Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 11, No. 6, Mes Junio, 2018
ISSN: 2306-2495 | RNPS: 2343
Pág. 27-42
http://publicaciones.uci.cu

Tipo de artículo: Artículo original
Temática: Soluciones informáticas
Recibido: 20/04/18 | Aceptado: 05/06/18 | Publicado: 29/06/18

Solución para la interoperabilidad de Odoo con otros sistemas

Solution for Odoo interoperability with other systems

Claudia Bravo Batista[1], Janier Treto Portal [2], Nemury Silega Martínez[3]

[1] Universidad de las Ciencias Informáticas, cbravo@uci.cu
[2] Empresa de Tecnología de la Información para la Defensa, jtreto@xetid.cu
[3] Universidad de las Ciencias Informáticas, nsilega@uci.cu

[*] Autor para correspondencia: cbravo@uci.cu

Resumen

Los sistemas de planificación de recursos de la empresa (ERP por sus siglas en inglés) son sistemas de gestión de información que integran y automatizan las prácticas de negocios asociadas con los aspectos operativos y productivos de una empresa. En la Universidad de las Ciencias Informáticas se desarrolla una personalización de Odoo que es sistema ERP, con bases tecnológicas potentes y una estructura única, proyectado a la gestión de los procesos empresariales en Cuba. La proyección de Odoo sobre la base de intercambiar y utilizar información de otros sistemas ERP, brindar y consumir servicios de otras aplicaciones web, se encuentra bajo las condiciones de garantizar una mejora en los procesos y servicios que se ofrecen en la empresa. La presente investigación ofrece una solución informática, que permite que Odoo pueda interoperar consumiendo y compartiendo información, con otros sistemas desarrollados en diferentes lenguajes y tecnologías.

Palabras clave: flash; interoperabilidad; Odoo; servicios REST

Abstract

The systems Enterprise Resource Planning (ERP) are information management systems that integrate and automate the business practices associated with the operational and productive aspects of a business. At the University of Computer Science develops a customization of Odoo that is ERP system, with powerful technological bases and a unique structure, projected to the management of business processes in Cuba. Odoo's projection based on exchanging and using information from other ERP systems, providing and consuming services of other web applications, is under the conditions to guarantee an improvement in the processes and services offered in the company. The present investigation offers a computerized solution, that allows Odoo to interoperate by consuming and sharing information, with other systems developed in different languages and technologies.

Keywords: *flash; interoperability; Odoo; REST services*

Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 11, No. 6, Mes Junio, 2018
ISSN: 2306-2495 | RNPS: 2343
Pág. 27-42
http://publicaciones.uci.cu

Introducción

Los sistemas de planificación de recursos de la empresa (ERP por sus siglas en inglés) son sistemas de gestión de información que integran y automatizan las prácticas de negocios asociadas con los aspectos operativos y productivos de una empresa. El Sistema de Gestión de Entidades Cedrux del proyecto ERP-Cuba, desarrollado en la Universidad de las Ciencias Informáticas (UCI), con una importancia significativa para el país. Se caracteriza por la administración de los recursos naturales, humanos y financieros de la empresa, para un mejor control, distribución y planificación de los recursos. Con la misma línea de investigación se desarrolla el sistema Odoo, que es un sistema ERP, con bases tecnológicas potentes y una estructura única, proyectado a la gestión de los procesos empresariales en Cuba.

La proyección de Odoo sobre la base de intercambiar y utilizar información de otros sistemas ERP, brindar y consumir servicios de otras aplicaciones web, se encuentra bajo las condiciones de garantizar una mejora en los procesos y servicios que se ofrecen en la empresa.

Sobre la base del planteamiento anterior constituye un **Problema a resolver:** ¿Cómo garantizar que Odoo pueda intercambiar y utilizar información con otros sistemas ERP?

Se identifica como **Objeto de estudio:** Servicio Web basado en REST, enmarcado en el **Campo de acción:** Flask como entorno de desarrollo web para Python. Para darle respuesta al problema planteado se identifica el siguiente **Objetivo General:** Desarrollar una solución informática de interoperabilidad de Odoo con otros sistemas que permita garantizar el intercambio y la utilización de la información.

Se pretende obtener como **Posibles resultados:** Solución informática para garantizar la interoperabilidad de Odoo con otros sistemas.

Materiales y métodos o Metodología computacional

Método histórico- lógico: Se aplicó para recolectar información sobre los conceptos y elementos que caracterizan la interoperabilidad entre los sistemas, principalmente los sistemas ERP. Se analizaron los servicios REST y la implementación de los mismos, utilizando Flask como entorno de desarrollo web.

Método analítico sintético: Se aplicó para analizar y sintetizar la información extraída. Se evidencia en el núcleo de la investigación que es la interoperabilidad como garantía de que Odoo pueda consumir y brindar información con otros sistemas ERP.

Modelación: Se aplicó con el objetivo de diseñar un mecanismo de comunicación de Odoo con otros sistemas, desarrollados con diferentes tecnologías y lenguajes de programación. Se utilizó los servicios que se brindan en REST y los protocolos de comunicación, para garantizar que Odoo pueda consumir y brindar información con otros sistemas.

Métodos empíricos

Entrevista: Se aplicó para obtener la información de los servicios que se deben de consumir y brindar por Odoo, se realizó a los especialistas del proyecto Odoo en la universidad. Conocimientos que abarco principalmente elementos del negocio descrito y la necesidad de establecer una comunicación con sistemas ERP.

Definición de Interoperabilidad en ODOO

La investigación representa el diseño de una solución que permita la interoperabilidad de ODOO con otros sistemas implementados en diversos lenguajes de programación. Según la IEEE (Institute of Electrical and Electronics Engineers) la interoperabilidad es la capacidad de dos o más sistemas o componentes para intercambiar la información y utilizarla. El diseño contiene los elementos que deben garantizar que en el sistema Odoo se pueda intercambiar información de manera que ambos sistemas se actualicen de manera concurrente.

REST (Representational State Transfer) es un tipo de arquitectura de desarrollo web que se apoya totalmente en el estándar HTTP. REST permite crear servicios compatibles con cualquier dispositivo o cliente HTTP. Se caracteriza por contener una interfaz de comunicación que abstrae para el cliente el funcionamiento interno del servidor, ninguna información de contexto del cliente se almacena en el servidor, cada petición del cliente contiene toda la información necesaria para que el servidor sea capaz de preparar la respuesta. El cliente puede querer cachear las respuestas, por lo que estas deben ser cacheables. Es decir, debe incluirse en los headers de las respuestas la cabecera Last-Modified y soportar la recepción de la cabecera If-Unmodified-Since en las peticiones GET del cliente.

En el extremo del servidor, el estado y la funcionalidad de la aplicación se dividen en recursos. Un recurso es un elemento de interés, una identidad conceptual que se expone a los clientes. Cada recurso es de acceso único a través de una URI (Universal Resources Identifier-identificador de recurso universal). Todos los recursos comparten una interfaz uniforme para la transferencia de estados entre el cliente y servidor. Se usan métodos estándar HTTP como GET, PUT, POST y DELETE.

REST contiene una interfaz uniforme de identificación de recursos, manipulación de recursos mediante su representación, mensajes autodescriptivos, hipermedia como motor de estado.

El servidor debe seguir una arquitectura de capas, en la que cada capa conoce únicamente la capa inmediatamente inferior. De esta forma, la capa superior (interfaz de interacción) abstrae el comportamiento interno. El cliente no

debe conocer nada sobre la implementación del servidor. Se provee al cliente con código capaz de manejar la respuesta del servidor. El mejor ejemplo de código en demanda son los *scripts* de JavaScript que contienen las páginas web.

Diseñar un servicio Web basado en REST

- Identificar todas las entidades conceptuales que se desean exponer como servicio.
- Crear una URL para cada recurso. Los recursos deberían ser nombres no verbos (acciones).
- Categorizar los recursos si los clientes pueden obtener una representación del recurso o si pueden modificarlo. Para el primero, debemos hacer los recursos accesibles utilizando un HTTP GET. Para el último, se debe hacer los recursos accesibles mediante HTTP POST, PUT y/o DELETE.
- Todos los recursos accesibles mediante GET no deben tener efectos secundarios. Es decir, los recursos deben devolver la representación del recurso. Por tanto, invocar al recurso no debe ser resultado de modificarlo.
- Ninguna representación debe estar aislada. Es decir, es recomendable poner hipervínculos dentro de la representación de un recurso para permitir a los clientes obtener más información.
- Especificar el formato de los datos de respuesta mediante un esquema (DTD, W3C Schema,..). Para los servicios que requieren un POST o un PUT se recomienda proporcionar un esquema para especificar el formato de la respuesta.
- Describir como el servicio va a ser invocado, mediante un documento WSDL/WADL o simplemente HTML.

Se propone la utilización del framework de Python flask el cual se integra en un módulo de Odoo. Flask permite la implementación de una API REST la cual interactúa con la base de datos de Odoo. Para ello se identifican las tablas con las que se va a interactuar y se registran como recursos en la API creándose los servicios web que permiten la modificación de los datos. De esta forma, al iniciarse el servidor de Odoo se inicia automáticamente la aplicación flask y se publican los servicios definidos proporcionando elementos de seguridad para el consumo de los mismos. A partir de este momento cualquier aplicación podrá interoperar con Odoo con independencia del lenguaje en el que este desarrollada. Para ello deberá implementar un consumidor de servicios que utilizando los métodos GET, POST, PUT y DELETE definidos por el protocolo HTPP realice las peticiones adecuadas que permitan listar, insertar, modificar y eliminar información en el servidor de odoo. Como resultado de estas peticiones recibirá una respuesta de la API REST en formato json con el resultado de la operación solicitada. Como resultado de este proceso ambos sistemas podrán interoperar compartiendo información entre ellos.

Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 11, No. 6, Mes Junio, 2018
ISSN: 2306-2495 | RNPS: 2343
Pág. 27-42
http://publicaciones.uci.cu

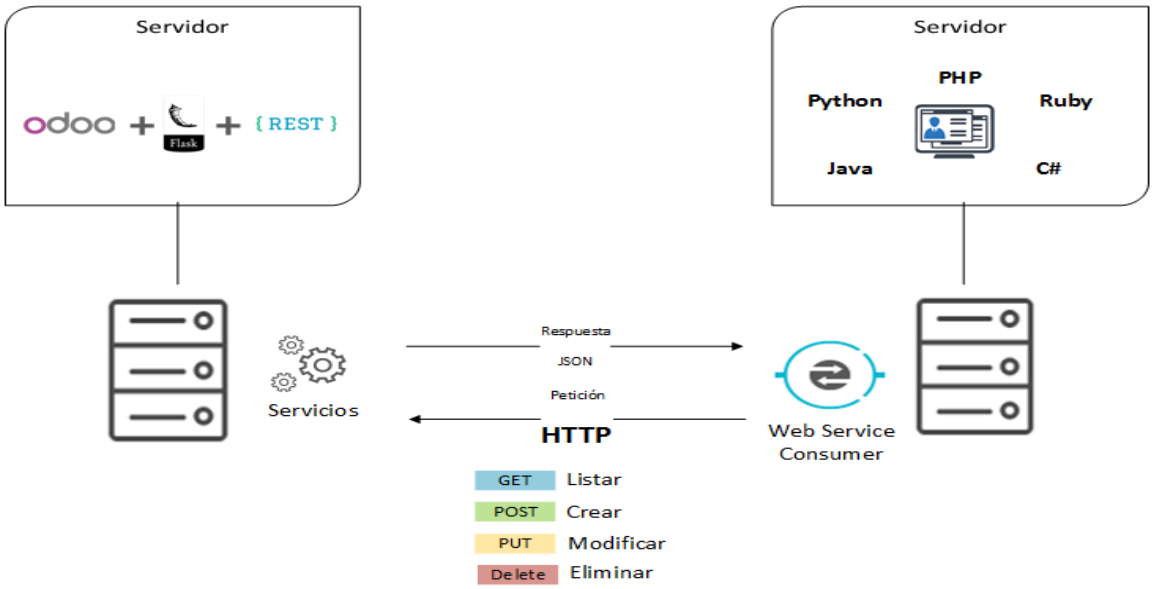

Figura1. Arquitectura del sistema

Resultados y discusión

Interoperabilidad de ODO con los lenguajes de programación

Flask es un entorno de desarrollo web para Python, que permite implementar servicios REST. El núcleo dispone de un número reducido de características, las mínimas para implementar las abstracciones del protocolo HTTP y su manipulación. Es fácilmente extensible y puede añadirse paquetes según las necesidades. Tiene integración con SQLAlchemy, un popular ORM de Python, se puede añadir los procesos de registro y autenticación con Flask-Security.

Para el desarrollo con Flask se debe tener en cuenta los siguientes elementos:

1. **Instanciar una aplicación Flask**

```
from flask import Flask
from common.odoo_auth import OdooServer
from common.tools import auth
odoo_server = OdooServer(
    application.config.get('ODOO_SERVER'),
    application.config.get('ODOO_PORT'),
```

Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 11, No. 6, Mes Junio, 2018
ISSN: 2306-2495 | RNPS: 2343
Pág. 27-42
http://publicaciones.uci.cu

```
        application.config.get('ODOO_DB')
    )
@auth.verify_password
    def verify_pw(username, password):
        user = odoo_server.login(username, password)
        if user:
            g.user = user
            return True
        return False
    component = Flask (_name_)
```

2. **Definir un servicio, ejemplo**

```
@componente.route('/')
def index(): return 'Servicio'
```

3. Ejecutar la aplicación

```
If _name_ = = '_main_':
app.run (debug = True, port='5900', host = '127.0.0.0') //permite realizar cambios en el código sin relanzar la aplicación.
```

4. Se le da permisos de ejecución

```
    chmod 777 -R /directorio de la aplicación/app.py.
    5. Se ejecuta la aplicación
    ./python app.py
```

REST está comprendida por una seria de limitaciones y principios arquitectónicos, si una aplicación o diseño cumple con esas limitaciones y principios, se considera RESTful.

Para que un servicio sea considerado RESTful debe cumplir los siguientes requisitos:

1. Uso correcto de URIs
2. Uso correcto de HTTP
3. Implementación de Hypermedia

Uso correcto de URIs

La URL (Uniform Resource Locator) son un tipo de URI (Uniform Resourse Identifier) que permiten identificar de forma única un recurso. Tienen la forma:

<protocolo>://<hostname>[:puerto]/<ruta_del_recurso>

Existe una serie de reglas a la hora de nombrar recursos:

. El nombre de un recurso no debe implicar una acción. Las acciones ya se especifican mediante los verbos de HTTP.

. Deben ser únicas. No se debe tener más de una URI para identificar un mismo recurso.

. Deben ser independientes del formato. /prueba/2.pdf no seria correcto.

. Deben mantener una jerarquía lógica, de lo más amplio a lo más concreto. Incorrecto: /algoritmos/111/prueba/142. Correcto: /prueba/142/algoritmos/111

. Los filtrados de información de un recurso no se hacen en la URI, sino utilizando parámetros HTTP:

Incorrecto: /facturas/orden/desc/fecha-desde/2007/pagina/2

Correcto: /facturas?fecha-desde=2007&orden=DESC&pagina=2

Uso correcto de HTTP

Los RFC (Request For Comments) contienen la especificación de HTTP:

RFC 7230 - Message syntax and routing (https://tools.ietf.org/html/rfc7230)

RFC 7231 - Semantics and Content (https://tools.ietf.org/html/rfc7231)

RFC 7232 - Conditional requests (https://tools.ietf.org/html/rfc7232)

RFC 7233 - Range requests (https://tools.ietf.org/html/rfc7233)

RFC 7234 - Caching (https://tools.ietf.org/html/rfc7234)

RFC 7235 - Authentication (https://tools.ietf.org/html/rfc7235)

RFC 7236 - Authentication Scheme Registrations(https://tools.ietf.org/html/rfc7236)

RFC 7237 - Method Registrations (https://tools.ietf.org/html/rfc7237)

HTTP propone 5 métodos:

GET: Para leer un recurso

HEAD: Para leer metainformación de un recurso

POST: Para crear un recurso

PUT: Para editar un recurso

DELETE: Para eliminar un recurso

PATCH: Para editar partes de un recurso

Códigos de estado

Para que un servicio pueda ser considerado RESTful es vital que haga un uso correcto de los códigos de estado que ofrece HTTP. Existe un código preestablecido para cada posible situación, los mismos se agrupan de la siguiente forma (solo se enumeran algunos):

. **1XX: Respuestas informativas**. Se utilizan cuando un cliente quiere saber si su petición será aceptada o rechazada por los headers antes de enviar al servidor todo el cuerpo de la misma (si es de gran tamaño).

.100: La petición será aceptada

.101: Conmutando protocolos

.102: Procesando

. 2XX: Peticiones correctas. La petición fue recibida correctamente, entendida y aceptada.

.200: OK

.201: Elemento creado

.202: Petición aceptada, pero aún se está procesando. Podría no completarse si se produce algún error.

.203: Información no autoritativa. La respuesta ha recibido alguna transformación por parte de un *proxy*.

. 204: Respuesta vacía

.205: El servidor solicita al cliente que recargue la página.

.206: Contenido parcial. Utilizado para dividir una descarga o interrumpirlas y reanudarlas.

.207: Estado múltiple. El cuerpo es un XML con subcódigos de estado, dependiendo de las sub-peticiones hechas.

. 3XX: **Redirecciones**. El cliente tiene que realizar una acción adicional. No está permitido hacerlo más de 5 veces seguidas: bucle infinito de redirección.

.300: Se ofrecen distintas posibilidades al cliente. Ejemplo: distintos formatos de vídeo.

.301: Movido permanentemente.

.302: Movido temporalmente. Es el ejemplo más popular de redirección, así como un ejemplo de contradicción del estándar.

.303: See other. La respuesta a la petición se encuentra haciendo GET a otra URI. Muchas veces se utiliza 302 como si fuera 303.

.304: Indica que el recurso no ha sido modificado desde la última vez que se pidió. Ahorra ancho de banda.

.307: Redirección temporal. Como 303 pero debe soportar cualquier método (no solo GET).

4XX: Errores del cliente. La solicitud es incorrecta o no puede procesarse.

400: Solicitud incorrecta.

401: No autorizado para acceder al recurso en este momento (o la autorización ha fallado).

403: No autorizado para acceder al recurso. La autorización se ha realizado correctamente pero el acceso es denegado.

404: Recurso no encontrado.

405: Método (GET, POST...) no permitido para la URI utilizada.

406: Los formatos aceptados por el cliente (header Accept) no están disponibles.

410: Indica que el recurso ya no está disponible ni lo estará (pero lo ha estado en el pasado). En ocasiones se utiliza erróneamente 404 en lugar de 410.

423: Recurso bloqueado.

5XX: Errores del servidor. El servidor falló al completar una solicitud aparentemente válida.

500: Error interno, genérico, ajeno a la naturaleza del propio servidor web.

501: Funcionalidad no implementada.

503: Servicio no disponible en este momento.

505: Versión de HTTP no soportada.

509: Límite de ancho de banda excedido (no es un código oficial, pero es ampliamente utilizado).

Formatos de contenido

Para indicar los tipos de formato aceptados, el cliente debe especificar el header Accept. La Interfaz de programación de aplicaciones, abreviada como API del inglés: Application Programming Interface, es un conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción, debe devolver el resultado en el primer formato disponible, o un error 406 si ninguno de los formatos aceptados está disponible.

1. GET /people/1234
2. **Accept**: application/epub+zip, application/pdf, application/j son

En la respuesta se debe indicar el header Content-Type para indicar el formato del cuerpo.

1. **Status Code** 200.
2. **Content-Type**: application/pdf

Implementación Hypermedia

Con la Hypermedia se conecta los recursos con otros mediante enlaces. Se automatiza la utilización de APIs sin que haya interacción humana. También para no tener que conocer la ubicación exacta de todos los recursos. Por ejemplo:

```
<pedido>
<id>1234</id>
<status>Pending</status>
<links>
<link rel="factura">
http://example.com/api/pedido/1234/factura
</link>
</links>
</pedido>
```

El cliente debe entender la información, de forma que pueda separar el recurso en sí con la información para enlazarla con otros recursos. Para ello se utilizan los headers Accept y Content Type. Es una forma de que el cliente y el servicio sepan que están utilizando el mismo idioma. Este idioma debe especificarse en algún documento que el cliente de la API debe conocer.

Evaluación de la Interoperabilidad de ODO con el lenguaje de programación PHP

Para realizar pruebas de funcionamiento, analizar los headers y cuerpos de las respuestas de los servicios, se utiliza CURL.

CURL es una biblioteca que permite conectarse y comunicarse con diferentes tipos de servidores y diferentes tipos de protocolos. A continuación, se muestra una implementación del consumidor de servicios (Interfaz) utilizando curl en php para consumir servicios REST.

```
<?php
class webServiceConsumer
{
   public $server ;
   public $port ;
   public $user;
   public $password;
   /**
    * Constructor de la clase
    * @param string $server - Dirección del servidor
```

```
 * @param integer $port - Puerto del servidor
 * @param string $user - Usuario
 * @param string $password - Contraseña
 */
public function __construct($server, $port, $user, $password){
  $this->server = $server;
  $this->port = $port;
  $this->user = $user;
  $this->password = $password;
  }
/**
 * Lista los recursos existentes en base de datos
 * @param string $resource - Nombre de recurso
 * @return array $response - retorna un listado de recursos
 */
public function listResources($resource)
{
   //Se inicia la conexión a la url del api rest
   $ch = curl_init($this->server.":".$this->port.DIRECTORY_SEPARATOR.$resource);
   //Se establecen los parámetros de conexión a odoo
   curl_setopt($ch, CURLOPT_USERPWD, $this->user.":".$this->password);
   //a true, obtendremos una respuesta de la url, en otro caso,
   //true si es correcto, false si no lo es
   curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
   // Se establece la petición  http a utilizar
   curl_setopt ($ch, CURLOPT_CUSTOMREQUEST, "GET");
   //obtener la respuesta
   $response = curl_exec($ch);
   // Se cierra el recurso CURL y se liberan los recursos del sistema
   curl_close($ch);
```

```
        if(!$response) {
            return false;
        }else{
            return $response;
        }
    }
    /**
     * Adiciona un nuevo recurso
     * @param json $data - json con los datos
     * @param string $resource - Nombre de recurso
     * @return int $response - retorna el id del nuevo recurso
     */
    public function addResource($data,$resource)
        {
         //Se inicia la conexión a la url del api rest
        $ch = curl_init($this->server.":".$this->port.DIRECTORY_SEPARATOR.$resource);
        //Se establecen los parámetros de conexión a odoo
        curl_setopt($ch, CURLOPT_USERPWD, $this->user.":".$this->password);
                //a true, obtener una respuesta de la url, en otro caso,
                //true si es correcto, false si no lo es
                curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
        //establecer la petición http que se quiere utilizar
                curl_setopt($ch, CURLOPT_CUSTOMREQUEST, "POST");
                //enviar el json data
        curl_setopt($ch, CURLOPT_POSTFIELDS, $data);
        //Establecer las cabeceras http
        curl_setopt($ch, CURLOPT_HTTPHEADER, array("Content-Type: application/json", "Content length: ".strlen($data)));
        //obtener la respuesta
                $response = curl_exec($ch);
```

```
                // Se cierra el recurso CURL y se liberan los recursos del sistema
                curl_close($ch);
                if(!$response) {
                   return false;
                }else{
                        return $response;
                }
    }
   /**
    * Modifica un recurso
    * @param json $data - json con los datos a modificar
    * @param string $resource - Nombre de recurso
    * @param int $id - identificador del recurso a modificar
    * @return boolean $response - retorna TRUE si fue modificado
    */
public function modifyResource($data,$resource,$id)
    {
      //Se inicia la conexión a la url del api rest
$ch=curl_init($this>server.":".$this>port.DIRECTORY_SEPARATOR.$resource.DIRECTORY_SEPARATOR.
$id);
      //Se establecen los parámetros de conexión a odoo
      curl_setopt($ch, CURLOPT_USERPWD, $this->user.":".$this->password);
                //a true, se obtiene una respuesta de la url, en otro caso,
                //true si es correcto, false si no lo es
                curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
      //Se establece la petición http que se quiere utilizar
                curl_setopt($ch, CURLOPT_CUSTOMREQUEST, "PUT");
      //se envía el json data
      curl_setopt($ch, CURLOPT_POSTFIELDS, $data);
      //Se establece las cabeceras http
```

```
        curl_setopt($ch, CURLOPT_HTTPHEADER, array("Content-Type: application/json", "Content-length: ".strlen($data)));
                //obtenemos la respuesta
                $response = curl_exec($ch);
                // Se cierra el recurso CURL y se liberan los recursos del sistema
                curl_close($ch);
                if(!$response) {
                   return false;
                }else{
                        return $response;
                }
     }
    /**
     * Elimina un recurso
     * @param string $resource - Nombre de recurso
     * @param int $id - identificador del recurso a eliminar
     * @return boolean $response - retorna TRUE si fue eliminado
     */
      public function deleteResource($resource,$id)
      {
        //Se inicia la conexión a la url del api rest
$ch=curl_init($this>server.":".$this>port.DIRECTORY_SEPARATOR.$resource.DIRECTORY_SEPARATOR.$id);
        //Se establecen los parámetros de conexión a odoo
        curl_setopt($ch, CURLOPT_USERPWD, $this->user.":".$this->password);
        //a true, se obtiene una respuesta de la url, en otro caso,
                //true si es correcto, false si no lo es
                curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
        // Se establece la petición http que se quiere utilizar
                curl_setopt($ch, CURLOPT_CUSTOMREQUEST, "DELETE");
```

```
                //se obtiene la respuesta
                $response = curl_exec($ch);
                // Se cierra el recurso CURL y se liberan los recursos del sistema
                curl_close($ch);
                if(!$response) {
                   return false;
                }else{
                        return $response;
                }
        }
}
/******************Ejemplos de uso de la clase*************************/
//Crear una instancia  de la clase de conexión a odoo-rest
$conn = new OdooConnection("localhost",5000,'admin','odoo');
/*******Listar los usuarios existentes en la base de datos de odoo*******/
$result = $conn->listResources('res_users?fields=all');
echo '<pre>';print_r($result);
/***************Adicionar un nuevo usuario de odoo************************/
//$data = '{"name":"nuevousuario","login":"nuevologin"}';
//$result = $conn->addResource($data,'res_users');
//print_r($result);
/*********************Modificar un usuario*********************************/
//$data = '{"name":"pepe"}';
//$result = $conn->modifyResource($data,'res_users',5);
//print_r($result);
/************************Eliminar un usuario*********************************/
//$result = $conn->deleteResource('res_users',5);
//print_r($result);
?>
```

Conclusiones

La solución de interoperabilidad de Odoo con otros sistemas, principalmente los sistemas ERP garantiza el intercambio y la utilización de la información.

El concepto de interoperabilidad no sólo se centra en el intercambio y la utilización de la información, sino que abarca una arquitectura de software donde no existan sistemas independientes, desarrollado con tecnologías diferentes, sino un solo proyecto integrado.

El diseño de un servicio Web basado en REST, proporcionó una visión genérica del proceso de comunicación de Odoo con otros sistemas, donde ambos puedan interoperar compartiendo y consumiendo información.

Referencias

1. ¿Cómo los servicios web aumentan la interoperabilidad? [En línea] 03 de 07 de 2016. http://www.alegsa.com.ar/Servicios/politicas.php.
2. Martínez, Nemury Silega. Guía metodológica para gestionar la integración de componentes en los proyectos del sistema CEDRUX. La Habana : s.n., 2010.
3. Seguridad e interoperabilidad. [En línea] 21 de 03 de 2017. https://programacion1class.wordpress.com/unidad-5-programacion-del-lado-del-servidor/.
4. Interoperabilidad entre sistemas de información. Rodríguez, Margarita Soto. ISSN 2007 - 7467, s.l. : Revista Iberoamericana para la Investigación y el Desarrollo Educativo, 2014.
5. Rodriguez, Alex. RESTful Web services: The basics. febrero : Copyright IBM Corporation 2008, 2015, 2015.
6. Seguridad en la APIs REST.
7. ELP-DSIC-UPV, Rafael Navarro Marset. Modelado, Diseño e Implementación de Servicios Web . 2006.
8. Ruby, Leonard Richardson e Sam. Web Services for the Real World. 2007. 978-0-596-52926-0.
9. Todd Fredrich. RESTful Service Best Practices. 2012.
10. MarkLogic. REST Application Developer's Guide. s.l. : Copyright © 2016 MarkLogic Corporation. All rights reserved, 2015. Patent No. 7,127,469.
11. Santamaría, Rodrigo. REST avanzado.
12. Web API Design. [En línea] Brian Mulloy, Apigee, 2011. http://apigee.com.