la Plataforma Web

La Plataforma Web se implementa sobre herramientas y tecnologías libres como se describen en la Tabla 1 .

Tabla 1
Tecnologías y herramientas

Tecnologías	**Herramientas**
• ExtJs (JavaScript) • Zend Framework (PHP) • Bootstrap (JavaScript) • ORM Doctrine (PHP) • Socket.io (JavaScript) • Socket TCP (PHP)	• PostgreSQL • Apache • PHP • JavaScript • Mozilla Firefox • Google Chrome

En la capa de presentación de la Plataforma Web se incluye la librería Socket.io para crear un cliente Web que se conecte al Servidor Node JS al cargar la página HTML. El siguiente código muestra un ejemplo de la creación de un socket utilizando la librería Web Socket.

```
$(document).ready(function(){
    var socketclienteweb = io('ws://ip:puerto');
    socketclienteweb.on('evento', function (data) {
      //procesar los datos recibidos
    }
}
```

El socket descrito anteriormente se mantiene conectado siempre que se mantenga abierto el navegador Web, escuchando todos los mensajes en formato JSON que le envía el Servidor Node JS a través de eventos como se aprecia a continuación.

```
socketservidorweb.emit("evento",{dato1:'dato1', dato2:'dato2'});
```

Por medio de este evento el cliente Web recibe los datos que son procesados en función de la lógica del negocio para ser visualizados en el navegador Web.

Cuando se desencadena algún evento que debe ser enviado al Servidor Node JS para que este notifique instantáneamente a la Plataforma Móvil, se transmite por medio del protocolo HTTP hacia el Servidor Web por medio de los métodos POST, GET o AJAX. Este lo recibe, realiza las operaciones en el Gestor de Base de Datos en caso de ser necesario y luego se conecta y envía el mensaje al Servidor Node JS como se ilustra a continuación.

```
$dirip = "localhost";
$puerto = 5900;
if(!($sock1 = socket_create(AF_INET, SOCK_STREAM, 0))){
   $errorcode1 = socket_last_error();
   $errormsg1 = socket_strerror($errorcode1);
   die("Couldn't create socket: [$errorcode1] $errormsg1 \n");
}
if(!socket_bind($sock1, $dirip)){
   $errorcode1 = socket_last_error();
   $errormsg1 = socket_strerror($errorcode1);
   die("Could not bind socket : [$errorcode1] $errormsg1 \n");
}
socket_connect($sock1, $dirip , $puerto);
$mensaje="Mensaje de ejemplo,# \n";
socket_write($sock1, $mensaje);
```

De esta forma el mensaje es transmitido al Servidor Node JS desde el Servidor Web montado sobre Apache utilizando la biblioteca socket que provee el lenguaje PHP.

3.2. Implementación del Servidor Node JS

El Servidor Node JS está implementado sobre herramientas y tecnologías libres como se describen en la Tabla 2 .

Tabla 2
Herramientas y tecnologías del Servidor Node JS

Tecnologías	**Herramientas**
• NET (JavaScript) • PG (JavaScript) • SOCKET.IO (JavaScript)	• DOCKER • NODE JS

Como es posible apreciar el Servidor Node JS está embebido en una máquina virtual que provee la herramienta DOCKER con el objetivo de facilitar el despliegue y administrar los recursos con mayor eficiencia. Bajo este escenario se crean varios sockets que actúan como

servidor al que se conectan diferentes clientes provenientes de la Plataforma Móvil o de la Web como se refleja en los siguientes ejemplos.

```
'use strict';
//Creando una instancia TCP de un socket
var socketservidortcp = require('net');
//Creando una instancia de conexión a Postgre SQL
const pg = require('pg');
const connectionString = process.env.DATABASE_URL ||
'postgres://usuario:clave@ip:puerto/basededatos';
const db = new pg.Client(connectionString);
db.connect();
//Creando un socket servidor bajo conexión TCP
socketservidortcp.on('connection', function (sock) {
      process.setMaxListeners(0);
      sock.setEncoding('utf8');
        sock.on('data', function (text1) {
            sock.write('mensaje,# \n');
      });
});
```

```
Device connected
Plataforma Móvil Cliente conectada: 1
::ffff:
```

El socket socketservidortcp permite comunicarse con los clientes de la Plataforma Móvil que se conectan, enviando y recibiendo mensajes.

```
//Creando un socket servidor bajo conexión TCP con la librería Socket.io
var socketservidorWeb = require('socket.io')(puerto);
socketservidorWeb.on('connection', function (socket) {
   socketservidorWeb.emit("evento",{dato1:'dato1', dato2:'dato2'});
});
```

```
Client web disconnected
Cliente Web conectado:
::ffff:
```

El socket socketservidorWeb permite enviar y recibir mensajes de los clientes Web que se conectan desde la plataforma.

3.3. Implementación de la Plataforma Móvil

Para la validación de este componente de la arquitectura, se aplicó en el desarrollo de una Plataforma Móvil para sistema operativo Android. La misma inicia instanciando una clase personalizada (***ClientSocket***) que encapsula toda la funcionalidad necesaria para conexión, reconexión y autenticación automática, recepción y envío de tramas personalizadas, así como chequeo de errores. El constructor de esta clase es llamado en el momento de instanciación y son pasados parámetros tales como: ip o url remoto, puerto, handler para comunicar tramas recibidas entre otros parámetros propios de toda aplicación Android.

```
public class ClientSocket {
…
public ClientSocket(Context context,String url, String ip, String port,
boolean needReconect,Handler handelrMsg)
{
  …
}
…

}
```

La clase **ClientSocket** implementa un método privado de escucha continua luego de la conexión y comunica tramas recibidas al servicio por medio de mensajes a través de ***handlerMsg***, además implementa un método público para envío de tramas con verificación de recepción y retransmisión incluida.

Las clases básicas incluidas en bibliotecas para comunicación por Sockets implementan métodos de lectura que bloquean el hilo en que se ejecutan hasta que no se reciban los datos esperados. En este sentido es importante implementar toda la comunicación en hilos independientes y evitar incluirlos en el hilo de Interfaz de usuario (UI) o algún otro hilo con mayor prioridad que pueda ser bloquedo.

Siguiendo esta idea, la clase personalizada para comunicación basada en sockets (**ClientSocket**) define tres clases que implementan la clase abstracta ***Runnable*** y sobrescriben sus métodos **run()** con las funcionalidades necesarias de cada una. Esto garantiza que al instanciar la clase ***ClientSocket*** al inicio del servicio toda su funcionalidad de conexión, recepción y envío de paquetes se ejecute en hilos independientes evitando posibles bloqueos. El siguiente código resume la declaración de estas clases.

```
...
private class ConnectSocket implements Runnable {
     public ConnectSocket(){
         estado  = 0;
     }
     @Override
     public void run() {
         if(ip.equals(""))
         {
             if(!url.equals(""))
             {
                 ip = ipAddr(url);// metodo privado para resolver IP
             }
         }
         socket = new Socket();
         SocketAddress sockaddr = new InetSocketAddress(ip, port);
         try {
             socket.connect(sockaddr, 10000); //timeout de 10seg
         }  catch (IOException e) {
             e.printStackTrace();
         }catch (NullPointerException e){
         }
...
```

En este punto ya se sabe si se ha logrado conexión verificando el resultado del método **socket.isConnected()**. De ser verdadero entonces se instancia la clase **Socketlistener** la cual implementa la lectura continua y comunica por medio de mensajes las tramas recibidas.

```
private class Socketlistener implements Runnable
{
    String st = "";
    BufferedReader reader = null;
    InputStreamReader streamReader = null;
    InputStream inputStream = null;
    boolean error;
    public Socketlistener(){
        error = false;
    }
...
```

En el método **run()** de esta clase se obtiene el flujo de entrada del socket, se instancian las clases **InputStreamReader** y **BufferedReader** que permiten lecturas de cadenas de forma más elaborada y se entra a un lazo de escucha infinito que llama al método **BufferedReader.readLine()**. Del ciclo de lectura solo se sale por errores de lectura debido a problemas de conexión de cualquier tipo en cuyo caso se procede a reconectar instanciando nuevamente la clase **ConnectSocket().**

Adicionalmente la clase de comunicación **ClientSocket** implementa una clase privada para envío de paquetes también en su propio hilo independiente. Esta clase se utiliza instanciándola en un método público de la clase principal.

```
...
class Socketsender implements Runnable
{
    boolean error = false;
    String pk;
    OutputStream outputStream;
    public Socketsender(String pack)
    {
        pk = pack;
    }
...
```

En el método **run()** de esta clase se obtiene el flujo de salida del socket y se instancian las clases **BufferedWriter** y **PrintWriter** que permiten de forma más elaborada enviar cadenas. La cadena se envía llamando al método **PrintWriter.println(pk)**. También en este método se implementa el chequeo de recepción utilizando métodos privados también implementados en la clase **ClientSocket**.

El siguiente código es el método público que permite enviar una cadena.

```
...
public void sendPack(String pack)
{
    new Thread(new Socketsender(pack)).start();
}
...
```

---

4. Conclusiones

Del desarrollo del presente trabajo permitió arribar a las siguientes conclusiones:

Las arquitecturas de integración basadas en SOA y REST son cada vez más implementadas en los SI distribuidos. No obstante, en la actualidad no implementan bibliotecas o protocolos que permitan la comunicación instantánea entre plataformas móviles y web.

La tecnología socket es una de las más implementadas en la actualidad en soluciones de mensajería instantánea, las plataformas web y móviles cuentan con bibliotecas que prendan el soporte necesario para su desarrollo.

La arquitectura de integración propuesta brinda una alternativa eficiente de comunicación instantánea basada en socket que permite el envío y recepción de mensajes entre plataformas móviles y web.

Herramientas como DOCKER facilitan el despliegue de soluciones de este tipo y contribuyen con la optimización de los recursos.

El caso de estudio implementado permite evidenciar el funcionamiento de la arquitectura de integración propuesta y la comunicación entre los diferentes componentes.

Referencias bibliográficas

Corporation, S. A. (2007). *El Modelo de Madurez BPM /SOA.* USA.

Echeverría, R. R., Macías, F., Pavón, V. M., Conejero, J. M., & Figueroa, F. S. (2015). Legacy Web Aplication Modernization by Generating a REST Service Layer. *IEEE LATIN AMERICA TRANSACTIONS*, 1-5.

Freitas, V. d. (2015). Una propuesta de arquitectura para los Sistemas Informáticos de Gestión del Conocimiento en Instituciones de Educación Superior. *Espacios*, 1.

Josué Gustavo Rojas Sáenz, J. M. (2013). Diseño de una técnica presupuestal para las microempresas agroindustriales de alimentos de la ciudad de Sogamoso, Fundamentado en la caracterización de cada una de ellas. *uptc*, 13-17.

Kay, J. (1994). Fundamentos del éxito empresarial:. *Revista de Economía Aplicada*, 255.

Manoela POPESCU, I. S. (2015). Considerations Regarding the Psychosomatics Importance in Achieving Business. *proquest.com*, 2.

Merchán, V., & Hernández, J. (2016). Strategic Vision Excellence in the Use of Information Technology. *IEEE LATIN AMERICA TRANSACTIONS*, 1-3.

OASIS. (2006). *Web Services Security: Soap Message Security 1.1.* USA: OASIS.

Ordóñez, R. P., & Vélez, M. P. (2010). *DEFINICIÓN DE UN ESQUEMA DE SEGURIDAD EN SERVICIOS WEB PARA LA UNIVERSIDAD TÉCNICA PARTICULAR DE LOJA.* Loja: UNIVERSIDAD TÉCNICA PARTICULAR DE LOJA.

R, K. T. (1993). *Investigación de mercados.* Santafé de Bogotá: McGraw-Hill.

Rafi Ashrafi, S. K.-B.-G. (2014). Achieving Business Success Through Information and Communication. *Middle-East Journal of Scientific Research*, 138-146.

Remonato, R., & Balbinot, Z. (2011). Una brecha tecnológica en la adopción de nuevas tecnologías en el sistema de enseñanza de una institución de educación superior en un país en vías de desarrollo industrial - Estructura CAGE. *Espacios*, 2-3.

Rowlands, I., & Bawden, D. (1999). Digital Libraries: a conceptual framework. *Libri Journal*, 192-202.

Sánchez, E. S., Clemente, P. J., Prieto, Á. E., Conejero, J. M., & Echeverría, R. R. (2017). MigraSOA: Migration of Legacy Web Applications to Service Oriented Architectures (SOA). *IEEE LATIN AMERICA TRANSACTIONS*, 1-4.

Santana, N. A., Lins, F. A., & Sousa, E. (2016). Performance Evaluation of Mobile Applications in Mobile Cloud Environments. *IEEE LATIN AMERICA TRANSACTIONS*, 2-4.

Vera, P. M., & Rodríguez, R. A. (2016). Creating and Using Proximity Events on Mobile Websites. *IEEE LATIN AMERICA TRANSACTIONS*, 1-2.

W3C. (24 de 09 de 2007). *W3C*. Obtenido de http://www.w3c.es/Divulgacion/GuiasBreves/ServiciosWeb

1. Doctor en ciencias y Profesor titular de la Facultad de Sistemas Computacionales y Telecomunicaciones de la Universidad Tecnológica ECOTEC, Samborondón, Guayas, Ecuador. ogomez@ecotec.edu.ec

2. Máster en automática y Profesor titular de la Facultad de Sistemas Computacionales y Telecomunicaciones de la Universidad Tecnológica ECOTEC, Samborondón, Guayas, Ecuador. amachado@ecotec.edu.ec

3. Máster en nuevas tecnologías y Gerente general de la empresa Information Technology XOA S.A, Guayas, Guayaquil, Ecuador. alexis@xoait.com

4. (Ordóñez & Vélez, 2010). Refleja el modelo de madurez de SOA el Gobierno en ejecución, el Almacenamiento Operacional y la Gestión Operacional de la arquitectura.

Revista ESPACIOS. ISSN 0798 1015
Vol. 38 (Nº 59) Año 2017

[Index]