

Documentación del Servicio de Búsqueda de Infomed



Autor: Lic. Yeleiny Bonilla Sierra
(Especialista principal)

Octubre - 2009

Tabla de Contenidos:

I. Introducción.....	3
1.1 Descripción del Servicio. Metabuscador.	3
1.2 Objetivos fundamentales del Servicio	4
1.3 Facilidades que debe brindar el Servicio	4
II. Diseño general del Servicio	6
2.1 Descripción de los componentes.....	6
2.1.1 Buscadores (Search Engines)	6
2.1.2 Fuentes de Información (Info Sources)	7
2.1.3 Adaptadores (Adaptors).....	8
2.2 Diseño de la Base de Datos.....	9
III. Implementación del Servicio	10
3.1 Implementación de los componentes fundamentales.....	10
3.1.1 Implementación de los Buscadores (Search Engines).....	10
3.1.2 Implementación de las Fuentes de Información (Info Sources).....	11
3.1.3 Implementación de los Adaptadores (Adaptors)	12
3.2 Implementación de la Función de búsqueda (search)	13
3.2.1 Implementación de un cliente XML-RPC. Búsqueda en el Lis.	14
3.2.2 Implementación de un cliente SOAP. Búsqueda en Base de Datos bibliográficas.	16
3.3 Implementación de la Función para adaptar formatos (adaptTo)	18
3.4 WebService del Servicio de Búsqueda	19
3.4.1 Generación del archivo descriptivo del WebService (WSDL).....	19
3.4.2 Implementación del Servidor (Server)	20
IV. Administración del Servicio de Búsqueda.....	21
4.1 Administración de Buscadores (Search Engines)	21
4.1.1 Adición de un nuevo Buscador (Add Engine).....	21
4.1.2 Confección del archivo de configuración (config.xml).....	22
4.1.3 Modificación o eliminación de Buscadores.....	23
4.2 Administración de Fuentes de Información (Info Sources).....	23
4.2.1 Adición de una nueva Fuente de Información (Add InfoSource)	23
4.2.2 Modificación o eliminación de Fuentes de Información	24
4.3 Administración de Adaptadores (Adaptors)	24
4.3.1 Adición de un nuevo Adaptador (Add Adaptor)	24
4.3.2 Modificación o eliminación de Adaptadores.....	25
V. Librerías útiles implementadas	26
5.1 Librería para parsear XML	26
5.1.1 Leyendo XML con DOM	27
5.1.2 Generando XML con DOM.....	28
VI. Implementación de OpenSearch	30
6.1 Características de OpenSearch. Implementación.....	30
6.2 OpenSearch en el Servicio de Búsqueda	32
VII. Estado actual del Servicio de Búsqueda	32
7.1 Interfaces del Servicio de búsqueda.....	32
7.2 Componentes ya insertados en el Servicio de búsqueda.....	33
7.3 Código fuente.....	34

I. Introducción

El servicio de Búsqueda de Infomed se concibe como un servicio independiente que mediante una interfaz de *WebServices* brinda todas las funcionalidades necesarias para realizar búsquedas sobre diversos Recursos de Información.

Actualmente servicio se está utilizando en el portal de INFOMED (el cual se aprecia en la figura circulado en rojo), y además puede ser usado desde otras aplicaciones, dígase POSH, CMS DRUPAL entre otras.



1.1 Descripción del Servicio. Metabuscador.

La concepción del Servicio es muy simple, en el sentido de que para efectuar una búsqueda el usuario solo deberá especificar una consulta y escoger la fuente de información en que desea buscar.

Para la definición del servicio se ha usado la idea de los **Metabuscadores**, concepto grandemente extendido en internet.

De Wikipedia, la enciclopedia libre cito.

*El metabuscador es un sistema que localiza información en los motores de búsqueda más usados y carece de base de datos propia y, en su lugar, usa las de otros buscadores y muestra una combinación de las mejores páginas que ha devuelto cada buscador. Un buscador normal recopila la información de las páginas mediante su indexación, como Google o bien mantiene un amplio directorio temático, como Yahoo. **La definición simplista sería que un metabuscador es un buscador de buscadores.***

Ventajas:

La ventaja principal de los metabuscadores es:

- Amplían de forma notoria el ámbito de las búsquedas que realizamos, proporcionando mayor cantidad de resultados. La forma de combinar los resultados depende del metabuscador empleado.

Puesto que muchos multibuscadores muestran en los resultados la posición de la web en los buscadores nos permite evaluar la relevancia de cada web mostrada.

Hay que tener en cuenta que cada buscador utiliza su propia estrategia a la hora recoger información de una página y ordenar los resultados de las búsquedas, por lo que las páginas de mayor relevancia en un buscador no tienen por qué coincidir con las del resto, aportando puntos de vista distintos.

Desventajas:

Una de las desventajas importantes es:

- Mientras que cada buscador dispone de su propia sintaxis de búsqueda los metabuscadores no distinguen entre las diferentes sintaxis. Por lo tanto, al buscar información muy específica es mejor emplear buscadores de los que conozcamos la sintaxis.

Es de notar que no resultan muy claros los criterios empleados por los diversos multibuscadores para la ordenación de sus resultados.

Al buscar en varias fuentes, la obtención de resultados suele ser más lenta que en un buscador normal. Muchos de los multibuscadores permiten establecer un tiempo máximo para realizar la búsqueda.

Según las necesidades de Infomed, y teniendo en cuenta el gran número de fuentes de información de las cuales se nutre nuestra red, muchas de las cuales ya tenían sus interfaces de búsqueda al estilo webservice; es por ello, que, definir el Servicio de Búsqueda como un Metabuscador, resultó ser una propuesta mayoritariamente aceptada.

1.2 Objetivos fundamentales del Servicio

Es válido destacar que el Servicio de Búsqueda surge por la necesidad de centralizar la búsqueda en las diversas fuentes de información de las que se nutre la red, las cuales se encontraban dispersas. Por tanto los principales objetivos para la implementación del mismo serán:

- Optimizar el proceso de búsqueda en fuentes de Información de INFOMED y BIREME implementando un servicio independiente para la realización del mismo.
- Acceder al Servicio de Búsqueda mediante un Sitio WEB (propio Portal de Infomed, por ejemplo), con interfaz amigable, que permita realizar búsquedas directamente en él, para cada una de las fuentes de información que tenga registradas.
- Permitir realizar búsquedas simples y avanzadas sobre los recursos de Información registrados.

1.3 Facilidades que debe brindar el Servicio

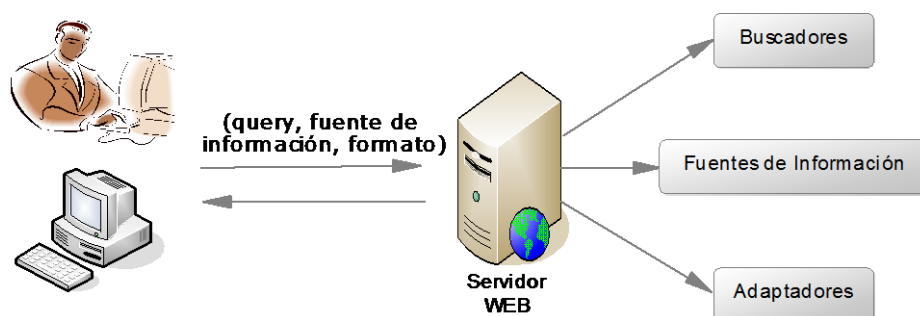
Siendo un Servicio concebido *por* y *para* el usuario, el mismo debe ser sencillo y fácil de usar. Brindando un conjunto de facilidades que satisfagan las necesidades de todos los que hagan uso del mismo, desde los usuarios más exigentes hasta los más inexpertos.

Es por ello que se definieron las siguientes premisas, que el mismo debía cumplir:

- Facilitar la búsqueda organizada sobre los diversos Recursos de Información (Ejemplo: LIS, Google, LisBireme).
- Posibilitar el registro de nuevas Fuentes y Recursos de Información donde buscar.
- Permitir realizar búsquedas sencillas (para todas las FI) y búsquedas avanzadas (sobre una FI).
- Poder ser usado desde cualquier otra aplicación de INFOMED.
- Facilitar la realización de análisis estadísticos sobre el uso del servicio.
- Servir de base para la implementación de otros servicios relacionados, tales como:
 - ◆ Realización de búsquedas automáticas.
 - ◆ Implementación de un Sistema de Recomendación.

II. Diseño general del Servicio

El Flujo General del Proceso de Búsqueda puede ser representado mediante un diagrama, en el mismo quedan identificadas cada una de las partes que interactúan en el proceso:



Como se puede observar en la figura, el Servicio de Búsqueda se tendrá corriendo en un Servidor Web. A este servicio podrán acceder los usuarios directamente, o podrá ser accedido desde otras aplicaciones. Para hacer uso del Servicio se especifica la consulta con el contenido que se desea buscar y se escoge la Fuente de Información donde se desea buscar, así como el formato en que se deseen devolver los datos.

Se definieron tres componentes fundamentales para la implementación del Servicio: *Buscadores*, *Fuentes de Información* y *Adaptadores*. Cada uno de los mismos se describe a continuación.

2.1 Descripción de los componentes

2.1.1 *Buscadores (Search Engines)*

Los buscadores son objetos que representan el recurso de información sobre el cual se realizarán las búsquedas (ejemplo: Google, Liscuba, LisBireme). Estos recursos tienen sus propios mecanismos de búsqueda y funciones que pueden ser invocadas generalmente vía webservice.

Los atributos definidos para estos son:

- *Nombre (name)*: Nombre que identifica al Buscador (Ej. liscuba). Por convenio debe ponerse con minúsculas este nombre.
- *Descripción (description)*: Descripción del Buscador. Debe ser un texto breve pero explicativo (Ej. Buscador para al Localizador de Información en Salud).
- *Implementación (implementation)*: Nombre del fichero php donde se encuentra la implementación del método de búsqueda para dicho buscador (*cliente webservice*). Por convenio estos ficheros deben nombrarse con el mismo nombre del buscador y a continuación el sufijo "_search_engine" (Ej. liscuba_search_engine.php)

- *Nombre de la Clase (class name)*: Nombre de la clase php implementada para dicho buscador. Dicha clase se encuentra implementada dentro del fichero de implementación. Por convenio la clase debe llamarse con el mismo nombre del buscador en mayúsculas, y a continuación el sufijo "SearchEngine" (Ej. LisCubaSearchEngine)
- *Parámetros (params)*: Los parámetros del buscador representan el conjunto de valores que permiten refinar las búsquedas en los mismos. Para la representación de estos se definió un XML con la siguiente estructura:

XML genérico:

```
<?xml version="1.0"?>
<parameters>           // Etiqueta madre parámetros
  <param>                //representa un solo parámetro
    <name>X</name>       //nombre del parámetro
    <type>list</type>    //si el param es de tipo list, entonces tiene options
    <options>            // opciones del param de tipo list
      <label value="0"> x1 </label> //tiene por casa label un value
      <label value="1"> x2</label>
      .....
      <label value="n"> xn</label>
    </options>
  </param>

  <param>
    <name>Y</name> //param de tipo text, este solo tiene name, el valor se
    <type>text</type> // introduce en la interfaz de administración
  </param>
</parameters>
```

2.1.2 Fuentes de Información (Info Sources)

Representan la fuente de información a partir de la cual se obtendrán el conjunto de resultados deseados para un determinado tema. Las fuentes de información son el recurso con el cual los usuarios realmente interactúan al solicitar sus búsquedas.

Cada fuente de información que se define debe ser asociada con un buscador. Estas no constituyen más que filtros para los buscadores, por lo que en la mayoría de los casos restringen algunos de los parámetros de estos.

Por ejemplo:

- *Sitios de Cuba*: se asocia con el buscador Google y restringe el parámetro url al dominio .cu.
- *Sitios de Salud*: se asocia con el buscador Google y restringe el parámetro url al dominio .sld.cu.

Los atributos definidos para estas son:

- *Nombre (name)*: Nombre que identifica a la Fuentes de Información. Por convenio debe ponerse con mayúsculas.
- *Descripción (description)*: Descripción de la Fuente de Información. Debe ser un texto breve y explicativo.
- *Buscador (search engine)*: Buscador asociado a la Fuente de Información.
- *Parámetros del Buscador (search engine params)*: Para los parámetros del buscador asociado a una fuente de información se definió un XML diferente al anterior, en este XML se definen, además de los nombres de los parámetros, sus valores.

Sitios de Salud

```
<?xml version="1.0"?>
<parameters>
//parámetro y valor
<param name="Site">sld.cu</param>
</parameters>
```

Sitios de Cuba

```
<?xml version="1.0"?>
<parameters>
//parámetro y valor
<param name="Site">.cu</param>
</parameters>
```

2.1.3 Adaptadores (Adaptors)

Los adaptadores son un objeto definido para adaptar los resultados de la búsqueda en una fuente de información del formato obtenido a otro formato especificado. Un adaptador se le asocia a uno o a varios buscadores, aunque pueden existir adaptadores que sean específicos para un único buscador.

Los atributos definidos para estos son:

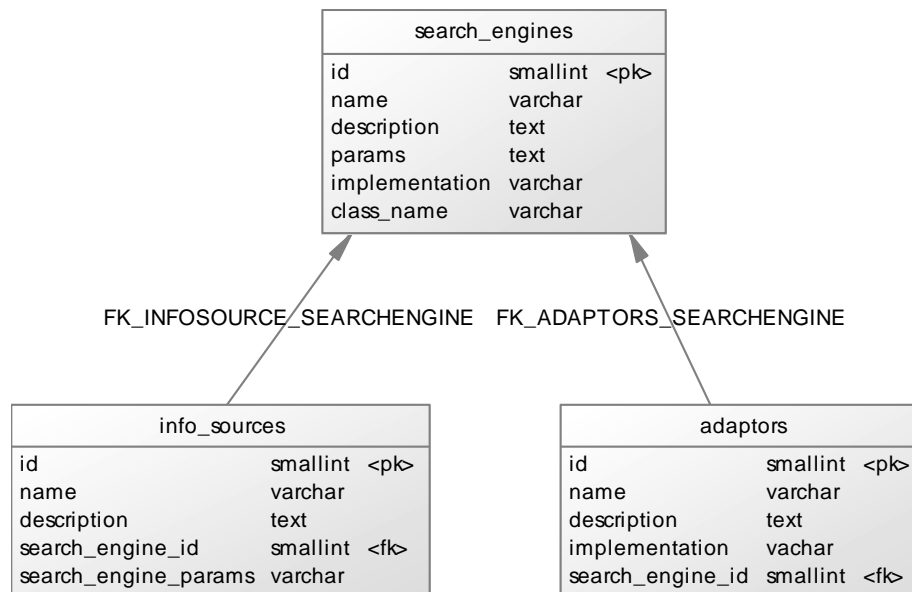
- *Nombre (name)*: Nombre que identifica al Adaptador (Ej. LisRSS, AIDiaRSS). Por convenio el nombre debe ser el del buscador con el cual se asocia seguido por el formato al cual adapta.
- *Descripción (description)*: Descripción del Adaptador. Debe ser un texto breve pero explicativo (Ej. Adaptador a RSS para LisCuba).
- *Implementación (file)*: Nombre del fichero php donde se encuentra la implementación del método para adaptar. Por convenio estos ficheros deben tener el sufijo "_adaptor" (Ej. general_adaptor.php).

- *Nombre de la Clase (class name)*: Nombre de la clase php implementada para dicho adaptador. Dicha clase se encuentra implementada dentro del fichero de implementación. Por convenio la clase debe llamarse con el mismo nombre del adaptador en mayúsculas, y a continuación el sufijo "Adaptor" (Ej. GeneralAdaptor).
- *Buscador (search engine)*: Buscador asociado al adaptador.

2.2 Diseño de la Base de Datos

A pesar de que por el propio concepto de Metabuscadore, el servicio de búsqueda no tendrá una base de datos de indexado ni de almacenamiento de las búsquedas, si contará con una pequeña base de datos en la que estarán almacenados cada uno de los recursos de información en los que se puede buscar y las características de los mismos.

Atendiendo a los componentes definidos anteriormente se realizó el siguiente diseño relacional de Base de Datos en MySQL.



Como se puede apreciar se ha creado una tabla para cada uno de los componentes, cuyos campos no son más que los atributos definidos anteriormente para los mismo. Las tablas *info_sources* y *adaptors* establecen relación de llave foránea con *search_engines*.

III. Implementación del Servicio

Para la correcta implementación del Servicio de Búsqueda se escogió el lenguaje de desarrollo de aplicaciones *Web PHP* en su versión 5.0, y además se hace uso de *Smarty* como sistema de templates que facilita la implementación modular separando el contenido de la presentación, mayormente usado para las interfaces de Administración. Se hace uso de *PEAR* con el objetivo de hacer los componentes de la aplicación más reutilizables y como lenguaje gestor de Bases de Datos se usa *MySQL*.

3.1 Implementación de los componentes fundamentales

Para cada uno de los componentes se definió una clase que encapsula las características y atributos fundamentales de cada objeto. Y además para la administración de los mismos, o sea, las operaciones de inserción, modificación y eliminación de los mismos de la BD se definieron los Manejadores (*managers*).

3.1.1 Implementación de los Buscadores (*Search Engines*)

Clase *SearchEngine*

```
class SearchEngine
{
    private $name, $description, $params, $implementation, $class_name, $id;

    function __construct($name,$description,$params,...){ ..... }

    //properties
    public function getId() { return $this->id; }

    public function getName() { return $this->name; }
    public function setName($value) { $this->name = $value; }

    public function getDescription() { return $this->description; }
    public function setDescription($value){ $this->description = $value; }

    public function getParams() {return $this->params;}
    public function setParams($value) { $this->params = $value; }

    public function getImplementation() { return $this->implementation; }
    public function setImplementation($value){$this->implementation=$value;}

    public function getClassName() {return $this->class_name;}
    public function setClassName($value) { $this->class_name = $value; }

}
```

Clase *SearchEngineManager*

De modo general en esta clase se implementan todas las funcionalidades que permiten el manejo y la administración de los Buscadores. La mayoría de las funciones de esta clase acceden a la tabla *search_engines* de la Base de Datos.

Las funciones definidas son:

- *Function get(\$id)*: Recibe un identificador de Buscador y accede a la BD para obtener todos los atributos relacionados con el mismo y devolver un objeto SearchEngine.
- *Function getAll()*: Devuelve un arreglo con objetos SearchEngine que representan todos los Buscadores que se tienen registrados en la BD.
- *Function update(\$search_engine, \$data)*: Dado un Buscador y un arreglo con valores, actualiza en la BD los columnas del Buscador con dichos valores.

```
public function update($search_engine, $data)
{
    $search_engine -> setName($data['name']);
    $search_engine -> setDescription($data['description']);
    $search_engine -> setParams($data['params']);
    $search_engine -> setImplementation($data['implementation']);
    $search_engine -> setClassName($data['class_name']);
}
```

- *Function save(\$search_engine)*: Salva el Buscador recibido como parámetro de la función en la BD.
- *Function delete(\$id)*: Elimina un Buscador de la BD dado un id del mismo.
- *Function validate(\$engine)*: Para validar los valores dados a los atributos de un Buscador antes de insertarlos a la BD, de este modo se evita que se inserten datos inconsistentes.

Ambas clases se encuentran implementadas en el fichero *search_engines.lib.php*.

Camino completo: Buscador/libs/search_engines.lib.php

3.1.2 Implementación de las Fuentes de Información (Info Sources)

Clase *InfoSourceEngine*

```

class InfoSourceEngine
{
    private $name, $description, $search_engine_id, $params, $id;

    function __construct($name,$description,$search_engine_id,$params,$id){...}

    public function getId() { return $this->id; }

    public function getName() { return $this->name;}
    public function setName($value) {$this->name = $value;}

    public function getDescription() { return $this->description;}
    public function setDescription($value) {$this->description = $value;}

    public function getSearchEngineParams() { return $this->params;}
    public function setSearchEngineParams($value) {$this->params = $value;}

    public function getSearchEngineId() {return $this->search_engine_id;}
    public function setSearchEngineId($value) {$this->search_engine_id = $value;}
}

```

Clase *InfoSourceEngineManager*

Esta clase tiene las mismas funciones que la clase *SearchEngineManager*, lo que lógicamente para manejar y administrar las Fuentes de Información. Por lo que estas funciones acceden a la tabla *info_sources* de la Base de Datos.

Además de los métodos descritos, en esta clase se implementa el método *getSearchEngineParamsArray(\$id)* que permite obtener los parámetros correspondientes al buscador asociado a una fuente de información, así como los valores de los mismos.

Ambas clases se encuentran implementadas en el fichero *infoSource_engines.lib.php*.

Camino completo: Buscador/libs/infoSource_engines.lib.php

3.1.3 Implementación de los Adaptadores (Adaptors)

Clase *AdaptorEngine*

```

class AdaptorEngine
{
    private $name, $description, $file, $search_engine_id, $id;

    function __construct($name,$description,$file,$search_engine_id,$id) {...}

    public function getId() {return $this->id;}

    public function getName() { return $this->name;}
    public function setName($value) { $this->name = $value;}

    public function getDescription() { return $this->description;}
    public function setDescription($value) { $this->description = $value;}

    public function getFile() { return $this->file; }
    public function setFile($value) { $this->file = $value;}

    public function getBrowserId() {return $this->search_engine_id;}
    public function setBrowserId($value) { $this->search_engine_id = $value;}
}

```

Clase *AdaptorEngineManager*

Esta clase contiene los mismos métodos que la clase *SearchEngineManager*, lo que para manejar y administrar los Adaptadores. Las funciones de la misma acceden a la tabla *adaptors* de la Base de Datos.

Ambas clases se encuentran implementadas en el fichero *adaptors_engines.lib.php*.

Camino completo: Buscador/libs/ adaptors_engines.lib.php

3.2 Implementación de la Función de búsqueda (search)

Para cada uno de los buscadores (search engines) que se vayan definiendo para el servicio de búsqueda se debe implementar una función de búsqueda (search), la cual será la encargada de interpretar (*parsear*) la consulta introducida y devolver los resultados obtenidos.

Con el fin de lograr cierta homogeneidad en el modo de programación y no romper con el diseño orientado a objetos seguido hasta el momento, se decidió definir una *interfaz*, que contiene el método *search*. Dicha interfaz deberá ser implementada por cada uno de los buscadores que se definan.

```

interface SearchInterface
{
    public function search($query, $count_elem, $init_elem, $parameters);
}

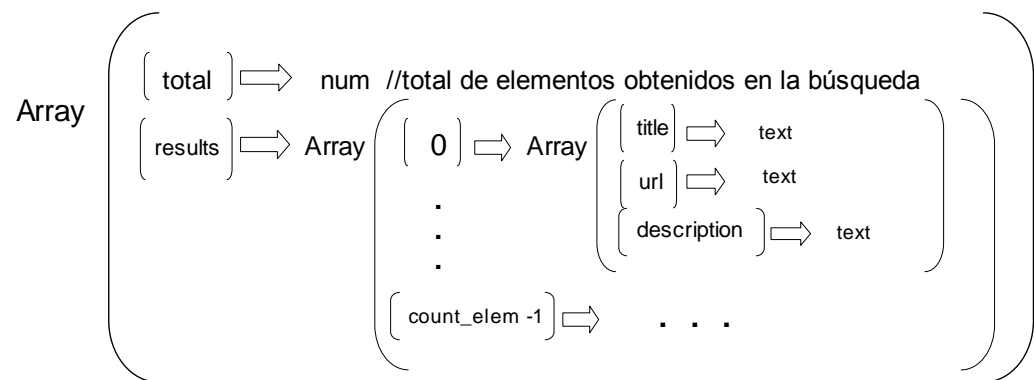
```

Camino completo: Buscador/libs/search_interface.php

Como se puede apreciar en el recuadro anterior el método search recibe los siguientes parámetros:

- *\$query*: Consulta introducida por el usuario o la aplicación cliente que esté usando el servicio.
- *\$count_elem*: Cantidad de elementos a devolver.
- *\$init_elem*: Elemento inicial a partir de cual se quiere empezar a buscar (para el paginado).
- *\$parameters*: Parámetros de filtro del buscador y sus valores.

De modo general el método *search* devolverá un arreglo asociativo con la siguiente estructura:



La implementación de esta función *search* para cada uno de los buscadores que se vayan registrando en el servicio, no será más que la propia implementación un cliente webservice.

Debido a que las inmensa mayoría de los recursos de información que se han necesitado añadir al servicio brindaban webservice XML-RPC o SOAP, este tipo de clientes son los que se han implementados. Las características y un ejemplo de cada uno se describen a continuación.

3.2.1 Implementación de un cliente XML-RPC. Búsqueda en el Lis.

Un poco de XML-RPC

XML-RPC es un protocolo de llamada a procedimientos remotos que usa XML para codificar los datos y HTTP como protocolo de transmisión de mensajes.

Es un protocolo muy simple ya que sólo define unos cuantos tipos de datos y comandos útiles, además de una descripción completa de corta extensión. La simplicidad del XML-RPC está en contraste con la mayoría de protocolos RPC que tienen una documentación extensa y requiere considerable soporte de software para su uso.

XML-RPC fue evolucionando hasta que se convirtió en lo actualmente conocido como **SOAP**. Una diferencia fundamental es que en los procedimientos en SOAP los parámetros tienen nombre y no interesan su orden, no siendo así en XML-RPC.

Por la simplicidad de este estándar aún en nuestros días es bastante usado. Algunos de los recursos de información que usa Infomed, sobre todo, los más antiguos tienen sus webservices implementados haciendo uso de XML-RPC, entre estos se destacan los que tienen bases de datos ISIS como las bases de datos bibliográficas (Lilacs, Medline, etc) y los de las bases de datos del CWIS como el LIS.

Ejemplo de implementación de cliente LisCuba:

De modo general, el Localizador de Información en Salud (LIS) es el portal de la Biblioteca Virtual de Salud que contiene el catálogo de fuentes de Información en salud disponible en Internet y seleccionada según criterios de calidad. Describe el contenido de estas fuentes y ofrece enlace con las mismas en Internet.

En el contexto del servicio de búsqueda que se implementa, se definió el buscador para LISCuba. El buscador LISCuba hace uso del WebService de Búsqueda del CWIS para el Lis Cuba <http://liscuba.sld.cu>. Este WebService se encuentra implementado en php y haciendo uso del protocolo *XMLRPC*. Entre las funciones que brinda dicho WebService es de nuestro interés la función *liscuba.buscaravanzado*.

La función *liscuba.buscaravanzado* permite realizar búsquedas avanzadas filtrando un conjunto de campos definidos:

- text -> string
- publisher -> int
- creator -> int
- contributor -> int
- subject -> int
- type -> int
- language -> int
- audience -> int
- format -> int
- classification -> int
- start -> int
- limit -> int
- bydate -> string

Devuelve un arreglo con estructura con el total de registros recuperados y los registros con sus datos.

Implementación:

Se implementó la clase *LisCubaSearchEngine* que implementa *SearchInterface*. La sintaxis general de esta clase y las funciones definidas en la misma se describen a continuación.

```

class LisCubaSearchEngine implements SearchInterface
{
    public function search($query, $count_elem, $init_elem, $parameters)
    {
        $path= 'server/server.php';
        $server= 'liscuba.sld.cu';
        $port = 80;
        $debug=0;

        $client =& new xmlrpc_client($path, $server, $port);
        $client->return_type = 'xmlrpcvals';
        ....
        $client->setDebug($debug);
        //método del server
        $metodo = 'liscuba.buscaravanzado';
        $msg =& new xmlrpcmsg($metodo);
        .... //parameters ej
        $text =& new xmlrpcval($text, 'string');
            $msg->addparam($text);
        ....
        $res =& $client->send($msg, 0, "");

        if (!$res->faultcode())
        { .....
            //conformar arreglo de resultados
        }
        return array("total" => $total, "results" => $results);
    }
    else print "<p>XML-RPC Fault #" . $res->faultCode() ; .....
}
}

```

- *Function search(\$query, \$count_elem, \$init_elem, \$parameters)*: La implementación de esta función consiste en la creación de un cliente *XMLRPC* que recibe como argumento el camino al servidor de Webservices de Infomed. Se invoca el método *liscuba.buscaravanzado* pasándole todos los parámetros requeridos serializados. El resultado obtenido se procesa y se devuelve un arreglo de resultados con la estructura necesario.

3.2.2 Implementación de un cliente SOAP. Búsqueda en Base de Datos bibliográficas.

Un poco de **SOAP**:

SOAP (siglas de *Simple Object Access Protocol*) es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML. Este protocolo deriva de XML-RPC. SOAP Es uno de los protocolos utilizados en los servicios Web.

Ejemplo de implementación de cliente para buscar en bases de datos bibliográficas:

Para ejemplificar un cliente SOAP, veremos la implementación de un cliente para buscar en las bases de datos bibliográficas que se encuentran en ISIS entre ellas Lilacs, Medline y otras.

Se definió el buscador Bibliographic, el cual hace uso del Webservice de Búsqueda en bases de datos ISIS, cuya descripción se encuentra en <http://bases.bvs.br/webservices/index.php?wsdl>. Este Webservice se encuentra implementado en php y haciendo uso del protocolo SOAP. Entre las funciones que brinda dicho Webservice es de nuestro interés la función *search*.

La función *search* permite realizar búsquedas avanzadas filtrando un conjunto de campos definidos:

- collection -> la colección en la que se desea buscar (Lilacs, Medline, Adolec).
- expression -> consulta a buscar.
- from -> elemento por el cual se desea empezar a buscar.
- count -> cantidad de elementos a devolver.
- lang -> lenguaje.

Implementación:

Se implementó la clase *BibliographicSearchEngine* que implementa *SearchInterface*. La sintaxis general de esta clase y las funciones definidas en la misma se describen a continuación.

```
class BibliographicSearchEngine implements SearchInterface
{
    public function search($query, $count_elem, $init_elem, $parameters) {...}
    //ver codigo
}
```

- *Function search(\$query, \$count_elem, \$init_elem, \$parameters)*: La implementación de esta función consiste en la creación de un cliente SOAP que recibe como argumento el camino al servidor de Webservices de Infomed. Se invoca el método *search* pasándole todos los parámetros requeridos serializados. El resultado obtenido se procesa y se devuelve un arreglo de resultados con la estructura necesario.

3.3 Implementación de la Función para adaptar formatos (adaptTo)

Cuando se desee tener una función que adapte los resultados de la salida estándar a otro formato, se implementa un adaptador y la función (adaptTo), la cual será la encargada de adaptar del formato de salida estándar a otro formato especificado y devolver los resultados en este nuevo formato.

Igual que con los buscadores se decidió definir una *interfaz*, que contiene el método *adaptTo*. Dicha interfaz deberá ser implementada por cada uno de los adaptadores que se definan.

```
interface AdaptorInterface
{
    public function adaptTo($format, $results, $name, $description, $link,
        $path);
}
```

Camino completo: Buscador/libs/adaptors_interface.php

Como se puede apreciar en el recuadro anterior el método recibe los siguientes parámetros:

- *\$format*: Formato al cual se quieren adaptar los resultados.
- *\$results*: Arreglo de resultados que devuelve el buscador asociado, los cuales se quieren adaptar.
- *\$name*: Nombre del adaptador.
- *\$description*: Descripción del adaptador.
- *\$link*: Url del servicio de búsqueda
Ej. http://buscador.sld.cu/main_search.php.
- *\$path*: Camino físico donde se desea guardar el fichero con los resultados adaptados al nuevo formato (en caso de que se salven en un fichero Ej. "C:/xampp/htdocs/Buscador/modulos/adaptadores/RSS")

De modo general esta función deberá devolver el fichero salvado con los resultados adaptados al formato deseado.

3.4 WebService del Servicio de Búsqueda

El Servicio de Búsqueda de Infomed, concebido como un servicio independiente brindaría una interfaz de WebService para que otras aplicaciones hagan uso del mismo.

El WebService se implementó haciendo uso de la extensión SOAP de php5, la decisión de usar esta versión en vez de alguna de las otras variantes de SOAP para php, díganse NuSoap o PEAR SOAP, se debió fundamentalmente al hecho de que es muy fácil de mantener y al estar escrita en C es mucho más rápida que las otras versiones, que están escritas 100% en PHP. En el tipo de servicio que estamos tratando, que será muy usado desde diversas aplicaciones, garantizar la rapidez es una de las principales premisas a tener en cuenta a la hora de implementar.

3.4.1 Generación del archivo descriptivo del WebService (WSDL)

Para describir las posibilidades y funcionalidades que ofrece el Servicio de Búsqueda como servicio Web se implementó un archivo WSDL. La creación del mismo se realizó usando la librería NuSoap, debido a que esta ofrece un conjunto de opciones y métodos que facilitan la correcta y sencilla implementación de este tipo de archivos XML, que suelen ser algo tediosos.

A continuación se muestra la vista de la página de descripción del servicio, donde a la izquierda se pueden ver todos los métodos que ofrece el mismo. A la derecha se describe la función seleccionada, en este caso se encuentra desplegada la función *search*. <http://buscador.sld.cu/> es la dirección.

SearchService

View the [WSDL](#) for the service. Click on an option to view it's details.

- [search](#)
- [listInfoSources](#)
- [listSearchEngines](#)
- [listAdaptorEngines](#)
- [advancedSearch](#)
- [adaptTo](#)

[Close](#)

Name: search
Binding: SearchServiceBinding
Endpoint: http://buscador.sld.cu/websevice_soap/server.php
SoapAction: urn:Search_Service#search
Style: rpc
Input:
 use: encoded
 namespace: urn:Search_Service
 encodingStyle: http://schemas.xmlsoap.org/soap/encoding/
 message: searchRequest
 parts:
 query: xsd:string
 infoS_id: xsd:int
 count_elem: xsd:int
 init_elem: xsd:int
Output:
 use: encoded
 namespace: urn:Search_Service
 encodingStyle: http://schemas.xmlsoap.org/soap/encoding/
 message: searchResponse
 parts:
 return: ArrayOfSearchResult
Namespace: urn:Search_Service
Transport: http://schemas.xmlsoap.org/soap/http
Documentation: Search information in an information source

La construcción del WSDL se encuentra en el fichero *index.php*.

El archivo XML que constituye el WSDL completo puede observarse al dar click sobre el hipervínculo [WSDL](#).

Las seis funciones que se observan a la izquierda de la imagen, son las que se encuentran actualmente registradas e implementadas en el webservice. Si se deseara incluir una nueva función, el primer paso sería incluirla en el archivo *index.php* haciendo uso de la sintaxis *nusoap* (guiarse por las anteriores funciones registradas).

3.4.2 Implementación del Servidor (Server)

Luego de tener la descripción del servicio en el archivo WSDL, el próximo paso fue implementar el servidor (server). El servidor se implementó haciendo uso de la extensión SOAP de PHP5.

En este paso se escribieron los métodos como funciones php, y se añadieron los mismos al servidor. A continuación se muestra un fragmento de código. El código completo con la implementación detallada de cada uno de los métodos se encuentra en el fichero *server.php*.

Camino completo: Buscador/webservice_soap/server.php

```
ini_set('display_errors', '0');
ini_set('soap.wsdl_cache_enabled', '0'); // disabling WSDL cache

//require section //
.....
// Define the method as a PHP function
function search($query,$infoS_id,$count_elem,$init_elem) {...}
function listInfoSources() { ....}
function listSearchEngines(){ ....}
function listAdaptorEngines() { ....}
advancedSearch($query, $searchE_id, $count_elem, $init_elem, $parameters) {...}
adaptTo($format, $infoS_id, $results, $link, $path) {...}

$server = new
SoapServer($wsdl_path/*'http://localhost/Buscador/index.php?wsdl'*/);
$server->addFunction('search');
$server->addFunction('listInfoSources');
$server->addFunction('listSearchEngines');
$server->addFunction('listAdaptorEngines');
$server->addFunction('advancedSearch');
$server->addFunction('adaptTo');
$server->handle();
```

IV. Administración del Servicio de Búsqueda

La administración constituye unos de los elementos más importantes en el Servicio de Búsqueda, mediante la misma se manejan las acciones de adición, modificación o eliminación de los componentes esenciales del mismo, es por ello que se ha dedicado un capítulo a la descripción del proceso de administración para este.

4.1 Administración de Buscadores (Search Engines)

Los Buscadores (search engines) constituyen el componente esencial del servicio, pues son ellos los que tienen la lógica de búsqueda en su archivo de implementación.

4.1.1 Adición de un nuevo Buscador (Add Engine)

Una vez que se decide incluir un nuevo buscador en el servicio, la primera acción sería estudiar el buscador, identificar sus métodos de búsqueda y los posibles parámetros que recibe, así como los tipos de datos y los valores de los mismos. Luego de ello, ya se pueden seguir en orden lógico los siguientes **pasos**:

1. *Crear un fichero de configuración para el buscador (config.xml) (veremos más adelante las características de este xml).*
2. *Implementar el cliente de búsqueda para dicho buscador (este es el fichero php que implementa la interfaz SearchInterface, como vimos en el capítulo anterior).*
3. *Probar el cliente pasándole un juego de datos (parámetros con valores) para asegurarnos de que está funcionando correctamente.*
4. *Crear una carpeta con el nombre del buscador, dentro de ella guardar ambos ficheros (o sea el config.xml y el fichero de implementación) y compactar esta carpeta en **ZIP** (no en ningún otro formato).*
5. *Acceder a la interfaz de Administración de buscadores: <http://buscador.sld.cu/admin/buscadores/index.php> (en esta se observa el listado de los buscadores que ya están registrados en el sistema).*
6. *Dar click en el botón **AddEngine** y se pasa a la interfaz de adición.*
7. *Una vez en la interfaz de adición se deben introducir los datos del buscador que se desea crear (name, description) y en **Upload SearchEngine Files** cargar el archivo ZIP que habíamos creado anteriormente. Luego dar click en el botón **Save**.*

Luego del paso 7, se pasa automáticamente a la interfaz inicial que es la que muestra la *lista de los buscadores*. Una vez en esta, debe asegurarse de que el nuevo buscador aparezca listado, pues esto es indicador de que se creó correctamente.



Interfaz de Administración de Buscadores

4.1.2 Confección del archivo de configuración (config.xml)

Para cada buscador se debe definir un archivo de configuración (config.xml). Esta solución fue la vía escogida para registrar principalmente los parámetros de los buscadores y los valores asociados a los mismos, en caso de que tengan. Además, este archivo xml se aprovechará para registrar otros datos asociados al buscador como lo son el archivo de implementación y el nombre de la clase que implementa la interfaz SearchInterface.

Ejemplo de archivo de config.xml:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<configuration>

  <implementation>lis_search_engine.php</implementation>
  <classname>LisSearchEngine</classname>

  <parameters>

    <param>
      <name>custom</name>
      <type>text</type>
    </param>

    <param>
      <name>options</name>
      <type>lis</type>
      <options>
        <label value="1">Titulo</label>
        <label value="2">Materia</label>
        <label value="3">Resumen</label>
        <label value="4">Todos los campos</label>
      </options>
    </param>

  </parameters>

</configuration>
```

4.1.3 Modificación o eliminación de Buscadores

Los buscadores ya registrados en el servicio pueden ser modificados o eliminados.

Modificar:

Para modificar un buscador basta con dar click sobre el nombre del mismo que aparece listando en <http://buscador.sld.cu/admin/buscadores/index.php>. Y se pasa a una interfaz igual a la de Adición pero donde los valores de los campos están llenos para que puedan ser modificados. Luego de realizar los cambios, por supuesto, se debe dar click en el botón *Save*.

Eliminar:

Para eliminar un buscador existente se debe marcar el check box que se encuentra al lado del elemento que se desea eliminar, el cual debe aparecer listando en <http://buscador.sld.cu/admin/buscadores/index.php>. Y se hace click en el botón *Delete*. Inmediatamente el elemento (o sea el buscador) debe desaparecer del listado.

4.2 Administración de Fuentes de Información (Info Sources)

Los Buscadores por si solos no satisfacen las necesidades del Servicio de Búsqueda, ellos sirven de base para la búsqueda simple, pero de cara al usuario es necesario crear Fuentes de Información y asociarlas a los mismos. Es por ellos que son de gran importancia las pantallas de administración de Fuentes de Información.

4.2.1 Adición de una nueva Fuente de Información (Add InfoSource)

Antes de adicionar una nueva fuente de información se debe saber de antemano si ya está implementado y adicionado al servicio, el buscador con el que se asociará la misma. Para ello se puede consultar el listado de buscadores en <http://buscador.sld.cu/admin/buscadores/index.php>. De no estar creado el buscador, esto es lo primero que se debe hacer siguiendo los pasos del anterior epígrafe.

Suponiendo que ya esté incluido en el servicio el buscador, entonces para la adición de una fuente de información se deben seguir en los siguientes **pasos**:

1. *Acceder a la interfaz de Administración de fuentes de información: http://buscador.sld.cu/admin/fi/index_finfo.php (en esta se observa el listado de las fuentes de información que ya están registradas en el sistema).*
2. *Dar click en el botón **Add** y se pasa a la interfaz de adición.*
3. *Una vez en la interfaz de adición se deben introducir los datos de la fuente de información que se desea crear (name, description).*
4. *Se debe escoger en el listado de los buscadores, el buscador al cual se desea asociar la fuente. Y posteriormente ponerle valores a los*

*parámetros del mismo que se deseen filtrar. Luego dar click en el botón **Save**.*

Luego del paso 4, se pasa automáticamente a la interfaz inicial que es la que muestra la *lista de las fuentes de información*. Una vez en esta, debe asegurarse de que la nueva fuente aparezca listada, pues esto es indicador de que se creó correctamente.

4.2.2 Modificación o eliminación de Fuentes de Información

Las fuentes de información ya registradas en el servicio pueden ser modificadas o eliminadas.

Modificar:

Para modificar una fuente de información basta con dar click sobre el nombre de la misma que aparece listada en http://buscador.sld.cu/admin/fi/index_finfo.php. Y se pasa a una interfaz igual a la de Adición pero donde los valores de los campos están llenos para que puedan ser modificados. Luego de realizar los cambios, por supuesto, se debe dar click en el botón *Save*.

Eliminar:

Para eliminar una fuente de información existente se debe marcar el check box que se encuentra al lado del elemento que se desea eliminar, el cual debe aparecer listando en http://buscador.sld.cu/admin/fi/index_finfo.php. Y se hace click en el botón *Delete*. Inmediatamente el elemento (o sea la fuente de información) debe desaparecer del listado.

El diseño de las pantallas de administración de Fuentes de Información, es similar al de las pantallas de administración de buscadores.

4.3 Administración de Adaptadores (Adaptors)

Los Adaptadores son objetos que como mencionamos, tienen la funcionalidad a adaptar del formato de salida por defecto a otro formato especificado. Estos componentes también tienen sus interfaces de Administración.

4.3.1 Adición de un nuevo Adaptador (Add Adaptor)

Una vez que se decide incluir un nuevo adaptador en el servicio, se deben seguir los siguientes **pasos**:

1. *Implementar el Adaptador, o sea crear al fichero php que contiene la clase que implementa la interfaz `AdaptorInterface` y la función `adaptTo` (como vimos anteriormente).*
2. *Probar la función `adaptTo` pasándole un juego de datos para asegurarnos de que está funcionando correctamente.*
3. *Acceder a la interfaz de Administración de adaptadores: http://buscador.sld.cu/admin/adaptadores/index_adaptors.php (en esta se observa el listado de los adaptadores que ya están registrados en el sistema).*

4. Dar click en el botón **Add** y se pasa a la interfaz de adición.
5. Una vez en la interfaz de adición se deben introducir los datos del adaptador que se desea crear (name, description).
6. Se debe escoger en el listado de los buscadores, el buscador al cual se desea asociar este adaptador.
7. Se debe escoger del listado de files el fichero de implementación del adaptador y escribir en el textbox *Class* el nombre de la clase implementada. Luego dar click en el botón **Save**.

Luego del paso 7, se pasa automáticamente a la interfaz inicial que es la que muestra la *lista de los adaptadores*. Una vez en esta, debe asegurarse de que el adaptador aparezca listado, pues esto es indicador de que se creó correctamente.

4.3.2 Modificación o eliminación de Adaptadores

Los adaptadores ya registrados en el servicio pueden ser modificados o eliminados.

Modificar:

Para modificar un adaptador basta con dar click sobre el nombre del mismo que aparece listando en http://buscador.sld.cu/admin/adaptadores/index_adaptors.php. Y se pasa a una interfaz igual a la de Adición pero donde los valores de los campos están llenos para que puedan ser modificados. Luego de realizar los cambios, por supuesto, se debe dar click en el botón *Save*.

Eliminar:

Para eliminar un adaptador existente se debe marcar el check box que se encuentra al lado del elemento que se desea eliminar, el cual debe aparecer listando en http://buscador.sld.cu/admin/adaptadores/index_adaptors.php. Y se hace click en el botón *Delete*. Inmediatamente el elemento (o sea el adaptador) debe desaparecer del listado.

El diseño de las pantallas de administración de Adaptadores, es similar al de las pantallas de administración de buscadores.

V. Librerías útiles implementadas

En el presente Capítulo se describen un conjunto de librerías implementadas, las cuales son de gran utilidad para la implementación de diversas partes del Servicio de Búsqueda.

5.1 Librería para parsear XML

Como se mencionó en el Capítulo II, donde se definen los elementos esenciales para la implementación del Servicio, los parámetros de los Buscadores serán representados con un XML (cuya estructura se describió también en el Cap. II). Recordando del Capítulo II, teníamos dos tipos de XML:

- XML de parámetros del Buscador: Representan los parámetros para filtrar la búsqueda sobre determinado buscador (quedan finalmente guardados en la tabla *searchengines*).

```
<?xml version="1.0"?>
<parameters>
  <param>
    <name>..... </name>
    <type> ..... </type>
  </param>
  .....
</parameters>
```

En este XML si el parámetro es de tipo lista (*list*) significa que existe más de una opción para restringir este parámetro, por lo que el documento XML se extiende de la siguiente forma:

```
<type> ..... </type>
<options>
  <label value = "... "> ..... </label>
  .....
</options>
```

- XML de parámetros del Buscador con sus valores: Representan los valores que le son restringidos a un determinado buscador, dada la fuente de información asociada el mismo (quedan finalmente guardados en la tabla *infosources*).

```
<?xml version="1.0"?>
<parameters>
<param name="..."> value </param>
.....
</parameters>
```

Evidentemente se hace necesario implementar funciones que permitan crear y leer estructuras XML como las anteriores. Con este fin se implementó la clase *XMLParamParser*, cuyos métodos usan la librería Document Object Model (DOM), la cual se encuentra compilada en algunas versiones de PHP. Dicha librería permite manipular archivos XML de modo fácil y óptimo.

```
class XMLParamParser
{
    //para el XML de parámetros del Buscador
    function readXMLParam($myxml) { .... }
    function writeXMLParam($array_params) { .... }

    //para el XML de parámetros del Buscador con sus valores
    function readXMLParamValor($myxml) { .... }
    function writeXMLParamValor($buscador_params) { .... }
}
```

Las funciones de la clase *XMLParamParser* que leen (*read*) XML reciben como parámetro de entrada un documento XML y devuelven un arreglo con cada uno de los elementos.

Las funciones de dicha clase que escriben (*write*) o generan XML reciben como parámetro de entrada el arreglo de parámetros del buscador y sus valores (en caso de ser necesario) y devuelven el documento XML con la estructura deseada.

5.1.1 Leyendo XML con DOM

La librería DOM es muy eficiente para la lectura de archivos XML debido a que lee el documento completo y lo almacena en memoria como un árbol, cuyos nodos son las etiquetas XML.

El primer paso para leer el archivo XML es crear un nuevo objeto *DOMDocument* y cargar el XML que se desea leer usando el método *LoadXML*. Posteriormente para obtener la lista de todos los elementos con un nombre determinado se usa la función *getElementsByTagName*. Con un ciclo se va recorriendo el árbol de elementos XML hasta llegar a los nodos hojas, a los cuales se les piden sus valores o sus atributos.

En el siguiente cuadro se muestra la implementación de la función `readXMLParam($myxml)`:

```
function readXMLParam($myxml)
{
    $doc = new DOMDocument();
    $doc->loadXML($myxml);
    $param_list = $doc->getElementsByTagName( "param" );
    $parameters = array();

    foreach( $param_list as $param )
    {
        $name = $param->getElementsByTagName( "name" )->item(0);
        $type = $param->getElementsByTagName( "type" )->item(0);

        $param_array= array();
        $param_array[0]= $name->nodeValue;
        $param_array[1]= $type->nodeValue;

        if($type->nodeValue=="list")
        {
            $option= $param->getElementsByTagName("options")->item(0);
            $label_list= $option->getElementsByTagName("label");

            $options_array=array();
            foreach($label_list as $label)
            {
                $label_key= $label->getAttribute('value');
                $label_name=$label->nodeValue;
                $options_array[$label_key]= $label_name;
            }
            $param_array[2]=$options_array;
        }
        $parameters []= $param_array;
    }
    return $parameters;
}
```

5.1.2 Generando XML con DOM

Para generar o escribir documentos XML usando la librería DOM, la idea es crear un *DOMDocument* vacío e irle añadiendo cada uno de los elementos y sus hijos formando el árbol de etiquetas. Para esto se usan los métodos *createElement* y *appendChild*. Para crear los nodos hojas y setear sus valores se usa *createTextNode* y *setAttribute*. Finalmente el objeto *DOMDocument* se salva a XML usando la función *saveXML()*.

En el siguiente cuadro se muestra la implementación de la función `writeXMLParam($myxml)`:

```

function writeXMLParam($array_params)
{
    $doc = new DOMDocument();
    $doc->formatOutput = true;

    $ps = $doc->createElement( "parameters" );
    $doc->appendChild( $ps );

    foreach($array_params as $param )
    {
        $p = $doc->createElement("param");

        //creando tag de name
        $name = $doc->createElement("name");
        $name->appendChild($doc->createTextNode($param[0]));//name

        $p->appendChild( $name );

        //creando tag de type igual que el de name pero en $param[1]

        if(count($param)==3) //el type es list
        {
            //creando tag de options
            $options = $doc->createElement( "options" );
            //cogiendo las llaves del arreglo de opciones,se encuentra en la pos 2
            $options_labels= array_keys($param[2]);

            foreach($options_labels as $label)
            {
                //creando los tag de label
                $l = $doc->createElement( "label" );
                $l->appendChild($doc->createTextNode($param[2][$label]));
                $l->setAttribute('value',$label);

                $options->appendChild($l);
            }
            $p->appendChild( $options );
        }
        $ps->appendChild( $p);
    }
    return $doc->saveXML();
}

```

VI. Implementación de OpenSearch

OpenSearch es un estándar para la publicación de resultados de una búsqueda en un formato adecuado para la sindicación y agregación. Es una forma de que las páginas web y los motores de búsqueda publiquen sus resultados de forma accesible. Es un protocolo creado por a9.com (Amazon) y actualmente es usado por muchos navegadores (principalmente Internet Explorer 7 y Firefox 2) en sus buscadores.

Algunas definiciones:

- *Sindicación*: redifusión de contenidos desde una fuente original a otro sitio web, es decir, poner contenidos a disposición de terceros, generalmente de forma gratuita. Es posible, por ejemplo gracias al formato RSS, que contiene información altamente estructurada.
- *Agregación*: consiste en "absorber" de fuentes de contenidos externas los archivos RSS para presentarlos en nuestro sitio. Es posible gracias a pequeñas aplicaciones, llamadas agregadores o lectores de feeds, que "leen" esos archivos.

6.1. Características de OpenSearch. Implementación

OpenSearch consiste en:

- *Archivos descriptivos OpenSearch (en inglés OpenSearch Description Documents)*: Archivos en formato XML que identifican y describen un motor de búsqueda. Contiene mediante un formato particular (OpenSearch Query Syntax) cómo obtener los resultados de una búsqueda.
- *OpenSearch RSS (en la versión 1.0) o OpenSearch Response (en la 1.1)*: Es un formato que permite abrir los resultados de una búsqueda.
- *Agregadores*: Sitios que pueden mostrar resultados realizados mediante OpenSearch.

Archivos descriptivos:

Example of a simple OpenSearch description document:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<OpenSearchDescription xmlns="http://a9.com/-/spec/opensearch/1.1/">
  <ShortName>Web Search </ShortName>
  <Description>Use Example.com to search the Web.</Description>
  <Tags>example web</Tags>
  <Contact>admin@example.com</Contact>
  <Url type="application/rss+xml"
template="http://example.com/?q={ searchTerms }&pw={ startPage? }&format
=rss"/>
</OpenSearchDescription>
```

Autodescubrimiento:

Example of an HTML document that includes OpenSearch autodiscovery link elements:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head profile="http://a9.com/-/spec/opensearch/1.1/">
<!-- ... -->
<link rel="search"
      type="application/opensearchdescription+xml"
      href="http://example.com/content-search.xml"
      title="Content search" />
<link rel="search"
      type="application/opensearchdescription+xml"
      href="http://example.com/comment-search.xml"
      title="Comments search" />
<!-- ... -->
</head>
<body> <!-- ... --> </body>
</html>
```

Resultados:

Example of a page of search results in the RSS 2.0 format:

```
<?xml version="1.0" encoding="UTF-8"?>
<rss version="2.0" xmlns:opensearch="http://a9.com/-/spec/opensearch/1.1/"
xmlns:atom="http://www.w3.org/2005/Atom">
<channel>
  <title>Example.com Search: New York history</title>
  <link>http://example.com/New+York+history</link>
  <description>Search results for "New York history" at x.com</description>
  <opensearch:totalResults>4230000</opensearch:totalResults>
  <opensearch:startIndex>21</opensearch:startIndex>
  <opensearch:itemsPerPage>10</opensearch:itemsPerPage>
  <atom:link rel="search" type="application/opensearchdescription+xml"
href="http://example.com/opensearchdescription.xml"/>
  <opensearch:Query role="request" searchTerms="New York History"
startPage="1" />
  <item>
    <title>New York History</title>
    <link>http://www.columbia.edu/cu/lweb/eguids/amerihist/nyc.html</link>
    <description>... Harlem.NYC - A virtual tour and information on businesses ... with
historic photos of Columbia's own New York neighborhood ... . ...
    </description>
  </item> </channel> </rss>
```

Consideraciones:

Ventajas:

- Librerías de desarrollo disponibles de distintos lenguajes (ejemplo: PHP, Python).
- Soportado por los principales navegadores (Internet Explorer 7 y Firefox 2).

Limitaciones:

- No provee la especificación de los valores admisibles para un parámetro (solucionable a través de extensiones o namespaces propios).

6.2. OpenSearch en el Servicio de Búsqueda

Se propone brindar OpenSearch en el servicio de búsqueda, o sea, ser proveedor de OpenSearch.

- OpenSearch en el Servicio de Búsqueda (General).
- OpenSearch en cada uno de los buscadores definidos.

Se propone soportar clientes OpenSearch, o sea, interpretar los buscadores que devuelvan dicho formato.

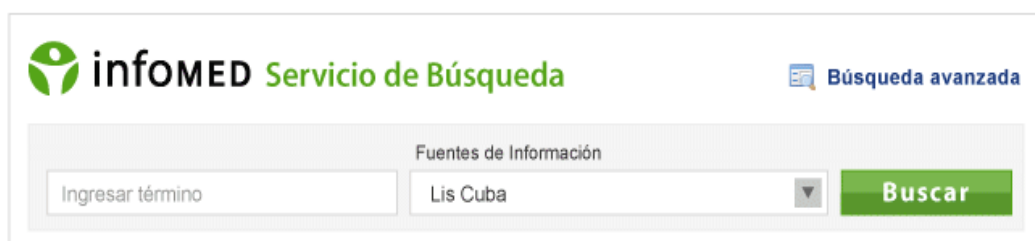
- Se podrán introducir al Servicio de Búsqueda buscadores que soporten OpenSearch.

VII. Estado actual del Servicio de Búsqueda

El Servicio de Búsqueda se encuentra publicado en el servidor Albarrán. Se puede acceder a la descripción del mismo mediante la dirección <http://buscador.sld.cu>.

7.1. Interfaces del Servicio de búsqueda

Para la búsqueda simple se debe acceder a la dirección http://buscador.sld.cu/main_search.php. Cuya interfaz debe ser:



A la interfaz de búsqueda avanzada se puede acceder a través de la de búsqueda simple o directamente accediendo a la url http://buscador.sld.cu/advanced_search.php. Cuya interfaz debe ser:

The screenshot displays the 'infOMED Servicio de Búsqueda' interface. At the top, there's a header with the logo and title. Below it, the 'Búsqueda avanzada' section is highlighted. It includes a search bar with the placeholder 'Ingresar término', a dropdown menu for 'Fuentes de Información' currently set to 'Lis Cuba', and a green 'Buscar' button. Underneath, the 'Searcher Parameters' section is visible, featuring three rows of input fields for 'Idfield 1', 'Idfield 2', and 'Idfield 3', each accompanied by an 'Operator2' dropdown menu. The first row has a dropdown for 'Idfield 1'. The second and third rows have text input fields for 'Expresion 2' and 'Expresion 3' respectively, with dropdowns for 'Idfield 2' and 'Idfield 3'.

7.2. Componentes ya insertados en el Servicio de búsqueda

Actualmente en el sistema (Octubre 2009) están registrados:

➤ *Buscadores (SearchEngines) -> 13*

1. liscuba
2. lisbireme (desactivado temporalmente)
3. medicine
4. statistics
5. aldia
6. infoenlaces
7. directory
8. decs
9. bibliographic
10. events
11. scielo
12. xssearcher (desactivado temporalmente)
13. Blogs

➤ *Fuentes de Información (Info Sources) -> 19*

1. LisCuba
2. Lis Bireme (desactivada temporalmente)
3. Eventos
4. Scielo Cuba
5. Scielo Regional
6. Medicamentos General
7. Medicamentos Reacciones Adversas
8. Medicamentos Principio Activo

9. Medicamentos Indicaciones
10. Anuario Estadístico
11. Al Día
12. Infoenlaces
13. Directorios de Instituciones de Salud
14. Directorio de Centros de Información
15. Directorio de Redes de Información
16. Descriptores en Centros de Salud (Decs)
17. LILACS
18. MEDLINE
19. Portal de Infomed xs (deshabilitado temporalmente)

7.3. Código fuente

El código fuente del Servicio se encuentra en el subversión <http://svn.sld.cu/svn/buscador/trunk/src>, para acceder el mismo se necesita tener permiso y acceso a la dirección anteriormente señalada.

La estructura de carpeta bajo la cual se encuentra implementado el código es la siguiente:

